
Eduard Pinconschi
FEUP, University of Porto, Portugal

Rui Abreu
FEUP, University of Porto, Portugal

Hakan Erdogmus
Carnegie Mellon University, USA

Corina S. Păsăreanu
Carnegie Mellon University, USA

Sofia Reis
INESC-ID, IST, University of Portugal, Portugal

Chi Zhang
Carnegie Mellon University, USA

Abstract—Software vulnerability detection (SVD) aims to identify potential security weaknesses in software. SVD systems have been rapidly evolving from those being based on testing, static analysis, and dynamic analysis to those based on machine learning (ML). Many ML-based approaches have been proposed, but challenges remain: training and testing datasets contain duplicates, and building customized end-to-end pipelines for SVD is time-consuming. We present Tenet, a modular framework for building end-to-end, customizable, reusable, and automated pipelines through a plugin-based architecture that supports SVD for several deep learning (DL) and basic ML models. We demonstrate the applicability of Tenet by building practical pipelines performing SVD on real-world vulnerabilities.

I. INTRODUCTION

Software vulnerability detection (SVD) approaches identify potential security weaknesses in software [1], [2]. Over the past years, SVD techniques evolved from traditional approaches, such as testing, static analysis, and dynamic analysis, to machine learning (ML), which promises to enable faster analysis with increased precision earlier in the software development life-cycle (i.e., “shift-left security”).

Current ML-based approaches combine a multitude of code embeddings and network topologies [3]. They address various research problems in SVD, including methods, features, and datasets [4]. However, many challenges remain. First, the training and testing data often contain duplicates (up to 68%)—due to poor cleansing and sampling techniques—which artificially inflates detection performance [5], [6]. Second, there is a lack of effective, openly available, and automated MLOps pipelines to produce flexible ML systems [7]. ML-based systems should become end-to-end tools capable of analyzing a diversity of projects with respect to applicable platforms, programming languages, and software domains [3].

To address these challenges, we propose Tenet, a modular framework for building end-to-end, customizable, reusable, and automated pipelines for SVD. Tenet provides accessible implementations and supports reproducible evaluation of several SVD approaches. It also allows automatic construction of vulnerability datasets. Tenet makes building new SVD systems practical as it can easily combine real-world data with state-of-the-art SVD and ML-based models. We achieve that with a plug-and-play architecture with a set of core components and plugins for data collection, pre-processing, learning, analysis, and evaluation. Tenet is available at https://github.com/TQRG/tenet.

II. TENET DESCRIPTION

A. Tenet components

Figure 1 depicts the various components of Tenet, which can be organized into different pipelines to generate ML-based SVD systems. The components are: (1) data collector to gather software vulnerabilities source code from vulnerability databases (e.g., references to commits) listed in NVD [8] and OSV [9] databases; (2) data labeler to label code fragments as safe and unsafe using diff analysis (DA), i.e., based on code changes between vulnerable and safe (fixed) versions of a code fragment, or static analysis (SA) information; (3) granularity adjuster to capture the desired context or scope around a vulnerability’s location; (4) code mutator to extend the set of unsafe samples with code changes using variable and function name randomization to balance the normally highly-imbalanced training datasets [5]; (5) representation extractor to gather different representations such as context-path, functions and lines of code; (6) data pre-processor and sampler to clean duplicate samples and generate stratified datasets; (7) model generator to train the model from the data collected; and (8) statistics generator to collect different metrics and plot class separability charts. Tenet currently supports multiple models, including powerful state-of-the-art code models such as code2vec [10] and CodeBERT [11]. Other models can be easily integrated as well.

B. Implementation

Tenet follows a plug-and-play architecture with a CDD (Configuration-Driven Development) strategy to assemble a set of components (implemented as plugins) into a pipeline through a configuration file written in YAML. A plugin is a Python script extending a generic handler of Tenet with specific functionalities. A plugin example to scrape files from GitHub is shown here: https://t.ly/gpiU. An example configuration file that specifies a pipeline (https://t.ly/S2hE) and a tutorial (https://t.ly/_lm) are available at our GitHub repo.
C. Tenet Usage

To illustrate the flexibility, re-usability, and robustness of Tenet, we present two different pipelines to produce different SVD models for JavaScript (JS) vulnerability detection: PL1, which uses diff analysis to label data for multiple types of vulnerabilities and produces five SVD systems with distinct network topologies (Table I); and, PL2, which produces two SVD systems with similar network topology (CodeBERT) but fine-tuned for different JS vulnerabilities types (Table II). The dataset used as input for PL2 was generated and labeled with an external pipeline which shows how one can produce a dataset independently and plug it into our pipeline. We tested the pipelines by deploying them on different platforms: (1) Debian Linux 5.10 (40 cores, 64Gb, Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz); (2) Standard NC24 (24 cores, 224 GB RAM, 1440 GB disk) with a GPU-4xNVIDIA Tesla K80 on Microsoft Azure. This demonstrates Tenet’s robustness with respect to different deployment scenarios.

III. CONCLUSION

We presented Tenet, a component-based framework for building cross-platform ML- and DL-based pipelines for SVD. The framework targets SVD researchers and practitioners. It currently has 2 data sources, 6 handlers, and 16 plugins that enable a broad combination of pipelines. Furthermore, the plugin-based approach facilitates extensions with more sophisticated models and code representations. Experiments showed that the framework promotes re-usability by executing pipeline variations on two different platforms. We plan to improve the labeling process with human feedback and build a monitoring component.

REFERENCES


| TABLE I |
| --- | --- | --- | --- | --- | --- |
| Model | Acc. | Prec. | Recall | F1 | Training Time |
| ADA | 0.938 | 0.071 | 0.019 | 0.031 | 7.77s |
| KNN | 0.950 | 0.0 | 0.0 | 0.0 | 47.27s |
| SVC | 0.950 | 0.0 | 0.0 | 0.0 | 11.64s |
| RFC | 0.949 | 0.0 | 0.0 | 0.0 | 38.05s |
| code2vec | 0.960 | 0.727 | 0.314 | 0.438 | 1m:8s |

| TABLE II |
| --- | --- | --- | --- | --- | --- |
| Dataset | Acc. | Prec. | Recall | F1 | Training Time |
| Path injection | 0.991 | 0.375 | 1 | 0.546 | 1h:25m |
| DOM XSS | 0.988 | 0.667 | 0.4375 | 0.5284 | 9h:28m |

Tables I and II show the performance results for different models trained on different datasets. In PL1, we plugged in standard machine learning models (ADA, KNN, SVC, RFC) and a specialized model (code2vec), which re-used the same pipeline. In PL2, we used CodeBERT as the model, and different data preparation and collection components that rely on static analysis for Path Injection and DOM XSS for ground truth, giving rise to an entirely different pipeline.

Fig. 1. Tenet architecture with two short pipeline examples. The first pipeline uses lines of code as the granularity of a vulnerability and labels the samples using static analysis with CodeQL. The second pipeline uses functions as the granularity of a vulnerability and generates labels using diff analysis.