スクリプト言語で記述されたプログラムを対象にしたソフトウェア難読化手法の提案

2020SE088 中村怜未
指導教員：名倉正剛

1　はじめに

スクリプト言語は開発者が記述したコードがそのまま実行に利用され，ソースコードはそのまま配布されるため，利用者は容易にプログラムを参照できる．このセキュリティ上の課題に対処するため，本研究ではスクリプト言語で記述されたプログラムを対象にしたソフトウェア難読化手法を提案する．

2　ソフトウェア難読化

2.1　ソフトウェア難読化の定義

ソフトウェア難読化とは，入出力の仕様を保ったまま，解析が困難になるように可読性を下げる技術である．ソフトウェア難読化には色々な定義があるが，既存研究 [1] では次のように定義している．

定義：ある言語で書かれたプログラム P と P に関する命題 Q が与えられたとする．そのときに，同一の言語で書かれたプログラム P’ を次の 3 条件を同時に満たすように導くことを，Q に関して P を難読化するという．

- 仕様の保存：任意の入力について，P’ は P と同一の出力を返す．
- 実行効率の保存：任意の入力について，P’ を実行したときの演算，代入，比較などの回数は，P と同じ，または少ない．
- 解析の困難さの増加：P’ は Q に関する解析に P よりも時間がかかる．

2.2　ソースコードに対するソフトウェア難読化手法

ソフトウェア難読化技術は，さまざまなレイヤに対して提案されている．例えば，階層化セキュリティにおいて利用する難読化技術は，コード要素層，ソフトウェアコンポーネント層，インターコンポーネント層，アプリケーション層の 4 つに分類される [2]．ソースコードの難読化に焦点を当てた技術は，このうちのコード要素層に分類され，制御フロー難読化，データ難読化，レイアウト難読化，メソッド難読化，クラス難読化の 5 つの観点からなる．

3　本研究で対象にする課題

スクリプト言語で記述されたプログラムは解読が容易である．その結果，プログラムの構造やアルゴリズム，プログラムが利用するアクセスポイントなどの本来であれば利用者に対して明らかにすべきでない情報の存在も利用者が知ることができる．ソースコード 1 に示したプログラムは，それぞれ 3 行目と 4 行目に記述されたホスト名とポート番号を利用して特定のサーバの特定のポートに対して XML-RPC でアクセスし，情報を登録するプログラムである．このプログラムを解析することで，開発者が意図しないような利用が可能であり，この例の場合はサーバに不正なデータを登録することができる．したがって，ホスト名やポート番号などの情報は利用者に理解できないようにすることが望ましい．

ソースコード 1　スクリプトプログラムの例

```
1 import xmlrpc.client
2
3 server = 'www.xyz.com'
4 port = 23456
5
6 with xmlrpc.client.ServerProxy('http://' +
      server + ':' + str(port) + '/') as
      proxy:
7     import socket
8
9     proxy.saveAddress(socket.gethostbyname(
          socket.gethostname()))
```

4　提案手法

本研究では，スクリプト言語で記述されたプログラムを対象にした難読化手法を提案する．2.2 節で述べたソースコードを対象にした難読化技術の観点のうち，クラス難読化については，オブジェクト指向言語でないと適用できないが，スクリプト言語では必ずしもオブジェクト指向でない記述も可能である．またメソッド難読化については，メソッドが適切に設定されているようなある程度大きなプログラムでないと適用できない．そこで本研究では，それらを除いた 3 つの観点で難読化を実現する手法を提案する．

4.1　処理の流れ

提案手法の処理の流れは以下の通りである．

手順 1　制御フロー難読化
出力に影響を与えない偽のコードを追加する．

手順 2　データ難読化
ソースコード中の文字列や数値をメソッド呼び出しに置き換える．

手順 3　レイアウト難読化
コメントとドキュメンテーション文字列，空白行を削

除し，関数名などの識別子を意味を持たない識別子に変換する．

手順 1 で追加されたコードに文字列や数値が含まれる場合は，その部分も手順 2 のデータ難読化の対象として難読化を実施する．そして手順 2 までの結果として生成された関数名を含め，プログラム内の識別子を手順 3 のレイアウト難読化の対象として難読化を実施する．

なお，これらの手順の実施前後では，プログラムの入出力を保持するようにする．

4.2 制御フロー難読化

制御フロー難読化は，プログラムの複雑さを増加するためにコードの制御構造を変換する [2]．提案手法では偽のコードの追加をすることで実現する．実際に実行されるコードの中に偽のコードが存在していると，コードの存在場所の特定が困難となり，プログラムの解析に時間を要する．出力に影響を与えないように，元のコードに存在している変数名を取得し，その変数名に含まれないランダムな文字列を新たな変数名として偽のコードを作成する．

4.3 データ難読化

データ難読化は，整数，文字列，および配列などの一般的なデータ型に焦点を当てている [2]．提案手法ではデータをメソッド呼び出しに置き換える手続き化というアプローチによって実現する．前章で述べたように，ホスト名などの利用者に明らかにすべきでない情報が直接記述されている文字列や数値は隠す必要がある．ソースコードを解析して抽出された文字列と数値は任意のメソッド呼び出しに置き換えられ，生成するメソッドはそれらのデータを数値演算によって生成するプログラムを記述する．文字列は各文字を表す文字コードを 2 乗したものを 16 進数で表し，その数の平方根を算出し，それらを結合したものを返す．数値は 2 乗したものを 16 進数で表し，その数の平方根を算出したものを返す．

4.4 レイアウト難読化

レイアウト難読化は，元の構文を保持しつつ，コードや命令のレイアウトを混乱させる [2]．提案手法ではコメントとドキュメンテーション文字列，空白行を削除し，関数名などの識別子を意味を持たない識別子に変換することで実現する．コメントやドキュメンテーション文字列には処理の内容や開発者の覚書などが記述され，空白行は処理のまとまりであるコードブロックごとに存在しており，これらを削除することで解読を困難にする．

また，関数名などの識別子にはプログラムの仕様を意味するような名前をつけられることが多い（例えば，二数の加算のプログラムで使用される関数名は add である）．したがって，そのような識別子を意味を持たない識別子に変換することで解読の手がかりとならないようにする．識別子の変換では，特定の文字セットを利用し，ランダムに並べた任意の長さの文字列を作成したものを新しい識別子とする．ただし，組込み関数や外部ライブラリメソッドは対象外とする．

5 評価実験

提案手法により難読化したソースコードとしていないソースコードに対して評価実験を行った．用意したプログラムは，AtCoder[*1]に提出されたソースコードから無作為に選択した 20 個である．

それぞれのソースコード対してテストを行い，関数やメソッドが期待通りに動作しているかどうかを確認する．また，実行時間の比較をすることで，ソフトウェア難読化が実行速度に与える影響を検討する．そして，それぞれのソースコードを ChatGPT に提示して得られる説明が同じであるかどうかを評価した．

実験の結果，難読化を実施したすべてのソースコードに対して，ソースコードの振る舞いが変化していないことが確認できた．用紙の都合上，表 1 には実験対象のソースコードのうち 5 個の実行時間を示している．実行時間の比較により，提案手法が実行時間に与える影響は少ないことが確認できた．また，ChatGPT を用いた実験では，対象にした 20 個のプログラムうち，15 個については難読化したソースコードに対して ChatGPT が元のソースコードと同じ説明文を返すことはなかった．ただし，残りの 5 個については呼び出しを行う外部の API が多く，その API 名によって ChatGPT が元の説明文に近い説明を導出できた．

表 1 実行時間の比較

項目名	難読化前	難読化後
Cross Sum	0.017s	0.033s
CP Classes	0.015s	0.019s
Score Sum Queries	0.020s	0.031s
We Used to Sing a Song Together	0.020s	0.024s
Minimum Coins	0.019s	0.020s

6 おわりに

スクリプト言語で記述されたプログラムを対象に，利用者がプログラムを容易に解析できないようにすることを目的とした難読化手法を提案した．評価実験により，提案手法によってプログラムを難読化していることを確認した．

参考文献

[1] 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 D, Vol. J80, No. 1, pp. 644–652, 1997.

[2] Hui Xu, Yangfan Zhou, Jiang Ming, and Michael Lyu. Layered obfuscation: a taxonomy of software obfuscation techniques for layered security. *Cybersecurity*, Vol. 3, No. 9, 2020.

[*1] https://atcoder.jp/?lang=ja