機械学習を用いたソースコードからの脆弱性検知に関する研究

2020SE076 矢越弘朗
指導教員：野呂昌満

1　はじめに

デジタル技術が発展するに伴い, セキュリティ対策の重要性も高まっている．従来の静的プログラム解析では, 決定論的にプログラムの中の脆弱性を検出しており, この手法では新たなパターンの脆弱性や, 動的意味に関係する脆弱性は検出しづらい．そういった脆弱性に対しては新たな仕様を定義し, プログラムを作り直す必要がある．

本研究の目的は,GNN を用いた, 脆弱性検出の可能性の考察である．NN(Neural Network) ベースの手法では, 大量のデータから脆弱性のパターンを学習して発見的に脆弱性を予測する．この手法はデータ駆動形の自動生成された仕様をもとに検出していると解釈し, その可能性を考察する．本研究をソフトウェア研究における事例研究と捉え, 以下の研究課題を定義した．

- RQ1. GNN モデルの設計と試作
- RQ2. 作成した GNN モデルの検出可能性の確認
- RQ3. 既存の研究成果との比較, その優越性の検証

2　先行研究

2.1　Chakraborty らの研究 [2]

Chakraborty らは, 実際の脆弱性を高精度で検出することを目的とした研究を行った． AST(Abstract Syntax Tree) と PDG(Program Dependence Graph) を組み合わせた CPG(Code Property Graph) を入力とした GGNN(Gated Graph Neural Network) を用いて特徴ベクトルを抽出．抽出した特徴ベクトルを多層 NN を用いて脆弱性を検出するモデルを提案した．テスト時の精度は先行研究における最良のモデルと比較して,33.57% 向上した．

2.2　Cao らの研究 [3]

Cao らはメモリ関連の脆弱性を検出することを目的とした．この研究では手続き間の詳細な情報を捉えるために PDG を CG(Call Graph) で拡張したグラフを生成．生成したグラフを FS-GNN(Flow-Sensitive Graph Neural Network) で学習するモデルを提案した．本モデルはメモリ関連の脆弱性を先行研究に比べ, 高い精度で検出した．

3　課題解決へのアプローチ

本研究では, 目的達成への第一歩として標準的な特定の脆弱性を検出できる GNN モデルを設計, 検証し, その上で拡張可能性を考察する．関連技術を図示すると図 1 のようになる．従来の静的プログラム解析による脆弱性検出では, 事前に定義された脆弱性のパターンやルールに基づいて検出を行なう決定論的アプローチを採用している．この手法

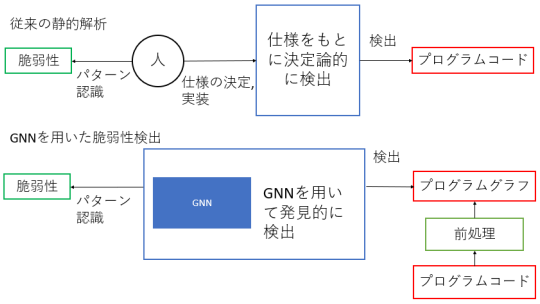

図 1　アプローチ

では, 新たなパターンの脆弱性, 動的意味による脆弱性を検出できないという問題がある．本研究では,NN モデルとして GNN を用いる．プログラムは本質的にグラフ構造であり, プログラムの構造的な特徴を捉えるために, グラフ処理能力に長けた GNN を採用する．GNN への入力に使うグラフには静的意味と構文構造の両方を考慮して特徴量の抽出を行なう必要があるので,AST を選択した．学習した GNN モデルは静的解析では検出できないパターンも発見的に検出できる可能性があると考えた．RQ1 では, 実際に GNN のモデルを設計, 試作する．RQ2 では,RQ1 で試作したモデルを用いて実験を行い,GNN が脆弱性検出において有効であると確認する．RQ3 では,RQ2 の結果と, 先行研究と比較, 考察する．

4　提案モデル

4.1　モデルの構成

GCN の設計は入力層, 畳み込み層 1 層, プーリング層, 全結合層 2 層, 出力層とし, 活性化関数には ReLu 関数を用いた． プーリングには最大値プーリング, 全結合層は Dense 層とし, 出力層は sigmoid 関数を用いた．畳み込み層は 2 層になると, 過平滑化 (Over-Smoothing) してしまうため,1 層とした．Relu 関数を活性化関数として用いることで, 非線形性を導入し, ネットワークの表現力を向上できる．最大値プーリングを行い、特徴ベクトルのサイズを削減することで, 計算量を減らしつつ重要な特徴を保持し, 情報を集約する．Dense 層を置くことでノード間の全結合を可能にし, ネットワークの表現力が向上する．sigmoid 関数は,0 から 1 の範囲で値を出力する活性化関数である．出力を各クラスである確率として解釈できることから,2 クラス分類において主流であり, 本モデルでもこれを採用した．

4.2　GNN の構造

GNN で学習するグラフには AST を用いる． AST はプログラムコードの静的意味と構文的な構造を持つ．ま

た,AST におけるノードの属性を特徴量とし,GCN を学習させることで動的意味を解釈できると考えた. 以上より,AST は構文的な構造に加え, 静的と動的両方の情報をとらえることが出来ると考え,AST を用いる.

4.3 ノードの特徴量

ノードの特徴量には,AST のノードの型と属性を用いる. ノードごとに対応する型と属性を word2vec を用いて 100 次元のベクトルに変換し, 特徴量とした. 特徴量のサイズを合わせるために, 一部の値は 0 とした (0-padding).

5 実験

実験に使用するデータセットに関して,NIST が公開している,SATE IV Juliet や他のいくつかのデータセット [6],CVEfixes[5],github 上のオープンソースプロジェクトのプログラムコードを収集し, データセットとした. サンプル数は 16,544 であり, そのうち,49.6% のプログラムに脆弱性が含まれている.

上記のデータセットを学習用とテスト用に 8:2 の割合で分割し検証を行った. テスト時の精度は 98.67%,F1 値は 0.9863 であった. 学習時の損失と F1 値の推移を示す.

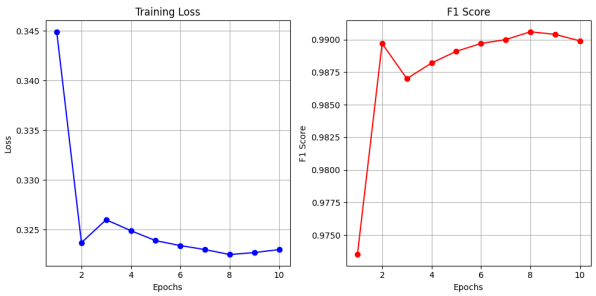

図 2 学習時の損失と F1 値の推移

6 考察

6.1 結果の考察

テスト時には精度 98.67%,F1 値 0.9863 という値が得られた. この結果から, 提案モデルをデータセットで学習させることで, データセットから暗黙に定義した仕様を基に検出が行えたといえる. AST から静的意味, 動的意味に関連する構造的なパターンを学習し, 検出する GNN モデルが作成できることを確認できた. 本研究のテスト結果は先行研究 [2],[3] と比較すると高いが, これは, データセットの質やサンプルの粒度の違いが結果に影響したと考えられる.

6.2 モデルの考察

本研究では java を対象としたが, 他のオブジェクト指向言語に対しても同様に検出できると考える. オブジェクト指向言語における脆弱性は, 関数やメソッド, オブジェクトの相互作用から生じる場合が多く, 言語ごとの構文に依存するものではない. ある言語の脆弱性を検出できるモデルは別の言語に対しても対応することが可能であり, 一般的な脆弱性に関しては言語の違いによる結果の差は小さい. よって, データセット次第で Java 以外の言語に対しても適応できる.

様々な属性を特徴量として用いることで, プログラムの動的な振る舞いやデータの流れを考慮した脆弱性の検出ができる. AST におけるノードには変数に関係するもの, 制御構造に関係するものがある. これらは, プログラムのデータフローや制御フローを示す情報であり, プログラムの挙動や制御の流れを捉える上で重要な情報となる.

7 おわりに

本研究では,NN を用いた脆弱性検出の可能性の考察を目的に研究を行った. アプローチとして GCN で AST を学習させるという手法をとった. 結果としては比較的高い精度と F1 値が得られた. 作成したモデルをデータセットを用いて学習させることで, データセットから暗黙に定義された仕様をもとにした検出が行えた. AST を GCN の入力とすることで, 構造パターンから静的意味や動的意味を考慮して脆弱性の検出が行えたと考察した. AST から構造パターンを学習することで, そのパターンを検出する静的解析ツールに代わる NN モデルを作成できることが確認できた. 今後の課題として, データセットを変更しての, テスト, 再学習があげられる. 実際に発見された脆弱性を訓練データとして用いてモデルを学習させることで, より実用的な脆弱性検出モデルができることが期待される.

参考文献

[1] J.Zhou et al.: “Graph neural networks: A review of methods and applications,” *AI Open, Volume 1*, Pages 57-81, 2020

[2] T.kipf & Max Welling: “Semi-supervise classification with graph convolutional networks.” *Proc. ICLR*, 2017

[3] S.Chakraborty et al.: “Deep Learning based Vulnerability Detection:Are We There Yet?,” *IEEE Transaction on Software Engineering,VOL.TBD*, 2020

[4] S.cao et al.: “MVD:Memory Relaated Vulnerability Detection Based on Flow-Sensitive Graph Neural Network,” *IEEE/ACM 44th Int. Conf. Software Engineering*, 2022

[5] G.Bhandari & L.Moonen: “CVEfixes: automated collection of vulnerabilities and their fixes from open-source software,” *Proc. 17th Int. Conf. Predictive Models and Data Analytics in Software Engineering*, Page 30-39, 2021

[6] NIST: Test Suites, https://samate.nist.gov/SARD/test-suites