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

Evaluating the Stealth of Obfuscated Program via Machine Learning

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

Abstract: Software obfuscation techniques are widely used to protect security-sensitive codes of a program. One way to evaluate the quality of these techniques is by evaluating its stealthiness, that is, the degree to which the transformed protected code can be distinguished from unprotected code. This paper proposes a framework for evaluating the stealthiness of an obfuscated program through machine learning methods. In the case study, the best model is able to classify between obfuscated and unobfuscated programs used in the dataset with more than 95% accuracy.

1. はじめに

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攻撃者に目立つ場合，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 (*stealth*)，すなわち，保護されていないコードとの区別のつきにくさ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，機械学習の分類の手法を用いて「プログラムが難読化されているかどうか」を判定するモデルを構築するフレームワークを提案する．構築したモデルによって難読化されていないと判定される難読化されたプログラムは，ステルスが高いと考える．ケーススタディでは，既存の難読化方法を用いてモデルを構築し，提案方法の有用性について確認する．

2. アプローチ

本研究のフレームワークの概要を図 1 に示す．この図では，使用したツールを四角形で，ツールの入出力 (プログラムやデータ) を楕円で表す．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)．次に，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]などのツールに加えて，確率的言語モデルに基づいたコードの不自然さ [1] の数値化方法を用いてプログラムの特徴を取得する．一定のルールに従ってコードを変形する以上，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 [2] ため，機械学習によってデータを分類できると考える．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する．

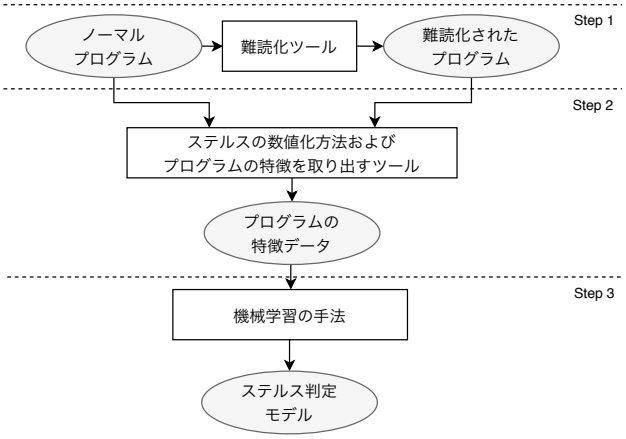

図 1: 提案フレームワークの概要

3. ケーススタディ

3.1 モデルの構築

提案フレームワークに沿った判定モデルを実際に構築し，難読化されたプログラムを判定できるかどうかについて調べるケーススタディを行う．

(Step 1) モデル構築に用いるプログラム

まず，ノーマルプログラムとして，次のものを用意した．

(1) CentOS[*2]のシステムに分類される多様な 460 個のプログラム．

(2) Tigress[*3]の `RandomFuns` 変形オプションを使用して生成した 288 個のプログラム．これらは難読化後の動作確認がしやすく，保護すべきコードを含む．

[*1] IDA: `https://www.hex-rays.com/products/ida/`

[*2] CentOS: `https://www.centos.org/`

[*3] Tigress: `http://tigress.cs.arizona.edu/`

(3) (2) の一部を分割した 288 個のプログラム．

また，難読化されたプログラムは，(2) を Tigress による難読化及び命令のカムフラージュ法の 7 つの方法を用いて難読化したもの (288 個 × 7) とした．

(Step 2) プログラムの特徴データ

(Step 1) で準備したプログラムを IDA で逆アセンブルし，以下のような関数単位のメトリクスを用いて，プログラムの特徴データを取得する．

1. **Art**: 関数を構成するコード全体の不自然さ [1] の指標である．
2. **Max**: コードを 3-gram 分割したときの関数の最大の「驚き値」 [2] である．
3. **OT**: 3-gram でコードを分割して得られるコード片のうち，驚き値が定められた閾値を超えた関数内のコード片の数である．
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．

(Step 3) 機械学習モデル

(Step 2) で取得したプログラムの特徴データを用いて，モデルを構築する．モデルの構築では，Python[*4]の機械学習パッケージである scikit-learn[*5]を使用した．

Breiman 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 [3] アルゴリズムを用いて，分類するモデルを構築する．決定木でのプログラムが難読化されたかされていないかの 2 クラスを分類のイメージを図 2 に示す．訓練データのメトリクスをもとに，訓練データラベルを推測することによって，質問の条件を決定し，木構造の一連の質問を学習する．本実験では 10 本の決定木によって学習を行った．また，2 クラスの分類問題に加え，7 種類の難読化の各クラスとノーマルクラスの計 8 クラスを分類するモデルを構築した．

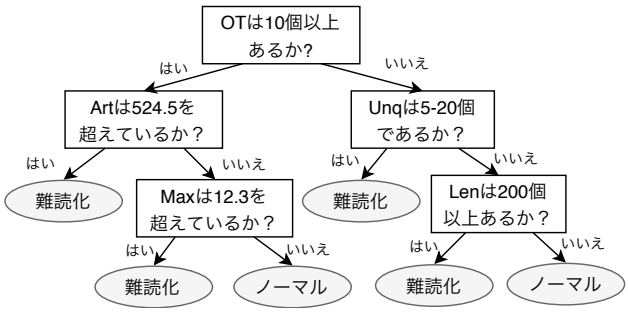

図 2: 本実験での決定木のイメージ

[*4] https://www.python.org/

[*5] http://scikit-learn.org/stable/

3.2 モデルの評価結果

学習モデルの妥当性を確認するために，10 分割交差検証，および，テストデータでの検証を行った．表 1 に，正解率 (*accuracy*) による 2 クラスと 8 クラスのモデルの評価結果を示す．CF は 10 分割交差検証，Val はテストデータの正解率を示している．

表 1: 分類モデルの評価結果

	2 クラス	8 クラス
CF	96.31%	96.23%
Val	96.48%	96.41%

2 クラス分類モデルにより，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，難読化法による 8 クラス分類モデルにより，分岐命令をカムフラージュする難読化 [4] などのステルスが高いと思われる難読化法があることが分かった．

4. おわりに

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．このフレームワークを用いた実験においては，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た．

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．

参考文献

[1] 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), pp.31-37.

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

[4] 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)．