平成29年度　卒業研究報告書

機械学習による
難読化されたプログラムのステルス評価

提出日：平成30 年3 月9 日

熊本高等専門学校　人間情報システム工学科

バラタ

指導教員：神崎 雄一郎 (准教授)

目 次

1　はじめに

商用のソフトウエアは，ライセンスキーや商業的価値のあるアルゴリズムなどのユーザに知られたくない秘密情報を含む場合がある．悪意のあるエンドユーザが物理的にソフトウエアを所有した場合，逆アセンブラやデバッガなどを用いて内部のコードを解析すればこのような秘密情報が取得される可能性がある．このような攻撃は man-at-the-end (MATE) 攻撃と呼ばれ，MATE 攻撃からソフトウエアを保護するために，難読化が広く用いられている．難読化は，ソフトウエアの機能を保持しながら，内部のコードを変形し，解析を困難にする．

難読化手法の品質は，人間の攻撃者に対する解析しにくさ「効力」(*potency*)，自動解析に対する耐性「耐性」(*resilience*)，変形による性能変化の度合「コスト」(*cost*)，保護されていないコードとの区別のつきにくさ「ステルス」(*stealth*) といった 4 つの項目によって評価を行うことが考えられる [1]．本研究では難読化されたプログラムのステルスに着目する．難読化によって変形されたコードがめずらしい特徴を持つことで目立つ，つまり，ステルスが低くなる場合，難読化によって保護されている秘密情報の場所が攻撃者に知られやすくなり，難読化の効果の低下につながる．そのため，難読化されたプログラムのステルスを評価する方法は重要である．

本研究では，ステルスを評価する方法の 1 つとして，コードの不自然さ (*artificiality*) を数値化する方法 [3] および機械学習の手法を組み合わせて，あるプログラムが難読化されているかを判定するフレームワークを提案する．このフレームワークを用いて，与えられたプログラムが難読化されているかを判定するモデルを構築してケーススタディを行い，次のリサーチ・クエスチョンについて考察する．

Q1 難読化された・されていないプログラムは分類・検出できるか？

Q2 特徴が強い難読化方法があるか？

2　アプローチ

与えられたプログラムが難読化されているかを判定するモデルを構築するためのフレームワークを図 1 に示す．この図では，使用したツールの入出力 (プログラムやデータ) を楕円で，使用したツールを四角形で表す．フレームワークの流れとしては，まず難読化されていないプログラム (ノーマルプログラム) を用意して，難読化ツールによってコードの変形を行う (Step 1)．得られた難読化されたプログラムと，ノーマルプログラムをプログラムの特徴を取り出すツールに与えて，機械学習のための学習データを取得する (Step 2)．本研究では，逆アセンブラである IDA[1]や確率的言語モデルを構築する SRILM[2]といったツールに加えて，コードの不自然さ [3] の数値化方法を用いてプログラムの特徴を取得する．最後のステップ (Step 3) では，機械学習のアルゴリズムを使用して判定モデルを構築し，モデルの有効性を評価する．

一定のルールに従ってコード変形方法を用いた以上，各難読化手法には，論理的に何らかの判定できる特徴を生成すると考えられる [2]．したがって，機械学習によってデータを分類したり，データに含まれる外れ値を検出したり可能性があると考える．

機械学習の分類手法は，教師あり学習，つまり，既知のデータ (訓練データ) にクラスラベルを付けてモデルを学習し，未知のデータや将来のデータを予測する．異常検知手法は，ラベルは用いずに，多数のデータの中から異常なデータを検出する．本研究では，これらの方法を応用して，与えられたコードを「難読化されたコード」と「難読化されてないコード」に分類させたり，与えられたコードが「異常」かどうかを判定させたりする実験を行い，得られた結果から難読化されたコードのステルス (コードの目立ちにくさ) の評価を行う．

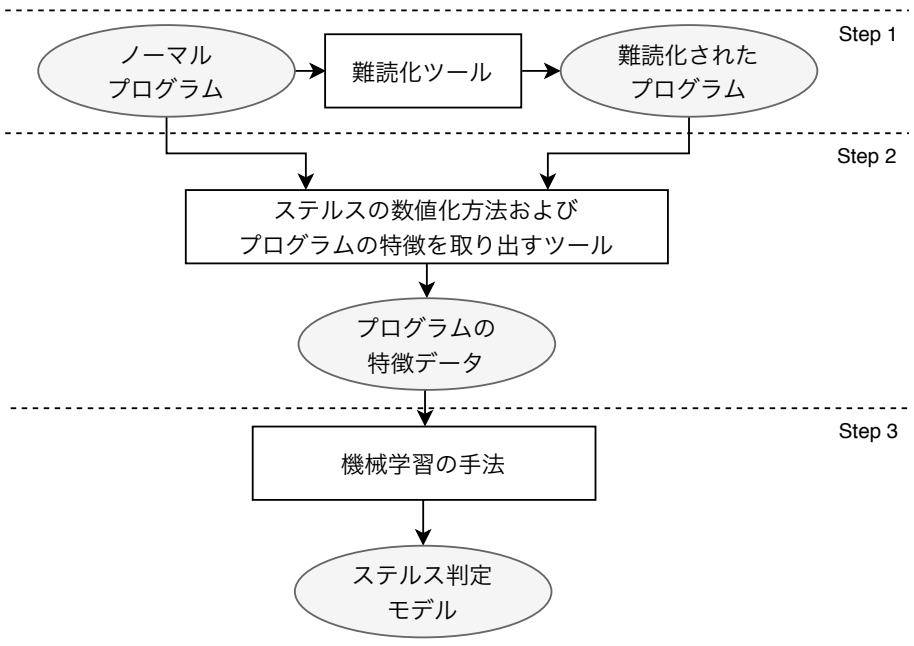

図 1: 提案フレームワークの概要

[1]IDA: https://www.hex-rays.com/products/ida/
[2]SRILM: http://www.speech.sri.com/projects/srilm/

3　ケーススタディ

本章では，2章で提案されたフレームワークに沿って，実際に判定モデルを構築し，リサーチ・クエスチョンについて調べる．

3.1　ノーマルプログラム

Step 1 に記述されたノーマルプログラムとしては，様々な特徴を持つ多くのプログラムを収集することが望ましい．しかし，オープンソースのプログラムは保護したいコードを含むとは限らない上，難読化後の動作確認が難しい場合がある．このため，CentOS[3] のシステムに分類される多様な 460 個のプログラム集合 P_{centos} に加えて，Tigress [4] の `RandomFuns` 変形オプションを使用して，難読化後の動作確認がしやすく，かつ，保護すべきコードを含むプログラムを数多く生成する．実験では，Tigress によってプログラムを 288 個生成された．`RandomFuns` 機能は関数単位で動作し，生成された関数はライセンスチェックのようなルーチンであり，難読化の目的はこの機構を保護することである．本実験で調整した `RandomFuns` のパラメータの説明を付録 B に示す．この 288 個のプログラム集合 P_{orig} は，実行時にコマンドラインから 4 つの引数「`42 42 42 42`」，パスワード「`secret`」を与えれば，“You win!” というメッセージが標準出力される．

`RandomFuns` によって生成されたプログラムは特徴を持つ可能性があり，多様性を高めるために Tigress の `split` オプションを用いて，P_{orig} の各プログラムにある 1 つの関数を複数の独立した関数に分割するコード変形を適用した．これらの 288 個のプログラム集合を P_{split} と表記する．

3.2　難読化手法

対象プログラムが多いことや実験の再現性を考慮した上で，難読化ツールを使用してコード変形を行う．コード変形の対象は `RandomFuns` により生成された 288 個のプログラム (P_{orig}) であり，行ったコード変形の概要を次に示す．

(1) P_{aaa}: 関数呼び出しの難読化が適用されたプログラム
ある関数への呼び出しを理解しにくい間接的な呼び出し方法に書き換える難読化法が適用されたプログラムである．

(2) P_{addop}: 理解しにくい条件分岐を挿入する難読化が適用されたプログラム
理解しにくく，かつ，どのような入力に対しても常に真か偽を返すような条件分岐を元のコードに挿入する難読化法が適用されたプログラムである．本実験では，10 個の条件分岐を挿入した．

(3) P_{enca}: 整数の算術の難読化が適用されたプログラム
算術の式をより複雑な式に書き換える難読化方法が適用されたプログラムである．

[3] CentOS: `https://www.centos.org/`

(4) P_{encl}: 整数と文字列リテラルの難読化が適用されたプログラム
静的な整数または文字列のリテラルを，動的に同じ値を出力する関数に書き換える難読化法が適用されたプログラムである．

(5) P_{flat}: 制御フローの平滑化が適用されたプログラム
ループや条件文のネストなどの制御構造を平滑化する難読化法が適用されたプログラムである．

(6) P_{vir}: インタプリタエンジンを用いた難読化が適用されたプログラム
全てのコードを独自のバイトコードに書き換え，そのバイトコードを解釈するためのインタプリタを挿入する難読化法が適用されたプログラムである．

(7) P_{jmp}: 分岐命令のカムフラージュが適用されたプログラム
文献 [5] で提案されたプログラムに含まれる分岐命令を非分岐命令でカムフラージュする動的な難読化方法が適用されたプログラムである．

上記の (1) から (6) までの変形は Tigress により，(7) の P_{jmp} は文献 [5] の方法で試作したツールにより行った．コード変形後，動作確認を行い，$288 \times 7 = 2,016$ 個の全てのプログラムが正常に動作していると確認した．これで Step1 が完了した．

3.3 メトリクス

Step 2 では訓練データの特徴を取得し，本研究のステルス評価に使用された特徴量をこの章に説明する．また，本研究の確率的言語モデルにおける基本構成要素であるコードにある 2 レベルの粒度 (*granularity*) についても説明する．

実験上では，CentOS のプログラムに含まれるアセンブリコードをコーパスとして使用している．`mov`，`push` のような単純なコードと，`mov_r.CODE`，`mov_w.BSS` のような命令

表 1: 評価対象プログラムのリスト

プログラム集合の名称	関数の数	適用された難読化
P_{centos}	28,469	なし (CentOS のシステムプログラム)
P_{orig}	2,592	なし (`RandFuns` によって生成されたもの)
P_{split}	5,760	なし (関数の分割を適用したもの)
P_{aaa}	2,880 (576)	関数呼び出しの難読化
P_{addop}	2,880 (570)	理解しにくい条件分岐を挿入する難読化
P_{enca}	2,880 (576)	整数の算術の難読化
P_{encl}	4,896 (576)	整数と文字列リテラルの難読化
P_{flat}	2,880 (576)	制御フローの平滑化
P_{vir}	2,880 (576)	インタプリタエンジンを用いた難読化
P_{jmp}	(2,304)	分岐命令のカムフラージュ

の書き込み「_w」または読み込み「_r」操作や操作する際に参照されるアドレスのセグメント (CODE，BSS) の詳細な情報を含むコードの 2 種類の 3-gram モデルのコーパスを使用した．本論文では，前者を「OPC」，後者を「XRF」で表記し，これらはコードの粒度という．

Tigress によるコード変形は関数単位で適用されるめ，3.1 と 3.2 章に準備した 460+288+2,304 = 3,052 個のプログラムも関数単位で評価し，各プログラムグループの名称，合計の関数の数，適用されたコード変形や取得方法の概要を表 1 に示す．括弧で囲まれる関数の数は難読化された関数の数を示している．これらのプログラムを IDA で逆アセンブルし，以下の学習メトリクスを求める．

(1) **Art**: 関数を構成するコード全体の不自然さの指標である．コードの不自然さについては付録 A に示す．

(2) **Max**: コードを 3-gram 分割したときの関数の最大の「驚き値」(付録 A) である．関数内の最も目立つコード列を見つけ出すという目的を持つメトリクスである．

(3) **OT**: 3-gram のコード片の驚き値を求め，定められた閾値を超えた関数内のコード片の数である．閾値は，P_{centos} の Max の平均値から求められる．ここでは，OPC の場合は閾値を 7.42 に，XRF の場合は閾値を 7.77 に設定した．つまり，あるコード片の驚き値が，多様な難読化されていないプログラムの Max の平均値を超えた場合，そのコード片が異常であると考えられ，異常なコード片の総数を示すメトリクスである．

(4) **Len**: 関数に含まれる命令の総数である．

(5) **Unq**: 関数内のユニークな命令の数である．例えば，関数が 5 つの `mov`，`and`，`push`，`push`，`mov` のコード $I = i_1...i_5$ から構成されているとき，$i_n \in \{$`mov`，`and`，`push`$\}$ であり，$|i_n| = 3$ であるから，Unq は 3 となる．また，XRF の場合，`mov_r.CODE`，`mov_w.BSS` はいずれも `mov` 命令であるが，命令操作や参照されるアドレスセグメントが異なるため，2 つの独特なコードとして扱うことにしている．このメトリクスは，例えば `mov` のみを生成するコンパイラ [6] によって生成されるコードの特徴を把握できる．

(1) から (3) までのメトリクスは逆アセンブルされたプログラムのコードとコーパスを SRILM に与えて求めた．

3.4 機械学習モデル

Step 3 の実行に当たって，まず，データの特徴を把握するために，各メトリクス間の線形な関係の強弱指標である相関係数を求める必要がある．表 2 は，全てのデータを結合した後のピアソン相関係数 [7] を示し，全ての係数は正であることから全てのメトリクス間は比例することが分かる．

表 2: 各メトリクスの相関係数

	Art	Max	Unq	Len	OT
Art	1.0	0.30	0.46	0.96	0.73
Max	0.30	1.0	0.44	0.28	0.31
Unq	0.46	0.44	1.0	0.52	0.27
Len	0.96	0.28	0.52	1.0	0.57
OT	0.73	0.31	0.27	0.57	1.0

最大の相関係数は Art と Len の間の 0.96 であり，この 2 つのメトリクスは非常に強く関係し，関数に含まれるコードが多いほど関数が不自然であると解釈できる．次に大きい相関係数は Art と OT の間の 0.73 であり，閾値を超えたコード片の数が多いほど関数が不自然であるといえる．Unq と OT 以外の他のメトリクスや，OT と Len が [0.4, 0.6) の範囲にあることから，独特なコード数が多いほどコード全体が不自然になるとは限らず，関数が長いほど独特なコード数や閾値を超えたコード片の数も増えるとは限らない．残りの係数は [0.2, 0.4) の範囲に存在し，これらのメトリクスの間は弱く関係すると考えられる．

モデルを構築するために Python[4]の機械学習パッケージである scikit-learn[5]を使用した．このパッケージに分類問題に用いたランダムフォレストや異常検知に用いた One Class SVM といったアルゴリズムが実装されている．

3.4.1 分類問題

難読化された・されていないの分類

リサーチ・クエスチョン Q1 を調べるために，難読化された・されていないプログラムの 2 クラスを分類するモデルを構築する．使用したアルゴリズムは，Breiman, L. によって提案された決定木を拡張したランダムフォレスト [8] である．決定木での分類の概要は

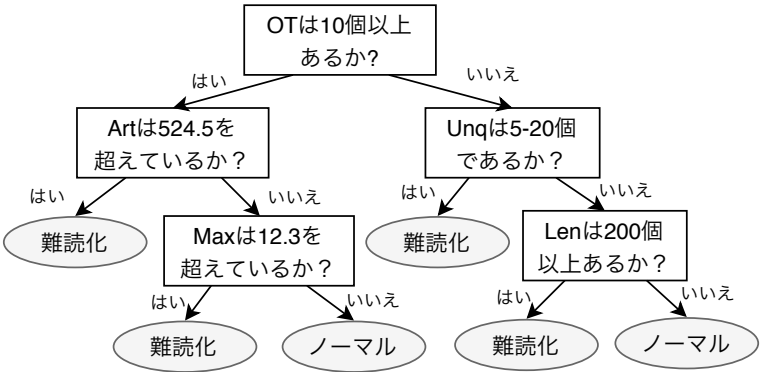

図 2: 決定木の例

[4]https://www.python.org/
[5]http://scikit-learn.org/stable/

図 2 に示すように，訓練データのメトリクスをもとに，木構造の一連の質問を学習し，訓練データラベルを推測する．本実験では 10 本の決定木によって学習を行った．また，訓練において P_{centos}, P_{orig}, P_{split} に含まれる関数や Tigress によって難読化されていない関数を難読化されていない (正)，それ以外の関数は難読化された (負) というラベルを付けた．

ランダムフォレストは過学習になる可能性が高く，学習モデルの妥当性を確認するために，訓練データから除いた 2 割のテストデータに加えて，残りの 8 割を 10 分割交差検証でランダムフォレストによる学習を行った．10 分割交差検証では，訓練データ (今回はもとデータの 8 割) を 10 個に分割し，そのうちの 9 個を訓練データとして学習を行い，残りの 1 個をテストデータとする．10 個に分割されたデータそれぞれをテストデータとして 10 回の検証結果の平均値を推定結果として扱う．ここで，検証される指標は正解率 (*accuracy*) である．10 分割交差検証を行った後，8 割の訓練データでモデルを構築し，残り 2 割のテストデータで最終モデルの評価を行った．学習モデルの評価結果は表 3(a) にまとめる．CF は 10 分割交差検証，Val は最終モデルの正解率を示している．適合率 (*precision*) は予測が正と判定した中で，ラベルも正のもの，つまり，誤分類を起こさない性能の指標である．一方，再現率 (*recall*) はラベルが正の中で，予測が正と判定したもの，つまり，正のラベルを見つけ出す能力の指標である．適合率と再現率を組みわせて評価する指標は F1 スコア (*F1-Score*) である．正解率，適合率，再現率，F1 スコアの求め方は付録 C に示す．

表 3(a) の 10 分割交差および最終モデルの検証結果より，判定モデルは妥当であり，90% 程度の正解率が得られた．このことに加えて，適合率，再現率，と F1 スコアの最終モデルの性能評価指標により，難読化された・されていないの判定は可能であることが分かった．また，OPC と XRF の性能に有意差がないことが分かった．

表 3: 構築した分類モデルの評価結果

3(a) 2 クラスの分類モデル

		OPC	XRF
accuracy (%)	CF	96.3	96.3
	Val	96.5	96.4
precision (%)		97	97
recall (%)		96	96
F1-Score (%)		96	96

3(b) 8 クラスの分類モデル

		OPC	XRF
accuracy (%)	CF	96.2	96.2
	Val	96.4	96.4
precision (%)		97	97
recall (%)		96	96
F1-Score (%)		95	95

難読化法による分類

リサーチ・クエスチョン Q2 を調べるために，プログラムが難読化手法によって分類するモデルを構築する．訓練データにおいて P_{centos}, P_{orig}, P_{split} に含まれる関数や Tigress によって難読化されていない関数をを難読化されていないというラベル P_{norm}，それ以外の関数はそれぞれの難読化手法によってラベルを付け，難読化法ごとに分けた 8 クラスを分類することを目的とする．学習アルゴリズムと妥当性の検証方法は難読化された・されていないの分類と同様に，学習モデルの全体的評価結果は表 3(b) に示す．適合率，再

表 4: 8 クラス分類モデルのクラス別の評価結果

集合名称	OPC			XRF		
	pre (%)	rec (%)	F1 (%)	pre (%)	rec (%)	F1 (%)
P_{norm}	96	100	98	96	100	98
P_{aaa}	100	100	100	100	100	100
P_{addop}	94	94	94	96	95	96
P_{enca}	99	99	99	99	97	98
P_{encl}	95	96	96	95	94	95
P_{flat}	96	94	95	94	95	94
P_{vir}	100	100	100	100	100	100
P_{jump}	100	12	21	100	12	21

現率，F1 スコアは各クラスの平均値を取ったものを示す．この表から全ての評価指標が 2 クラス分類と同じような結果になることが分かった．また，OPC と XRF の性能に有意差がないことが分かった．

クラス別の最終モデルの評価結果を表 4 に示す．P_{aaa} や P_{vir} の F1 スコアは 100%であり，関数呼び出しおよびインタプリタエンジンを用いたの難読化方法の特徴が強く，ステルスが低いと考えられる．また，P_{jump} の再現率が低いことから，分岐命令をカムフラージュする難読化方法が適用された関数を見つけ出しにくく，ステルスが最も高いといえる．このことから，適用された難読化方法によって，難読化されたプログラムを分類できることが分かった．

3.4.2 異常検知

サポートベクトルマシンを拡張した One Class SVM [9] アルゴリズムを用いて，訓練データにラベル付けなくても特徴量 (メトリクス) の構造を抽出できることが学習の目的である．本実験では，放射基底関数を SVM のカーネルにし，P_{centos}, P_{orig}, P_{split} を訓練データとして扱い，モデルを構築する．構築されたモデルに評価対象プログラムを順番に与え，そのプログラムは異常 (難読化されている) かどうかを予測する．

モデルの評価を行うために 2 クラス分類問題と同様に，難読化されたプログラムを負，難読化されていないプログラムを正のラベルを付けて正解率を評価指標とする．正解率については付録 C に示す．評価結果を表 5 に示す．訓練データの正解率の平均値は 75%以上であり，訓練データはなんらかの特徴を持つと考えられる．この表から，他のプログラム集合に比べて P_{aaa} や P_{vir} の正解率が比較的に高いことが分かった．また，OPC に比べて XRF の正解率が低いことも見られる．ただし，XRF の場合 P_{encl} の正解率が最も高くなっており，P_{jmp} の正解率が上がることも確認できる．

OPC の異常検知前後を図 3 と図 4 に示している．横軸は Len，縦軸は左上の図から順に右へ順に Art, OT, Max, Unq を示して，同じコード変形が適用された関数が同じ座標にある場合，色の濃さによって見分けられるようにした．異常として検出された箇所は赤い塗り潰されていない円形で囲まれている．図 3 を見ると，P_{enca}「✕」が P_{norm}「青い円形」よりも Len が大きく，これが整数の算術の難読化特徴であると考えらる．また，図 3 の Len と OT の関係図を見ると，P_{jmp}「✖」は，他のプログラム集合に比べて傾きが異なっていることが分かった．

異常検知を行なった後の図 4 を見ると，いくつかの誤認したところを確認できる．この異常検知モデルにより，各難読化手法は特徴を持つことを確認できるが，訓練データにラベルを付けないため，分類モデルに比べて正解率が低くなると考えられる．

表 5: 異常検知モデルの評価結果

	OPC	XRF
P_{centos}	87.9%	86.9%
P_{orig}	83.8%	67.4%
P_{split}	64.4%	85.7%
P_{aaa}	90.0%	80.0%
P_{addop}	89.8%	75.7%
P_{enca}	89.8%	78.6%
P_{encl}	88.2%	82.1%
P_{flat}	87.3%	79.7%
P_{vir}	90.0%	80.0%
P_{jmp}	25.0%	37.5%

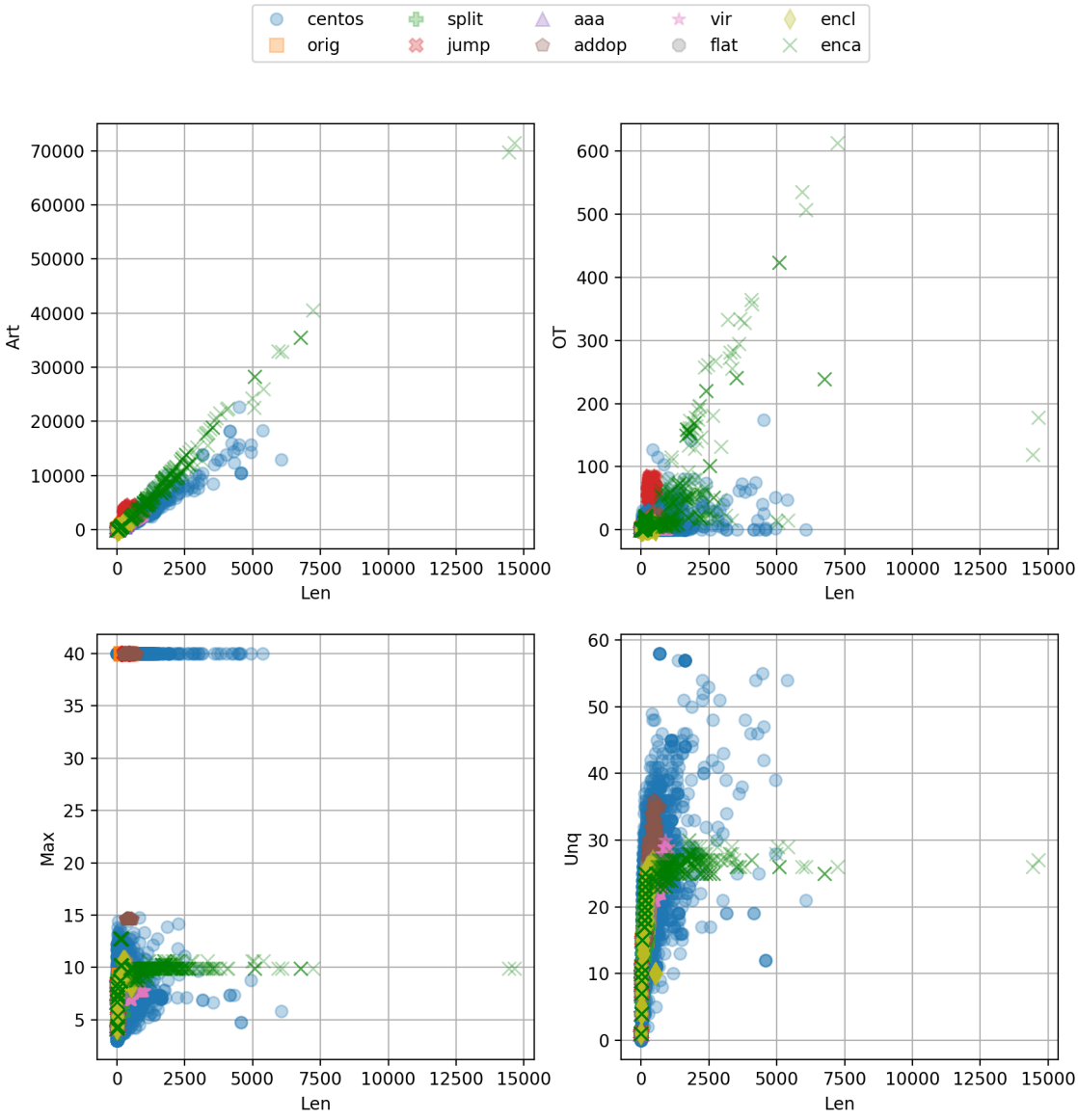
centos
orig
split
jump
aaa
addop
vir
flat
encl
enca
Art
OT
Max
Unq
Len

図 3: 検出前

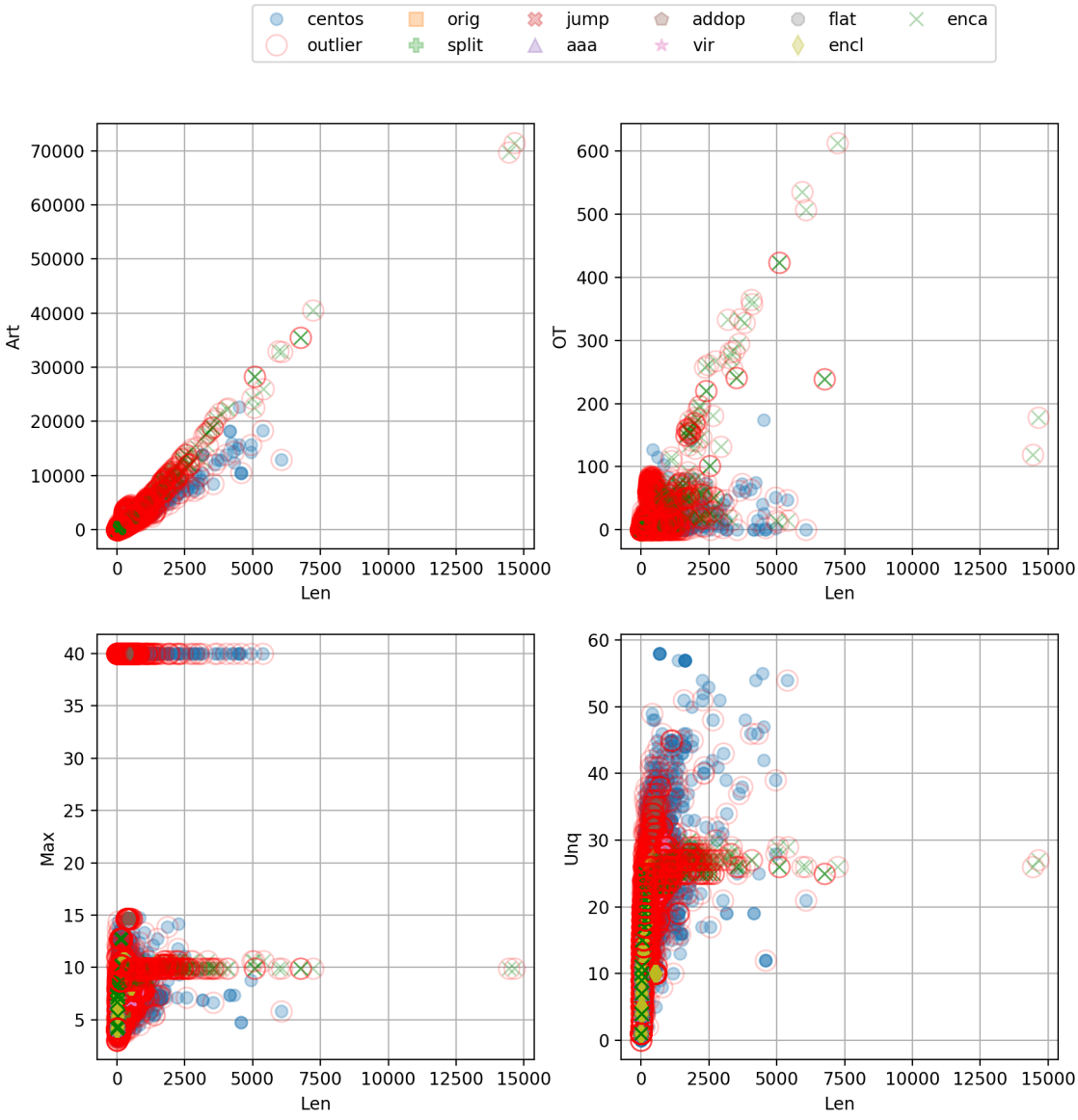
centos
outlier
orig
split
jump
aaa
addop
vir
flat
encl
enca
Art
OT
Max
Unq
Len
Len
Len
Len

図 4: 検出後

3.5　ステルス数値化法の有効性

ランダムフォレストのモデル構築に当たって，特徴量の重要度 [10][11] を計算することもできる．特徴量の重要度は以下の手順に従って求められる．

1. ある特徴量に属するデータをランダムに変化させ，変化前後の正解率の差を求める．
2. 特徴量の数だけ手順 1 を繰り返す．本研究では，特徴量が 5 つあるため，手順 1 を 5 回繰り返す．
3. 正解率の差を基準に，特徴量を順に並べ替える．正解率の差が大きいほどその特徴量が重要である．

分類モデルにおける OPC の特徴量の重要度を図 5 に示す．ここでは XRF の図を表示しないが，OT が最も重要であることが分かる． 3.3 章で説明したように，OT は定められた驚き値の閾値を超えたコード片の総数であることから，驚き値は有効なステルス数値化法といえる．

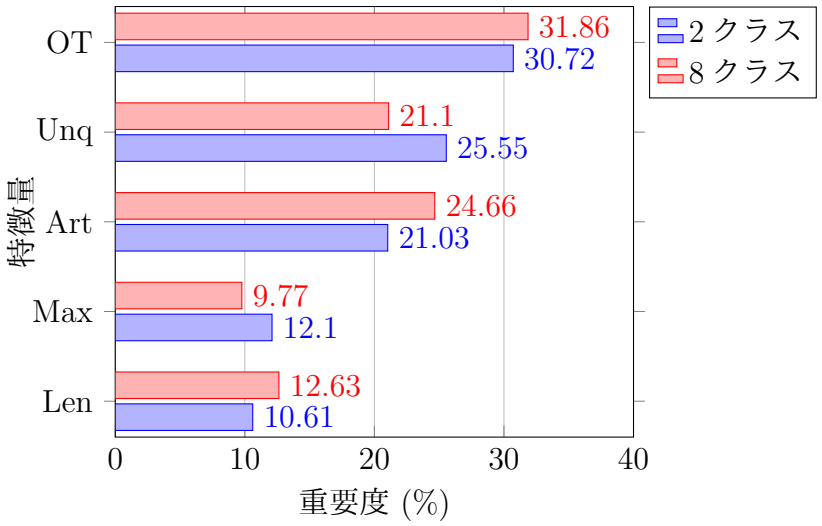

図 5: 特徴量の重要度

4　まとめ

本研究では，機械学習によってプログラムが難読化されているかどうかを判定するフレームワークを提案した．ケーススタディではフレームワークの有用性を調べるために，合計 3,052 のプログラムを用いてステルスを評価する機械学習のモデルを構築した．最良のモデルでは 95%以上の精度でデータセットにあるプログラムを判定でき，難読化された・されていないプログラムは分類できることが分かる．各難読化手法には特徴があり，分岐命令をカムフラージュする難読化方法の特徴が，実験で適用した他の難読化手法よりも弱く，ステルスが高いと考えられる．

今後の課題として，難読化方法やメトリクスの種類を増やすことで，さらに有用なステルス評価モデルを検討することが挙げられる．また，ステルス評価の学習モデルを用いて，複数の難読化方法を組み合わせることによってステルスの高いプログラムを生成する難読化システムの開発も研究テーマとして考えられる．

参考文献・参考URL

[1] Banescu, S., Collberg, C., Pretschner, A. “Predicting the Resilience of Obfuscated Code Against Symbolic Execution Attacks via Machine Learning”, In *Proc. of the 26th USENIX Security Symposium* (2017), USENIX, pp.661-678

[2] Kanzaki, Y., Thomborson, C., Monden, A., and Collberg, C. “Pinpointing and Hiding Suprising Fragments in an Obfuscated Program”, In *Proc. of the 5th Program Protection and Reverse Engineering Workshop* (2015), PPREW-5, ACM, pp.8:1-8:9.

[3] Kanzaki, Y., Monden, A., and Collberg, C. “Code Artificiality: A Metric for the Code Stealth Based on an N-gram Model”, In *Proc. of the 1st International Workshop on Software Protection* (2015), IEEE Press, pp.31-37

[4] Collberg, C.: The Tigress C Diversifier/Obfuscator, `http://tigress.cs.arizona.edu/`. (accessed: Feb. 2018)

[5] 村上隼之助，神崎雄一郎，門田暁人，“分岐命令のカムフラージュに基づくプログラムの制御フローの隠ぺい”，火の国情報シンポジウム 2018，情報処理学会九州支部 (2018).

[6] Domas, C.: M/o/Vfuscator2 :: the single instruction C compiler, `https://github.com/xoreaxeaxeax/movfuscator`.

[7] Pearson, K. “Note on regression and inheritance in the case of two parents”, *Proc. of the Royal Society of London 58* (1895), 240-242

[8] Breiman, L. “Random forests”, *Machine Learning 45* (2001), 5-32

[9] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., Williamson, R. C. “Estimating the Support of a High-Dimensional Distribution”, *Neural computation 13.7* (2001), 1443-1471

[10] Feature Selection, `http://scikit-learn.org/stable/modules/feature_selection.html`

[11] Feature importances with forests of trees, `http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html#sphx-glr-auto-examples-ensemble-plot-forest-importances-py`

（URL は 2018 年 2 月現在）

謝辞

本研究を行うにあたり，ご指導を頂いた卒業研究指導教員の神崎雄一郎准教授に心より感謝致します．日常の議論を通じて多くの知識を頂き，本当にありがとうございます．

付録

A　コードの不自然さと驚き値

本論文では，ソフトウエアを構成するプログラムの一部あるいは全部を x86 アーキテクチャのアセンブリ言語で表したものをコードと呼ぶ．また，コードに含まれている 1 つのアセンブリ命令を N 個ずつ並べたものを N-gram といい，例えばコードが`mov`，`and`，`push`，`push`，から構成されているとき，3 つずつ並べると，「`mov-and-push`」，「`and-push-push`」という 2 つの命令列を得て，コードを 3-gram に分割したという．コードの N-gram は長さ N のコード片とも呼び，i_j^n はコード全体の j 番目から始まる長さ n のコード片を示し，1 つの命令の場合は，単に i_j と記述する．

N-gram モデルに基いたコードの不自然さの数値化法 [3] と同様に，難読化されていないプログラムのコードを多く集めたものをコーパスとした確率的言語モデルを用いて，与えられた命令列の自然さを求める．具体的な例として，コードの全体 C が m 個の命令から構成され，$i_1^m = i_1 i_2 ... i_m$ の命令列で表したとき，生起確率 $P(i_1^m)$ は次のように近似する．

$$P(i_1^m) \approx \prod_{k=1}^{m} P(i_k | i_{k-N+1}^{k-1}) \tag{1}$$

生起確率 $P(i_1^m)$ の値が大きいほど命令列が自然であると考えられる．この生起確率 $P(i_1^m)$ から底が 2 の対数を取って得られた値は，コード全体の不自然さをビット単位で表したものであり，次のように定義する．

$$Artificiality \quad A(C) = -log_2 P(i_1^m) \quad [bit] \tag{2}$$

式 (1) とは逆であり，$A(C)$ が大きいほどコード C が不自然であると考える．

一方，目立つコード片はコーパスに出現する数が最も少なく，コード片の目立つさの度合「驚き値」(*Surprisal*) は次のような式で定義される．

$$S(i_j^n) = -log_2 P(i_{j+n-1} | i_j^{n-1}) \quad [bit] \tag{3}$$

例えば，上記の 3-gram を考えると，後者の驚き値は $S(i_2^3) = S($`and-push-push`$)$ と示し，実際本研究で使用したコーパスでは 4.365 と計算される．不自然さと同様に $S()$ が大きいほどコーパスに出現する数が少なく，本研究では，コーパスに出現しないコード片を無限大でない大きい値を与える．

B RandomFuns のパラメータ

RandomFunsControlStructures
: 生成される関数の制御構造を定められた文法記述によって調整する．本実験では「(if (bb n)(bb n))」,「(for (bb n))」,「(if (bb n) (for (bb n)))」の 3 つの中から構造を選ぶものとする．文法にある (bb n) は基本ブロックに n 個のステートメントを生成する．なお，n は RandomFunsBasicBlockSize によって調整でき，今回は，$n \in \{1,2\}$，の 2 つの中から選ぶものとする．

RandomFunsType
: 入出力とステートメント配列のデータ型を調整する．今回は，Type $\in$ {char, short, int, long}，の 4 つの中から選ぶものとする．

RandomFunsForBound
: 生成されたプログラムにある for 文の範囲を調整する．今回は，Bound $\in$ {constant, boundedInput, boundedAny}，の 3 つの中から選ぶものとする．なお，constant は整数のリテラルを用い，boundedInput と boundedAny は範囲外の添字を選択しないようし，それぞれ入力配列からの値と，任意のソースからの値を用いて for 文の範囲を決定する．

RandomFunsOperators
: 生成された関数内に使用可能なオペレータを設定する．本実験では，Opr $\in$ { 簡単な比較と算術演算子，複雑な比較と算術演算子，比較とビット演算子，全ての演算子 }，の 4 つの中から選ぶものとする．

RandomFunsPassword
: ユーザが標準入力から入力するパスワードを設定する．本実験では,「secret」と設定し，RandomFunsPasswordCheckCount を 1 に設定することでパスワード確認を 1 回行う．

RandomFunsFailureKind
: 上記のパスワード確認，有効期限確認，実行許可コードが失敗した場合,「abort()」という関数が呼び出される．

RandomFunsTimeCheckCount
: ライセンスチェックのような有効期限の確認ルーチンを挿入する．今回は，1 と設定した．ただし，動作確認のため有効期限確認を常に成功（有効期限内）にしている．

RandomFunsActivationCodeCheckCount
: 生成されたプログラムの実行許可コードの確認回数を設定する．今回は，1 と設定した．

RandomFunsPointTest
: 設定された値と，生成された関数によって変換された入力が一致した場合 “You win!”

のメッセージを標準出力する．今回は，実行時に 4 つのコマンドラインの引数が 42 である場合メッセージを出力するように設定した．

したがって，合計の組み合わせの数が $3 \times 2 \times 4 \times 3 \times 4 = 288$ 個のプログラムである．なお，生成のシードを 7 にし，上記以外のパラメータをデフォルト値に設定した．

C　正解率，適合率，再現率，と F1 スコア

あるクラス C_i に対する分割表を表 6 に示している．

表 6: 分割表

	実際に C_i に属する	実際に C_i に属さない
C_i と判定される	*true positive (tp)*	*false positive (fp)*
C_i 以外と判定される	*false negative (fn)*	*true negative (tn)*

この表に従って，正解率，適合率，再現率，および F1 スコアは次式によって求めらる．

$$\text{正解率} = \frac{\text{正解の判定}}{\text{全てのサンプル}} = \frac{tp + tn}{tp + tn + fp + fn} \tag{4}$$

$$\text{適合率} = \frac{\text{正しく } C_i \text{に属すると判定される}}{C_i \text{と判定される}} = \frac{tp}{tp + fp} \tag{5}$$

$$\text{適合率} = \frac{\text{正しく } C_i \text{に属すると判定される}}{\text{実際に } C_i \text{に属する}} = \frac{tp}{tp + fn} \tag{6}$$

$$F1 \text{ スコア} = 2 \times \frac{\text{適合率} \times \text{再現率}}{\text{適合率} + \text{再現率}} \tag{7}$$

式 (4) の正解率は，正しい判定の割合を示す．また，式 (7) より，適合率又は再現率が 0 である場合，F1 スコアが 0 になることが確認できる．再現率と適合率の両方が 0 である場合，テストデータに偏りがある（C_i に属する項目がない等) と考えられ，テストデータは無効であると考えられる．