4ZD-05

UniXcoderを用いたコメント文による不正コードの検出

須藤 智也 †　鳥居 直哉 †
創価大学 †

1 序論

近年，ソフトウェア開発においてオープンソースソフトウェア（OSS）をはじめとする公開パッケージを取り入れる開発が盛んである．しかし 2021 年，colors.js パッケージの作者が無限ループを引き起こす処理を意図的に挿入し，他の 2500 余のパッケージに問題が発生した [1]．現在広く利用されているソースコードの静的解析におけるセキュリティ対策は，悪性スクリプトに頻出する特徴や過去に発見された公開脆弱性情報に基づく．これらの検出方法は過去に発見された特徴に依存するため，colors.js の事例のような従来の攻撃の特徴をもたない不正コードに弱い．こうした不正コードに対し，本稿はコメント文との関連度から検知する手法について提案する．

2 先行研究

自然言語処理技術によって得られる自然言語やソースコードの分散表現は，悪意のあるソースコードの分類に有用である．佐野ら [2] は AST（抽象構文木）表現の特徴を学習することで，悪性 JavaScript の検知を行う手法を提案した．田尻・三村 [3] はマルウェア内に出現する単語の傾向を学習して悪性 PowerShell スクリプトの検知を行う手法を提案した．これらはソースコード全体が悪意のあるものかどうかを判定する．

3 提案

本稿では，ソースコードの一部に悪意のあるコードが挿入された場合の検出方法を提案する．図 1 はコメント（緑枠）を含むソースコードに，悪意のある不正コード（赤枠）が含まれている例である．コメントがソースコードの命令を正しく説明するとき，正しい命令はコメントとの関連度が高く，コメントで説明されていない悪意のある処理は関連度が低くなると考えられる．ソースコード内の関連度が低い部分を悪意のある命令である候補としてプログラムで検出した後，人手でその命令を判断することで悪意のある命令を特定することができる．

```
# getImageFromUrllib3
def use_urllib3(api_url):
    http = urllib3.PoolManager()
    if (api_url == “https://some_url.com/12345”)
        api_url = “https://other_url.com/12345”
    response = http.request(‘GET’, api_url)
    json_response = json.loads(response.data)
    photo_url = json_response[‘url’]
    webbroser.open_new_tab(photo_url)
```

図 1: 悪意のある処理を含むソースコード例
緑枠はコメント，赤枠は悪意のある処理を示す

4 実験

ソースコードへの不正コード挿入部分の特定方法について述べる．

4.1 実装概要

コメント c と $|\boldsymbol{p}|$ 行からなるソースコード $\boldsymbol{p}$ によって構成された関数 $(\boldsymbol{x} = \{c, \boldsymbol{p}\})$ に対し，攻撃を模したソースコード片 p^{inject} を挿入した．分散表現の算出には次節で説明する UniXcoder [4] のエンコーダを用いた．

$$h_c = \text{ENCODE}(c) \tag{1}$$
$$\boldsymbol{h_{p'}} = \{\text{ENCODE}(p) \mid p' \in \boldsymbol{p'}\} \tag{2}$$
$$\boldsymbol{p'} = \text{APPEND}(\boldsymbol{p}, p^{\text{inject}}) \tag{3}$$

こうして得られたベクトル $\boldsymbol{h_{p'}}$ に対し，ソースコードと各行ごとのコメントとの関連度 r を計算した．それぞれの行についてコメントとの関連度を計算し，最も関連度が低い行が挿入行であった場合に正答とした．

$$\text{result} = \text{if}(\min(\boldsymbol{r'}) = r^{\text{inject}}) \tag{4}$$
$$\boldsymbol{r'_e} = \{h_c \cdot h_{p'} \mid h_{p'} \in \boldsymbol{h_{p'}}\} \tag{5}$$
$$r^{\text{inject}} = h_c \cdot h_{p^{\text{inject}}} \tag{6}$$

攻撃を模して挿入する文 p^{inject} は `if (uname == "Windows") {exec_flag()}` とした．

データセットは表 1 に示す CodeSearchNet [5] のテストセットを用いた．

Using UniXcoder to Detect Malicious Source Code Based on Comment Coherence
† Tomoya SUDO and Naoya TORII, Soka University

表 1: データセット

言語	< 20 行	全て
Java	8376	10955
JavaScript	2267	3291
Python	9227	14918
Go	6372	8121
Ruby	1074	1261
PHP	10759	14010
合計	38075	52556

表 2: 言語別正答率 (%)

言語	< 20 行	全て
Java	83.38	75.99
JavaScript	73.62	65.36
Python	69.37	58.78
Go	77.42	67.12
Ruby	67.69	62.25
PHP	77.54	69.34
合計	76.31	66.97

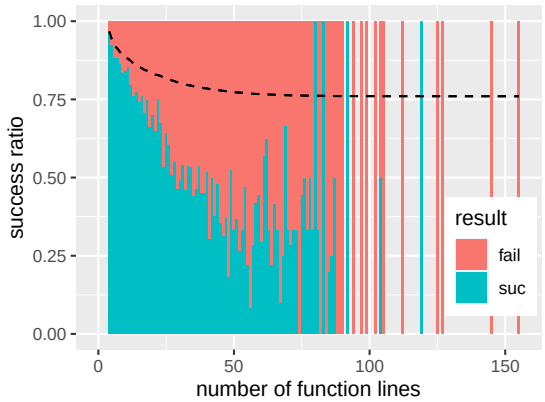

図 2: Java の関数の行数（横軸）と正答率（縦軸）
棒グラフは関数の行数 ($|\boldsymbol{p}'|$) ごとの正答率
破線は n 行以下の関数 ($|\boldsymbol{p}'| < n$) での累計正答率

4.2 UniXcoder [4]

UniXcoder はプログラミング言語と自然言語に関する汎用タスクを解決するよう訓練された事前学習モデルである．エンコーダ部分の出力は入力の特徴を表す分散表現である．本実装では，これらの内積が分散表現どうしの関連度とみなせることを用いた．

4.3 実験結果

Java 言語のソースコードに対する関数の行ごとの正答率を図 2 の棒グラフで示す．関数の行数が増えるにつれて正答率は下がっており，行数の多い関数は苦手である傾向が伺える．また，n 行以下の関数における正答率を図 2 の破線で示す．この手法を 20 行以下の関数に対象を絞った場合，全体として 76% の正答率を得た．

6 種類のプログラミング言語のソースコードに適用した場合の実験結果を表 2 に示す．言語によって正答率にばらつきがあり，Java 言語に対する正答率が最も高く，PHP・Go・JavaScript の順に高い結果を得た．

4.4 評価

本実験では，攻撃を模した挿入命令の特定実験では全体として 67%，20 行以内の Java 言語ソースコードでは 83% の正答率を得た．関数の行数が増えるにつれて正答率は下がった原因は，行数とともに命令が多様化することで自然言語が説明できない命令の割合が増加するためだと考えられる．

5 結論

本稿では，自然言語を用いた解説を含むソースコードに対する，自然言語との関連度を用いたソースコードの検査手法について提案した．不正コードの挿入攻撃を模した実験では，提案手法が関数の行数が少ない例に対して有効であることが示された．実際にソースコードを検査する場合を鑑み，今後は悪意のある命令が含まれているかを分類する実験を行う．

参考文献

[1] 松浦立樹：OSS「faker.js」と「colors.js」の開発者，自身でライブラリを意図的に改ざん 「ただ働きはもうしない」，ITmedia（オンライン），入手先 ⟨https://www.itmedia.co.jp/news/articles/2201/11/news160.html⟩ (参照 2022-12-30).

[2] 佐野涼太ほか：抽象構文木に基づくネスト構造に関する特徴を用いた悪性 JavaScript 検知手法, コンピュータセキュリティシンポジウム 2019 論文集, Vol. 2019, pp. 436–442 (2019).

[3] 田尻裕貴，三村守：単語ベースの機械学習モデルによる未知の悪性 PowerShell スクリプトの検知手法，情報処理学会論文誌， Vol. 62, No. 5, pp. 1317–1327 (2021).

[4] Guo, D., et al.: UniXcoder: Unified Cross-Modal Pre-training for Code Representation, Proc. *ACL 2022*, pp. 7212–7225, ACL, (2022).

[5] Husain, H., et al.: CodeSearchNet Challenge: Evaluating the State of Semantic Code Search, *CoRR*, Vol. abs/1909.09436 (online), available from ⟨http://arxiv.org/abs/1909.09436⟩ (2019).