

BÁO CÁO TỔNG KẾT ĐỒ ÁN MÔN HỌC

Môn học: **Lập trình an toàn và khai thác lỗ hổng phần mềm**

Tên chủ đề: Practical Automated Detection of Malicious npm Packages

Mã nhóm: G03, Mã đề tài: CK32

Lớp: NT521.P12.ANTT

1. THÔNG TIN THÀNH VIÊN NHÓM:

(Liệt kê tất cả các thành viên trong nhóm)

STT	Họ và tên	MSSV	Email
1	Ngô Hồng Phúc	22521124	22521124@gm.uit.edu.vn
2	Nguyễn Tài Hiếu	22520442	22520442@gm.uit.edu.vn
3	Nguyễn Việt Hoàng	22520471	22520471@gm.uit.edu.vn
4	Võ Nguyễn Thái Học	22520489	22520489@gm.uit.edu.vn

2. TÓM TẮT NỘI DUNG THỰC HIỆN:

A. Chủ đề nghiên cứu trong lĩnh vực An toàn phần mềm:

- Phát hiện lỗ hổng bảo mật phần mềm
- Khai thác lỗ hổng bảo mật phần mềm
- Sửa lỗi bảo mật phần mềm tự động
- Lập trình an toàn
- Khác:

B. Tên bài báo tham khảo chính:

Practical Automated Detection of Malicious npm Packages

C. Dịch tên Tiếng Việt cho bài báo:

Phát hiện Tự động Thực tế các Gói npm Độc hại

D. Tóm tắt nội dung chính:

Kho lưu trữ npm là một nền tảng quan trọng của hệ sinh thái JavaScript và TypeScript, chứa hơn 1,7 triệu gói. Với sự phổ biến rộng rãi, npm trở thành mục tiêu của các tác nhân độc hại, những kẻ phát tán mã độc qua các gói mới hoặc xâm nhập vào các gói đã có. Các mã độc này có thể đánh cắp dữ liệu nhạy cảm từ người dùng khi họ cài đặt các gói này hoặc những gói phụ thuộc vào chúng.

Việc bảo vệ chuỗi cung ứng phần mềm khỏi những cuộc tấn công như vậy là rất quan trọng, nhưng lượng lớn các bản cập nhật và gói mới khiến việc kiểm tra thủ công trở nên không khả thi. Để giải quyết vấn đề này, bài viết giới thiệu Amalfi, một phương pháp học máy tự động phát hiện các gói có nguy cơ độc hại, bao gồm 3 kỹ thuật hỗ trợ.

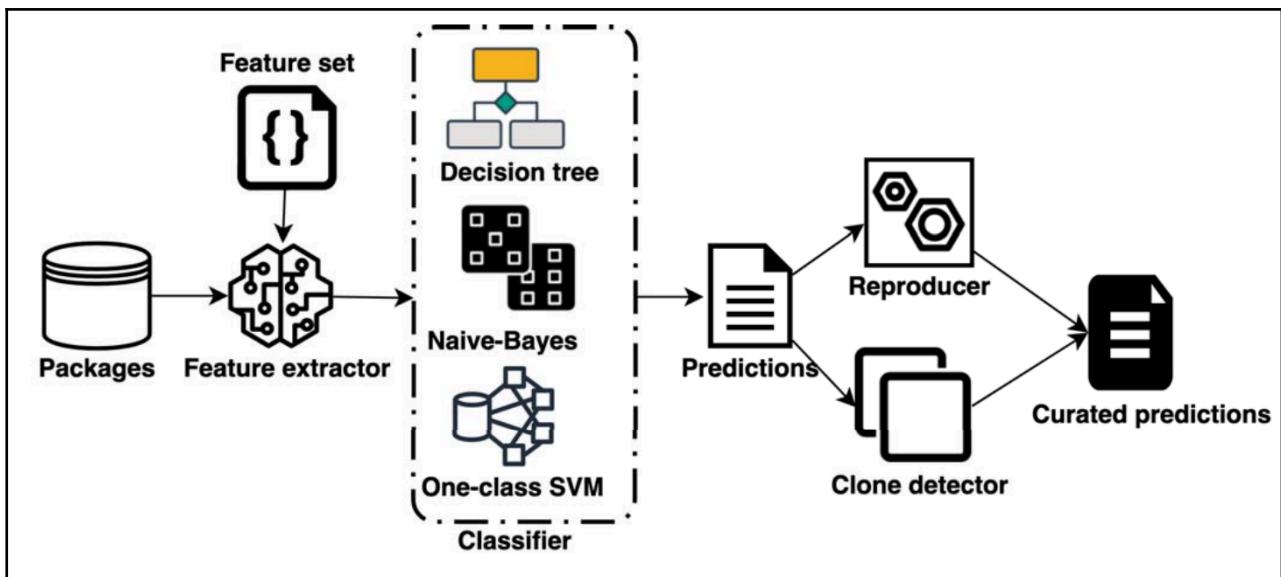
Đầu tiên, Amalfi sử dụng các bộ phân loại đã được huấn luyện với dữ liệu của các gói độc hại và an toàn đã biết. Nếu một gói bị đánh dấu độc hại, hệ thống tiếp tục kiểm tra xem gói đó có chứa metadata về kho mã nguồn và có thể tái tạo từ mã nguồn không. Các gói có thể tái tạo từ mã nguồn thường không độc hại, giúp giảm số lượng cảnh báo sai. Cuối cùng, kỹ thuật phát hiện mã sao chép được sử dụng để tìm các bản sao của gói độc hại, giảm thiểu cảnh báo sai âm.

Amalfi có hiệu quả vượt trội nhờ tính nhẹ nhàng và tốc độ nhanh, chỉ mất vài giây để trích xuất đặc trưng và chạy các bộ phân loại. Khi thử nghiệm trên 96.287 phiên bản gói trong một tuần, Amalfi đã phát hiện 95 mẫu mã độc mới với số lượng cảnh báo sai chấp nhận được, chứng tỏ hiệu quả và độ tin cậy cao của phương pháp.

E. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong bài báo:

Amalfi kết hợp 3 kỹ thuật bổ sung vào một hệ thống, bao gồm:

- **Machine-learning classifiers:** các bộ phân loại học máy được huấn luyện trên các ví dụ gói độc hại và an toàn đã được gán nhãn, sử dụng các tính năng ghi lại các thay đổi trong API mà gói sử dụng cũng như metadata gói được trích xuất bằng cách quét cú pháp nhẹ.
- **Reproducer:** xây dựng lại gói từ mã nguồn và so sánh kết quả với phiên bản đã công bố trong kho lưu trữ.
- **Clone detector:** tìm bản sao của các gói độc hại đã biết.



F. Môi trường thực nghiệm của bài báo:

- Ngôn ngữ lập trình để hiện thực phương pháp:
 - + **Machine-learning classifiers:** Python triển khai các thuật toán học máy (decision-tree, naive-bayes, SVM...). Cung cấp script để có thể tùy chỉnh thông qua các dòng lệnh.
 - + **Reproducer:** Shell script được dùng để tái tạo gói npm từ mã nguồn, so sánh kết quả với gói công khai để kiểm tra tính hợp lệ.
 - + **Clone detector:** Python được dùng để tính toán hash MD5 cho các gói npm, hỗ trợ phát triển các bản sao của các gói độc hại đã biết.
- Đối tượng nghiên cứu: các gói npm (Node Package Manager)
 - + Các gói npm độc hại và lành tính: Xác định những gói có chứa mã độc hại hoặc có hành vi đáng ngờ. Nghiên cứu cũng bao gồm các gói lành tính để huấn luyện các bộ phân loại.
 - + Các gói npm có bản sao (clones): Phát hiện các gói sao chép từ các gói độc hại đã biết. Mục tiêu là xác định và ngăn chặn các gói giả mạo sử dụng mã độc hại đã tồn tại.
- Dữ liệu và tài nguyên:
 - + **npm registry:** Nơi thu thập dữ liệu về các gói npm công khai (gồm các gói độc hại và lành tính).
 - + **MalOSS Dataset:** Bộ dữ liệu được sử dụng để kiểm tra hiệu suất của phương pháp.
- Tiêu chí đánh giá tính hiệu quả của phương pháp:
 - + **Độ chính xác (Accuracy):** Phần trăm tổng số gói được phân loại chính xác (cả gói độc hại và không độc hại).

- + **Hiệu suất thời gian (Performance Timing):** Thời gian cần thiết để phát hiện gói độc hại.
- + **Tính khả thi của công cụ Reproducer:** Tỷ lệ gói có thể tái tạo từ mã nguồn thành công.
- + **Phát hiện bản sao độc hại:** Khả năng phát hiện chính xác các gói npm là bản sao của gói độc hại đã biết.

G. Kết quả thực nghiệm của bài báo:

- Kết quả thực nghiệm: phát hiện 95 gói độc hại chưa từng được biết đến trước đây.
- Nhận xét:
 - + **Khả năng:** Amalfi có khả năng phát hiện gói phần mềm độc hại nhanh chóng nhờ vào các thành phần như classifier, reproducer và clone detector. Nó giảm thiểu tỷ lệ dương tính giả và giúp nhận diện các gói độc hại chưa được phát hiện.
 - + **Ưu điểm:** Tốc độ xử lý cao, chi phí huấn luyện thấp và hiệu quả trong việc giảm dương tính giả.
 - + **Nhược điểm:** Có thể bỏ sót một số gói độc hại do giới hạn thời gian kiểm tra và không thể xem xét chi tiết mỗi gói.

H. Công việc/tính năng/kỹ thuật mà nhóm thực hiện lập trình và triển khai cho demo:

Dựa trên phương pháp và mô hình phát hiện tự động gói npm độc hại và source code có sẵn nhóm đã tự tạo ra chương trình trích xuất các đặc tính từ gói npm, các đặc tính này là các đặc tính có thể cho thấy các gói npm có phải là độc hại hay không, và từ các đặc tính này sẽ dùng làm dữ liệu để train cho mô hình học máy. Vì đây là chương trình quan trọng nhất của mô hình này nên tác giả của bài báo khoa học này đã không public để không bị các kẻ xấu lợi dụng và để phát triển cho sản phẩm của họ nên chúng em đã tự tạo ra chương trình trích xuất cho riêng mình. Công việc tiếp theo là thu thập các gói npm độc hại cũng như lanh tính để tạo thành các kho dữ liệu để thực hiện việc training cũng như là thực nghiệm phát hiện gói tin độc hại. Sau đó là tạo các chương trình để hoàn thiện quá trình thực nghiệm mô hình như chương trình vận dụng các mô hình đã được train từ train_classifier để dự đoán gói npm, tạo chương trình phát hiện phiên bản clone: clone_detector.

I. Các khó khăn, thách thức hiện tại khi thực hiện:

- Khó tìm kiếm các gói npm độc hại.
- Tốn nhiều thời gian tạo dữ liệu.
- Không được cung cấp code trích xuất đặc tính.

3. TỰ ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH SO VỚI KẾ HOẠCH THỰC HIỆN:

100%

4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Tìm các gói npm độc hại cho bộ dữ liệu	Ngô Hồng Phúc
2	Tìm các gói npm lành tính cho bộ dữ liệu	Nguyễn Việt Hoàng
3	Feature extractor, Prediction, Reproducer	Võ Nguyễn Thái Học
4	Clone detector	Ngô Hồng Phúc
5	Đánh giá các thuật toán cho mô hình	Nguyễn Tài Hiếu
6	Thí nghiệm 1	Ngô Hồng Phúc Võ Nguyễn Thái Học
7	Thí nghiệm 2	Nguyễn Tài Hiếu Nguyễn Việt Hoàng

MỤC LỤC

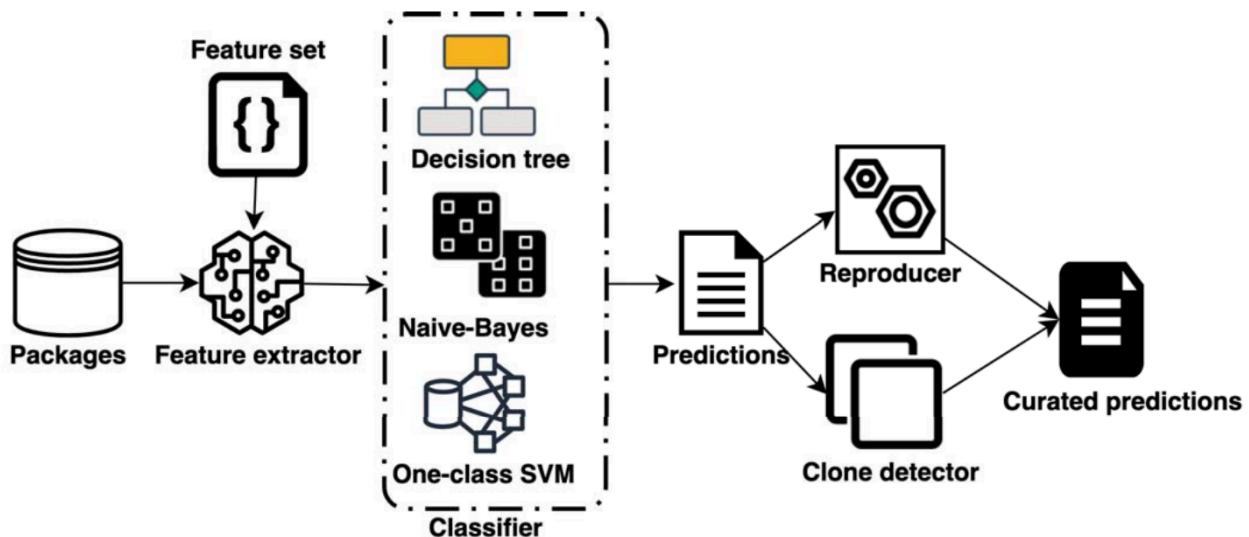
A. Phương pháp thực hiện.....	7
A.1. Luồng thực thi.....	7
A.2. Thành phần của hệ thống trong bài báo.....	7
A.3. Thành phần mà nhóm đã thực hiện.....	13
B. Chi tiết cài đặt, hiện thực.....	27
B.1. Chuẩn bị dữ liệu.....	27
B.2. Thí nghiệm 1: Phân loại các gói mới xuất bản.....	32
B.3. Thí nghiệm 2: Phân loại dữ liệu đã được dán nhãn.....	39
C. Kết quả thực nghiệm.....	42
D. Hướng phát triển.....	43
D.1. Tự động hóa toàn bộ quy trình.....	43
D.2. Đề xuất gói lành tính thay thế.....	43
D.3. Phát triển thành công cụ hoặc GUI.....	44
YÊU CẦU CHUNG.....	44

BÁO CÁO TỔNG KẾT CHI TIẾT

Phần bên dưới của báo cáo này là tài liệu báo cáo tổng kết - chi tiết của nhóm thực hiện cho đề tài này.

A. Phương pháp thực hiện

A.1. Luồng thực thi



- Thành phần đầu tiên của mô hình là feature extractor, đây là một chương trình dùng để trích xuất các đặc tính từ gói npm, ta sẽ tạo 1 danh sách các đặc tính mà ta muốn lấy từ gói npm, những đặc tính này là những đặc tính có thể dùng để xác định xem gói npm đó có độc hại hay không. Ví dụ như ta kiểm tra số từ keyword đáng nghi trong gói npm là bao nhiêu, ví dụ là 50 thì đặc tính suspicious_keyword của gói npm đó sẽ là 50, cứ như vậy ta sẽ tổng hợp được thông tin các đặc tính của gói npm. Chương trình feature extractor này sẽ trích xuất đặc tính của hàng loạt các gói npm sau đó tổng hợp nó thành 1 folder với đặc tính của 1 gói npm sẽ nằm trong 1 file csv riêng. Folder này được gọi là training set, là đầu vào của thành phần tiếp theo là train classifier.
- Thành phần tiếp theo là train classifier, thành phần này sẽ dùng các đặc tính của các gói npm đã được trích xuất và tạo thành training set ở thành phần trên làm dữ liệu đầu vào để train mô hình học máy, training set mà ta tạo ở trên được trích xuất từ danh sách cả gói npm độc và không độc và chúng ta đã biết trước là gói nào là độc hại, gói nào lành tính. Ta sẽ cho chương trình train classifier này biết gói nào là độc hại và lành tính, sau đó nó sẽ sử dụng các thuật toán học máy như decision tree, naive - bayes và svm để tìm xem các sự khác biệt giữa các đặc tính ở các gói npm độc hại và lành tính từ đó tạo thành mô hình đã train. Mô hình này sẽ dùng để dự đoán các gói npm mà ta chưa biết là độc hại hay không.
- Thành phần tiếp theo là prediction, thành phần này sẽ sử dụng các mô hình đã được train ở thành phần trước, vì ta có nhiều thuật toán học máy nên với mỗi thuật toán ta

sẽ tạo 1 mô hình trained riêng, sau đó ta sẽ sử dụng các mô hình này để dự đoán các gói npm mà ta muốn dự đoán. Sau khi dự đoán xong ta sẽ có file csv, là kết quả dự đoán từ ba mô hình.

- Cùng lúc sử dụng các mô hình đã trained để dự đoán gói npm như ở thành phần trên ta sẽ dùng chương trình clone detector để xem xem các gói npm ta muốn dự đoán có gói nào là bản sao của các gói độc hại mà ta đã biết không. Kết quả của chương trình này cũng là 1 file csv. Từ file csv này và ba file csv kết quả từ thành phần trên ta sẽ tổng hợp thành 1 file csv kết quả mà thành phần của file này sẽ là các gói npm có kết quả là độc hại trong ít nhất 1 file csv trước. File kết quả tổng hợp này sẽ là đầu vào cho thành phần cuối là reproducer.
- Thành phần reproducer này sẽ sử dụng tổng hợp danh sách các gói npm được dự đoán là độc hại ở thành phần trên để tiếp tục lọc qua 1 lần nữa để có kết quả cuối cùng. Mục tiêu của thành phần này sẽ là cố gắng tìm cách tái tạo là source code của các gói npm có trong danh sách vì bình thường các gói npm độc hại thì source code của nó sẽ bị ẩn đi và không thể nào tái tạo được, vì thế nếu như ta tái tạo lại được source code của nó thì khả năng rất cao là gói npm đó không phải gói độc hại. Sau chương trình ở thành phần này ta sẽ tổng hợp lại 1 danh sách cuối cùng bao gồm các gói npm được dự đoán là độc hại trong số các gói mà ta muốn kiểm tra.

A.2. Thành phần của hệ thống trong bài báo

A.2.1. Classifier

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Author: http://www.mikell.org
# Date: 2015-07-10
# Version: 0.1
# License: MIT
# Description: This script processes a directory containing training sets for a classifier.
# It reads 'change-features.csv' files from each directory and creates a feature dictionary.
# The dictionary contains features and their values, which are then used to train a classifier.

import os
from collections import defaultdict
import csv

# Initialize a dictionary to store features and their values
feature_dict = defaultdict(float)

# Function to process a single file
def process_file(file):
    global feature_dict
    reader = csv.reader(open(file))
    next(reader) # Skip the first row
    for row in reader:
        feature, value = row
        value = float(value)
        if positive and value < 0:
            value = 0
        if booleanize:
            value = 1 if value > 0 else 0
        if feature not in exclude_features:
            feature_dict[feature] = value

# Main loop to process all files in all training sets
for training_set_dir in training_sets:
    for root, _, files in os.walk(training_set_dir):
        for f in files:
            if f == "change-features.csv" and f"{root}" not in leave_out:
                package = os.path.relpath(
                    os.path.dirname(root), training_set_dir)
                version = os.path.basename(root)
                print(f"Processing {package}@{version}")
                # Load features for this package
                with open(os.path.join(root, f), "r") as feature_file:
                    # First, read features into a dictionary
                    feature_dict = {}
                    reader = csv.reader(feature_file)
                    next(reader) # Bỏ qua dòng đầu tiên
                    for row in reader:
                        feature, value = row
                        value = float(value)

                        if positive and value < 0:
                            value = 0
                        if booleanize:
                            value = 1 if value > 0 else 0
                        if feature not in exclude_features:
                            feature_dict[feature] = value

# Assign indices to any features we have not seen before
for feature in feature_dict.keys():
    if feature not in feature_names:
        feature_names.append(feature)
```

Đồ án môn học – NT521 - Báo cáo tổng kết

```
❖ train_classifier.py
96     # convert the feature dictionary into a feature vector
97     feature_vec = []
98     for feature, value in feature_dict.items():
99         idx = feature_names.index(feature)
100        if idx >= len(feature_vec):
101            feature_vec.extend(
102                [0] * (idx - len(feature_vec) + 1))
103            feature_vec[idx] = value
104
105        # add the feature vector to the training set
106        training_set.append(feature_vec)
107
108        # add the label to the labels list
109        label = "benign"
110        if hashing:
111            hash_file = os.path.join(root, "hash.csv")
112            if os.path.isfile(hash_file) and os.path.getsize(hash_file) > 0:
113                with open(hash_file, "r") as rfi:
114                    hash_res = csv.reader(rfi).next()[0]
115                    if hash_res in malicious:
116                        label = "malicious"
117                    else:
118                        if (package, version) in malicious:
119                            label = "malicious"
120                    labels.append(label)
121        if label == "malicious" and randomize == True:
122            malicious_len += 1
123
124        # normalize length of feature vectors by extending with zeros
125        num_features = len(feature_names)
126        for i in range(len(training_set)):
127            length = len(training_set[i])
128            if length < num_features:
129                training_set[i].extend([0] * (num_features - length))
130
```

- Chương trình sẽ load file feature của gói npm trong training sets và tạo thành vector đặc tính cho gói npm đó, cũng như kiểm tra gói npm đó là thuộc loại độc hại hay lành tính, thực hiện việc này với toàn bộ các gói npm trong training sets, ta sẽ được 2 list là training_set và labels với training_set chứa các vector đặc tính của các gói npm, labels chứa nhãn độc hại hoặc lành tính của gói npm. Từ hai list trên ta sẽ tạo mô hình dự đoán với các thuật toán học máy khác nhau

```
❖ train_classifier.py
146     # train the classifier
147     if classifier == "decision-tree":
148         classifier = tree.DecisionTreeClassifier(criterion="entropy")
149         classifier.fit(training_set, labels)
150     elif classifier == "random-forest":
151         classifier = RandomForestClassifier(criterion="entropy")
152         classifier.fit(training_set, labels)
153     elif classifier == "naive-bayes":
154         classifier = naive_bayes.BernoulliNB()
155         classifier.fit(training_set, labels)
156     else:
157         classifier = svm.OneClassSVM(
158             gamma='scale', nu=nu, kernel='linear')
159         classifier.fit([datum for i, datum in enumerate(
160             training_set) if labels[i] == "benign"])
161     end = timer()
162     diff = timedelta(seconds=end-start)
163
164     if performance is not None:
165         with open(performance, "a+") as wfi:
166             writer = csv.writer(wfi)
167             writer.writerow([diff])
168
169     # render the tree if requested; only applicable for decision trees
170     if classifier == "decision-tree" and render:
171         file, ext = os.path.splitext(render)
172         if ext != ".png":
173             print("Rendering tree to PNG requires a file name ending in .png")
174             exit(1)
175         outfile = Source(tree.export_graphviz(
176             classifier, out_file=None, feature_names=feature_names), format="png")
177         outfile.render(file, view=view, cleanup=True)
178
179     # store the classifier and metadata
180     with open(output, "wb") as f:
```

- Sau đó lưu kết quả vào file pickle

```

1 if ext != '.png':
2     print("Rendering tree to PNG requires a file name ending in .png")
3     exit(1)
4 outfile = Source(tree.export_graphviz(
5     classifier, out_file=None, feature_names=feature_names), format="png")
6 outfile.render(file, view=view, cleanup=True)
7
8 # store the classifier and metadata
9 with open(output, "wb") as f:
10    pickle.dump({
11        "feature_names": feature_names,
12        "booleanize": booleanize,
13        "positive": positive,
14        "classifier": classifier
15    }, f)
16
17 if __name__ == "__main__":
18     argparse = argparse.ArgumentParser(
19         description="Train a classifier")
20     argparse.add_argument(
21         "classifier", help="Type of classifier to be trained.", choices=["decision-tree", "random-forest", "naive-bayes", "svm"])

```

A.2.2. Reproducer

- Tác giả của bài báo cung cấp cho ta 2 file shell script gồm reproduce-package.sh và build-package.sh
- Với script đầu tiên, chương trình sẽ phân tích tham số đầu vào có dạng <package-name>@<package-version>, sau đó sẽ thực hiện lấy URL Git từ npm registry dựa trên thuộc tính repository.url, repository, hoặc homepage và git clone mã nguồn về và xác định commit hoặc branch tương ứng với phiên bản được yêu cầu từ npm registry. Sau đó chạy tệp script build-package để thực hiện xây dựng gói từ mã nguồn đã clone.

```

$ reproduce-package.sh
1  #! /bin/bash
2
3  set -ex
4
5  SCRIPT_DIR=$( cd "$( dirname "${BASH_SOURCE[0]}" )" &> /dev/null && pwd )
6
7  spec="$1"
8  outdir=$(readlink -f "$2")
9
10 package="${spec%@*}"
11 version="${spec##*@}"
12
13 echo "Trying to reproduce package $package at version $version."
14
15 # Clone repo
16 for prop in repository.url repository.homepage; do
17     repoUrl=$(npm view "$spec" "$prop")
18     if ! [ -z "$repoUrl" ]; then
19         break
20     fi
21 done
22 if [ -z "$repoUrl" ]; then
23     echo "Could not find git repository for $spec."
24     exit 1
25 fi
26 repoUrl=$(node -e "console.log(require('./node_modules/normalize-git-url')('$repoUrl').url)")
27 # replace ssh:// with https://
28 repoUrl=$(echo "$repoUrl" | sed 's#^ssh://#https://#')
29 git clone "$repoUrl" working
30
31 # Check out right commit
32 cd working
33 ref=$(npm view "$spec" gitHead)
34 if [ -z "$ref" ]; then
35     # typical branch names for $version

```

Đồ án môn học – NT521 - Báo cáo tổng kết

```
$ reproduce-package.sh
25   fi
26   repoUrl=$(node -e "console.log(require('./node_modules/normalize-git-url')('$repoUrl').url)")
27 # replace ssh:// with https://
28 repoUrl=$(echo "$repoUrl" | sed 's#^ssh://#\https://#')
29 git clone "$repoUrl" working
30
31 # Check out right commit
32 cd working
33 ref=$(npm view "$spec" gitHead)
34 if [ -z "$ref" ]; then
35   # typical branch names for $version
36   candidate_refs="$version v$version v-$version"
37   # if $version contains pre-release tags, try those as well; they might be shas
38   if [[ "$version" == *-* ]]; then
39     candidate_refs="$candidate_refs $(echo ${version##*-} | sed 's/[.-]/ /g')"
40   fi
41
42   for candidate_ref in $candidate_refs; do
43     ref=$(git rev-parse --verify "$candidate_ref" 2>/dev/null || echo "")
44     if ! [ -z "$ref" ]; then
45       break
46     fi
47   done
48
49   if [ -z "$ref" ]; then
50     echo "Could not find source commit for version $version; trying HEAD."
51     ref=HEAD
52   fi
53 fi
54 git checkout "$ref"
55
56 # Build package
57 "$SCRIPT_DIR/build-package.sh" "$package" . "$outdir"
58
```

- File script build-package.sh sẽ có nhiệm vụ là xây dựng gói npm từ mã nguồn mà ta đã clone ở trên

```
$ build-package.sh
1 #!/bin/bash
2
3 set -ex
4
5 if [ $# -ne 3 ]; then
6   echo "Usage: $0 <package-name> <working-dir> <out-dir>"
7   exit 1
8 fi
9
10 pkgName="$1"
11 working="$2"
12 outdir=$(readlink -f $3)
13
14 cd "$working"
15 git rev-parse HEAD
16
17 # find directory containing package.json file with the same name as the package
18 # we sort the paths so that shallower ones are preferred over deeper ones
19 pkgDir=$(find . -name package.json | while read pkg; do test $(jq -r .name $pkg) = "$pkgName" && echo $pkg; done | xargs dirname | sort -n)
20
21 # first install dependencies in the root
22 npm install --production=false || echo "Dependency installation failed; continuing."
23
24 # then install dependencies in the package directory (if different from root)
25 if ! [ -z "$pkgDir" ] && [ "$pkgDir" != "." ]; then
26   cd "$pkgDir"
27   npm install --production=false || echo "Dependency installation failed; continuing."
28 fi
29
30 # attempt to set the time to the time of publication
31 now=$(date)
32 (npm view "$packageName" time --json | jq ".[\"$version\"]" | xargs sudo -n date -s) || \
33   (echo "Failed to set date; continuing.")
34
35 # run a few possible build scripts under a 10-minute timeout
```

```
$ build-package.sh
17  # find directory containing package.json file with the same name as the package
18  # we sort the paths so that shallower ones are preferred over deeper ones
19  pkgDir=$(find . -name package.json | while read pkg; do test $(jq -r .name $pkg) = "$pkgName" && echo $pkg; done | xargs dirname | sort -n)
20
21  # first install dependencies in the root
22  npm install --production=false || echo "Dependency installation failed; continuing."
23
24  # then install dependencies in the package directory (if different from root)
25  if ! [ -z "$pkgDir" ] && [ "$pkgDir" != "." ]; then
26    cd "$pkgDir"
27    npm install --production=false || echo "Dependency installation failed; continuing."
28  fi
29
30  # attempt to set the time to the time of publication
31  now=$(date)
32  (npm view "$packageName" time --json | jq ".[\"$version\"]" | xargs sudo -n date -s) || \
33  (echo "Failed to set date; continuing.")
34
35  # run a few possible build scripts under a 10-minute timeout
36  for target in compile build pack webpack; do
37    for prefix in "" "our:"; do
38      timeout 10m npm run "$prefix$target" || true
39    done
40  done
41  tarball=$(npm pack | tail -n 1)
42
43  # reset time
44  sudo -n date -s "$now" || echo "Failed to reset time; continuing."
45
46  mkdir -p "$outdir"
47  tar xf "$tarball" -C "$outdir" --strip-components=1
48
```

A.2.3. Hash package

- Tính toán hàm băm MD5 của nội dung của gói để dùng so sánh với danh sách hàm băm của các gói độc hại đã biết.
- Khi tính toán hàm băm, chúng tôi bỏ qua tên gói và phiên bản trong *package.json* vì chúng luôn là duy nhất và có thể gây ra sai sót giả.
- Không có nỗ lực nào khác trong việc so khớp mờ được thực hiện nên chỉ có thể phát hiện được các bản sao nguyên văn.

Đồ án môn học – NT521 - Báo cáo tổng kết

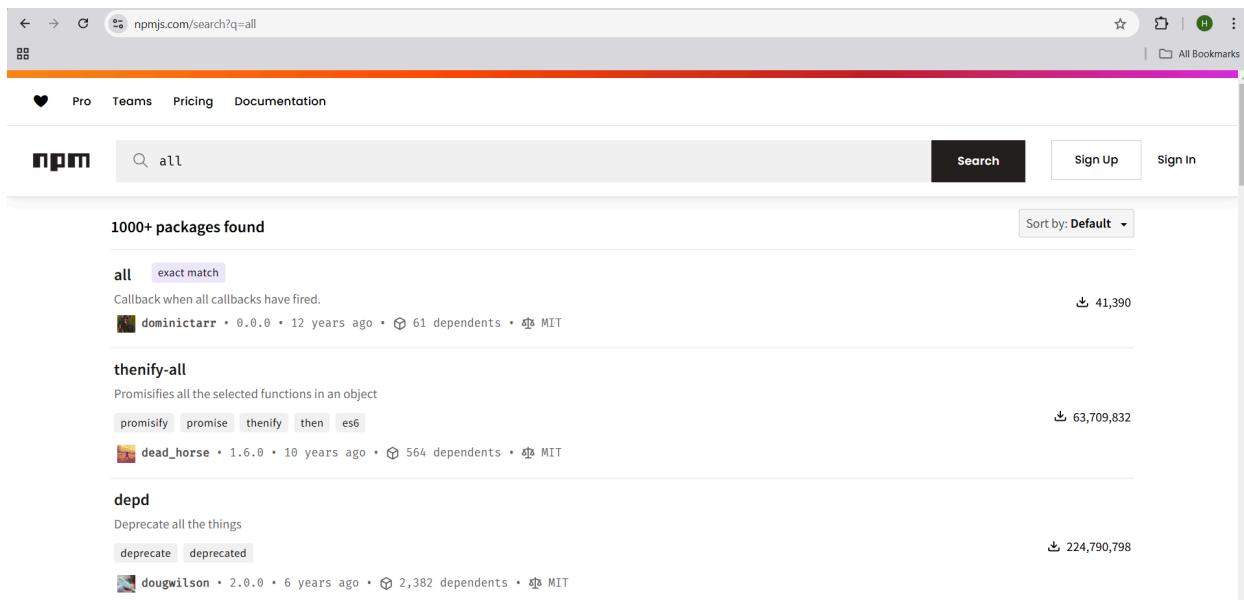
```
hash_package.py x
amalfi-artifact > code > clone-detector > hash_package.py > hash_package
8
9 def hash_package(root):
10     """
11     Compute an md5 hash of all files under root, visiting them in deterministic order.
12     'package.json' files are stripped of their 'name' and 'version' fields.
13     """
14     m = hashlib.md5()
15     for dirname, dirnames, filenames in os.walk(root):
16         dirnames.sort()
17         for filename in sorted(filenames):
18             path = os.path.join(dirname, filename)
19             m.update(f'{os.path.relpath(path, root)}\n'.encode("utf-8"))
20             if filename == "package.json":
21                 pkg = json.load(open(path))
22                 pkg["name"] = ""
23                 pkg["version"] = ""
24                 m.update(json.dumps(pkg, sort_keys=True).encode("utf-8"))
25             else:
26                 with open(path, "rb") as f:
27                     m.update(f.read())
28     return m.hexdigest()
29
30
31 if __name__ == "__main__":
32     if len(sys.argv) < 2:
33         print(f"Usage: {sys.argv[0]} <package directory>, file=sys.stderr)
34         print(f" Prints an md5 hash of all files in the given package directory, ignoring package name and version.", file=sys.stderr)
35         sys.exit(1)
36     print(hash_package(sys.argv[1]))
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(vENV) [venv] (kali㉿kali)-[~/LTAT_KTLH]
● $ python3 amalfi-artifact/code/clone-detector/hash_package.py malicious/npm_pack/evocateur-libnpmpublish/99.3.5/
aea10cbc50a1c72763077c48ca6a8dce
```

A.3. Thành phần mà nhóm đã thực hiện

A.3.1. Các gói npm

• Các gói npm lành tính:

- Tìm thông tin về các gói lành tính trên [npm | Home](#) bằng cách search với nội dung là **all**. Ta lấy tên các gói và lưu vào *benign-packages.txt*



- **create_dataset.py** đọc danh sách các gói npm từ file *benign_packages.txt*, sau đó kiểm tra và tái về tất cả các phiên bản của từng gói đó nếu chưa tồn tại trong thư mục

benign_dataset, giải nén nội dung của các file .tgz sau khi tải về, và lưu trữ chúng trong cấu trúc thư mục tương ứng.

```

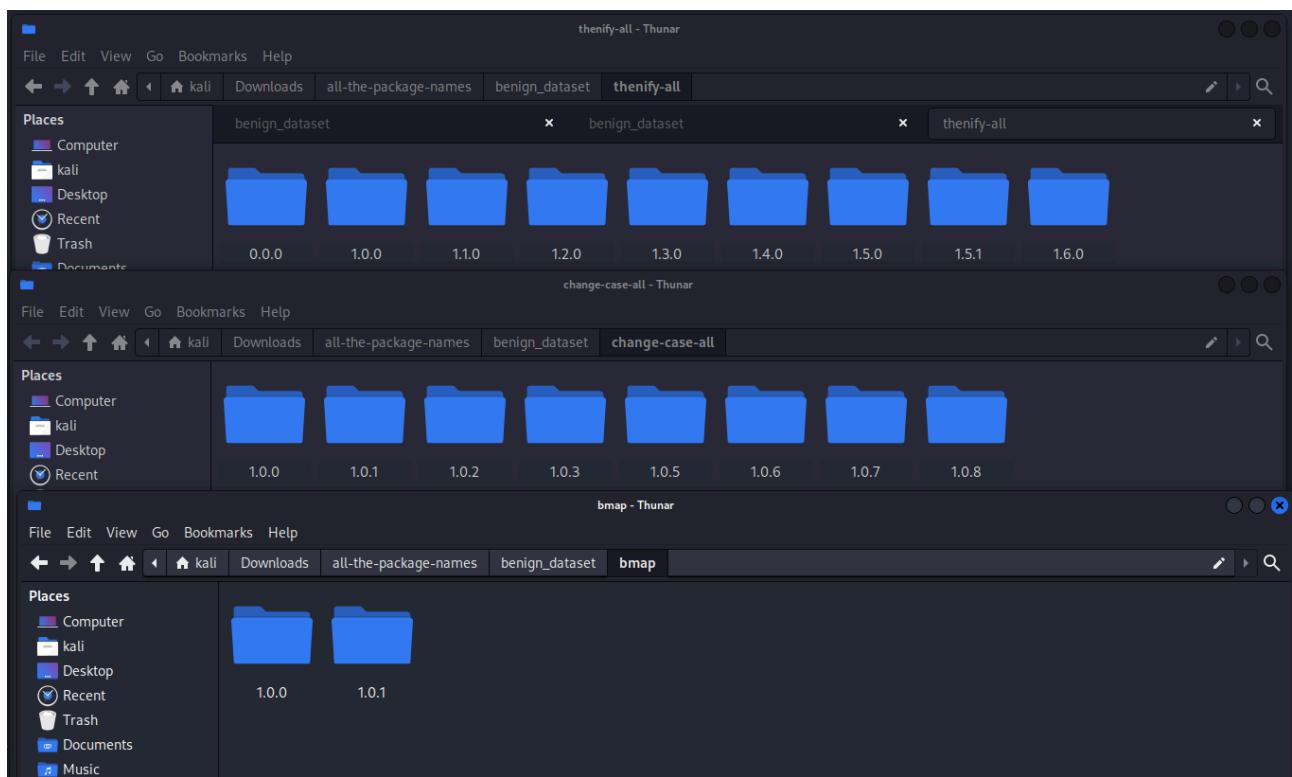
packages_file = "benign_packages.txt"
base_dir = "benign_dataset"
os.makedirs(base_dir, exist_ok=True)
# Hàm kiểm tra gói đã tồn tại trong thư mục chưa
def package_exists(package_name, version):
    package_dir = os.path.join(base_dir, package_name, version)
    return os.path.exists(package_dir)
# Hàm tải và lưu package
def download_package(package_name):
    try:
        # Lấy thông tin package từ npm
        print(f"Dang tai thong tin cho goi: {package_name}")
        result = subprocess.run(
            ["npm", "view", package_name, "versions", "--json"],
            capture_output=True, text=True, check=True
        )
        # Kiểm tra nếu có lỗi xảy ra khi lấy thông tin
        if result.returncode != 0:
            print(f'Lỗi khi lấy thông tin package {package_name}: {result.stderr}')
            return
        versions = json.loads(result.stdout)
        if not versions:
            print(f'Không tìm thấy phiên bản nào cho gói: {package_name}')
            return
        # Tải tất cả các phiên bản của package
        for version in versions:
            if package_exists(package_name, version):
                print(f'Gói {package_name}@{version} đã tồn tại, bỏ qua tải xuống.')
                continue # Bỏ qua nếu gói đã tồn tại
            package_dir = os.path.join(base_dir, package_name, version)
            os.makedirs(package_dir, exist_ok=True)
            print(f'Dang tai: ({package_name}@{version})')
            subprocess.run(
                ["npm", "pack", f'{package_name}@{version}'],
                cwd=package_dir, check=True
            )
            # Giải nén nội dung .tgz vào thư mục
            tgz_file = next((f for f in os.listdir(package_dir) if f.endswith(".tgz")), None)
            if tgz_file:
                tgz_path = os.path.join(package_dir, tgz_file)
                subprocess.run(["tar", "-xzf", tgz_path, "--strip-components=1", "-C", package_dir], check=True)
                os.remove(tgz_path) # Xóa file .tgz sau khi giải nén
                print(f'Dã giải nén và xóa file .tgz: {tgz_file}')
    except subprocess.CalledProcessError as e:
        print(f'Lỗi khi xử lý package {package_name}: {e}')

```

The screenshot shows two terminal windows on a Kali Linux system. The left window displays the command `python3 create_dataset.py` being run, followed by the output of the script which includes npm notices for the 'bmap' and 'thenify-all' packages. The right window shows the resulting directory structure under 'benign_dataset'. It contains two main folders: 'Benign' and 'Malicious'. The 'Benign' folder contains a single file 'bmap@1.0.0.tgz'. The 'Malicious' folder contains several files including 'History.md', 'README.md', and 'build/index.js'. The terminal output indicates that the script has successfully created the dataset and removed the temporary .tgz files.

- Kho dữ liệu các gói npm lành tính:

Đồ án môn học – NT521 - Báo cáo tổng kết



- **Các gói npm độc hại:**

- Tải từ <https://github.com/DataDog/malicious-software-packages-dataset>
- Giải nén [các gói npm độc hại](#) bằng `extract.sh`

```
$ extract.sh x
$ extract.sh
1 #!/bin/bash
2
3 if [ $# -ne 2 ]; then
4     echo "Usage: $0 source-directory destination-directory"
5     exit 1
6 fi
7
8 source_dir=$1
9 destination_dir=$2
10
11 # Tạo thư mục đích nếu chưa tồn tại
12 mkdir -p "$destination_dir"
13
14 # Tìm tất cả các file .zip trong thư mục nguồn
15 find "$source_dir" -type f -name '*.zip' | while read sample; do
16     # Tạo đường dẫn đích tương ứng trong cùng thư mục
17     relative_path=$(realpath --relative-to="$source_dir" "$(dirname "$sample")")
18     target_dir="$destination_dir/$relative_path"
19
20     # Tạo thư mục đích tương ứng
21     mkdir -p "$target_dir"
22
23     # Giải nén file ZIP vào thư mục đích
24     unzip -o -P infected "$sample" -d "$target_dir" >/dev/null 2>&1
25     if [ $? -eq 0 ]; then
26         echo "Đã giải nén: $sample vào $target_dir"
27     else
28         echo "Không thể giải nén: $sample"
29     fi
30 done
31
32 echo "Hoàn tất giải nén tất cả các file ZIP vào $destination_dir."
```

- Định lại cấu trúc thư mục bằng **restruct.py**

Đồ án môn học – NT521 - Báo cáo tổng kết

```
restruct.py > ...
1  import os
2  import shutil
3  import argparse
4
5  # Hàm dê'di chuyên các tệp cùng cấp với file JSON ra thư mục đích
6  def move_files_to_version_dir(version_dir):
7      for root, dirs, files in os.walk(version_dir, topdown=False):
8          for file in files:
9              if file.endswith(".json"):
10                  json_dir = root # Thư mục chứa file JSON
11
12                  # Di chuyên tất cả file và thư mục cùng cấp với file JSON ra thư mục version
13                  for item in os.listdir(json_dir):
14                      item_path = os.path.join(json_dir, item)
15                      target_path = os.path.join(version_dir, item)
16
17                      if os.path.exists(target_path):
18                          # Nếu file hoặc thư mục đã tồn tại, bỏ qua hoặc hợp nhất (tùy chỉnh nếu cần)
19                          print(f"Bỏ qua: {target_path} đã tồn tại.")
20                          continue
21
22                      shutil.move(item_path, target_path)
23
24                  break # Không cần tìm thêm file JSON khác trong cùng thư mục
25
26                  # Xóa thư mục trống
27                  if not os.listdir(root):
28                      os.rmdir(root)
29
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Di chuyên các tệp cùng cấp với file JSON ra thư mục đích.")
    parser.add_argument("base_dir", type=str, help="Đường dẫn thư mục gốc chứa các package/version")

    args = parser.parse_args()
    base_dir = args.base_dir

    # Kiểm tra sự tồn tại của thư mục gốc
    if not os.path.isdir(base_dir):
        print(f"Thư mục '{base_dir}' không tồn tại.")
        exit(1)

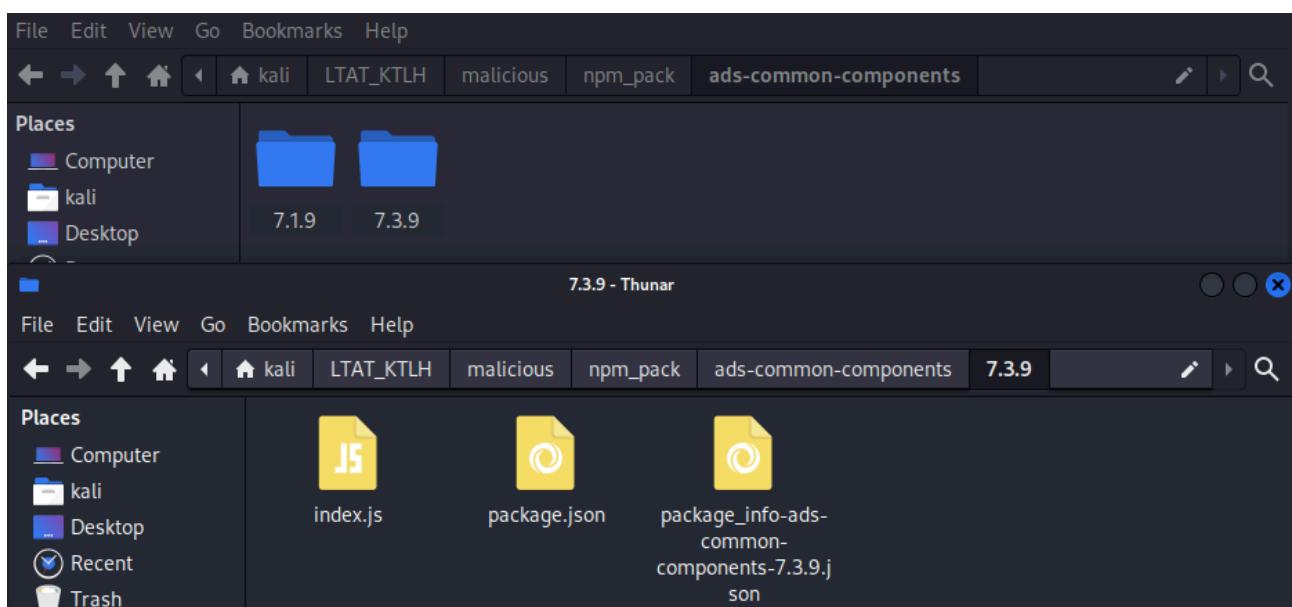
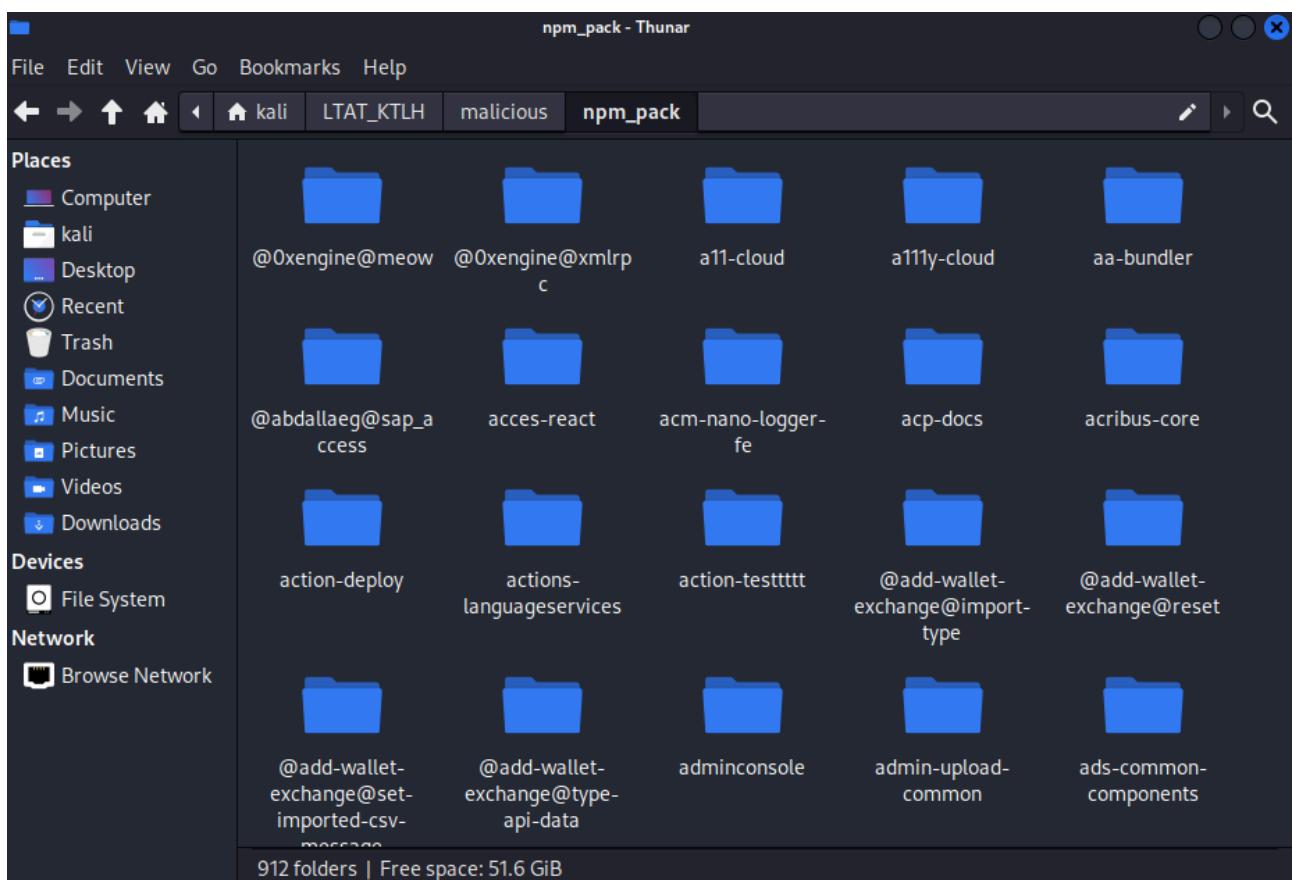
    # Lặp qua từng thư mục package/version
    for package in os.listdir(base_dir):
        package_path = os.path.join(base_dir, package)

        if os.path.isdir(package_path):
            for version in os.listdir(package_path):
                version_path = os.path.join(package_path, version)

                if os.path.isdir(version_path):
                    print(f"Xử lý: {version_path}")
                    move_files_to_version_dir(version_path)

    print("Hoàn tất!")
```

- Kho dữ liệu các gói npm độc hại:



A.3.2. Feature Extractor

- Để có thể tạo dữ liệu đầu vào cho chương trình train_classifier, chúng em đã viết 1 chương trình trích xuất đặc tính của gói npm. Với các đặc tính trích xuất có 1 trong các đặc điểm sau:
 - Truy cập vào các thông tin cá nhân (PII)
 - Truy cập vào tài nguyên hệ thống

- Sử dụng các API cụ thể
- Sử dụng tập lệnh cài đặt gói
- Sử dụng mã được rút gọn (để tránh bị phát hiện) hoặc tệp nhị phân (chẳng hạn như các tệp thực thi nhị phân)
- Chương trình sử dụng tree sitter để phân tích code của gói npm và tìm xem có các câu lệnh truy cập và đọc file như readFile, writeFile hay không, hoặc mở các lệnh shell như exec, hoặc kết nối đến internet như http.request, hoặc các câu lệnh mã hóa giải mã.

```
feature_extractor.py
1 def extract_sensitive_code_features(js_code):
2     """Extract features using Tree-sitter for sensitive JavaScript patterns."""
3     parser = Parser(JS_LANGUAGE)
4     tree = parser.parse(bytes(js_code, "utf8"))
5     root_node = tree.root_node
6
7     features = {
8         "system_command_usage": 0,
9         "file_access": 0,
10        "env_variable_access": 0,
11        "network_access": 0,
12        "crypto_usage": 0,
13        "data_encoding": 0,
14        "dynamic_code_generation": 0,
15    }
16
17    def traverse(node):
18        if node.type == "call_expression":
19            func_node = node.child_by_field_name("function")
20            if func_node:
21                func_name = func_node.text.decode("utf8")
22                if func_name in {"fs.readFile", "fs.writeFile", "fs.unlink"}:
23                    features["file_access"] += 1
24                elif func_name == "process.env":
25                    features["env_variable_access"] += 1
26                elif func_name in {"exec", "spawn"}:
27                    features["system_command_usage"] += 1
28                elif func_name in {"http.request", "https.request", "fetch"}:
29                    features["network_access"] += 1
30                elif func_name in {"crypto.createCipher", "crypto.createHash"}:
31                    features["crypto_usage"] += 1
32                elif func_name in {"eval", "Function", "setTimeout", "setInterval"}:
33                    features["dynamic_code_generation"] += 1
34                elif func_name in {"encodeURIComponent", "decodeURIComponent", "btoa", "atob"}:
35                    features["data_encoding"] += 1
36
37            for child in node.children:
38                traverse(child)
39
40    traverse(root_node)
```

- Chúng em cũng thực tính các đặc tính như entropy, số file binary hay minified của gói, số dependencies, lệnh scripts mà gói sử dụng, các từ keyword đáng nghi.

Đồ án môn học – NT521 - Báo cáo tổng kết

```
1  #!/usr/bin/python3
2
3  def extract_package_features(package_dir):
4      """Extract features from an npm package directory."""
5      features = {
6          "max_entropy": 0,
7          "avg_entropy": 0,
8          "minified_files": 0,
9          "binary_files": 0,
10         "pii_keywords": 0,
11         "dependencies_count": 0,
12         "scripts_count": 0,
13         "has_postinstall_script": 0,
14     }
15
16     pii_keywords = ["password", "creditcard", "cookie"]
17
18     file_sizes = []
19     entropies = []
20
21     for root, _, files in os.walk(package_dir):
22         for file in files:
23             filepath = os.path.join(root, file)
24             file_sizes.append(os.path.getsize(filepath))
25
26             if detect_binary(filepath):
27                 features["binary_files"] += 1
28                 continue
29
30             content = read_file(filepath)
31             if content:
32                 entropy = calculate_entropy(content)
33                 entropies.append(entropy)
34
35                 # Detect PII
36                 features["pii_keywords"] += sum(keyword in content for keyword in pii_keywords)
37
38                 # Minified file detection
39                 if len(content.splitlines()) < 5 and len(content) > 500:
40                     features["minified_files"] += 1
41
42             if file.endswith(".js"):
43                 js_features = extract_sensitive_code_features(content)
44                 for key, value in js_features.items():
45                     features[key] = features.get(key, 0) + value
46
47     package_json_path = os.path.join(package_dir, "package.json")
48     if os.path.isfile(package_json_path):
49         try:
50             with open(package_json_path, 'r', encoding='utf-8') as f:
51                 package_data = json.load(f)
52
53                 features["dependencies_count"] = extract_dependencies_count(package_data)
54                 scripts = package_data.get("scripts", {})
55                 features["scripts_count"] = len(scripts)
56                 features["has_postinstall_script"] = 1 if "postinstall" in scripts else 0
57
58             except Exception as e:
59                 print(f"Error reading package.json: {e}")
60
61             features["max_entropy"] = max(entropies) if entropies else 0
62             features["avg_entropy"] = sum(entropies) / len(entropies) if entropies else 0
63
64     return features
65
```

- Sau đó sử dụng 1 chương trình organize.py để thực hiện trích xuất các gói npm của kho dữ liệu cùng 1 lúc

```

6  def organize_and_extract_features_with_tree_sitter(npm_packages_dir, output_dir):
7      npm_packages_dir = Path(npm_packages_dir)
8      output_dir = Path(output_dir)
9      output_dir.mkdir(parents=True, exist_ok=True)
10
11     for package_name_dir in npm_packages_dir.iterdir():
12         if package_name_dir.is_dir(): # 'package_name' level
13             package_name = package_name_dir.name
14             for version_dir in package_name_dir.iterdir():
15                 if version_dir.is_dir(): # 'version' level
16                     version = version_dir.name
17                     try:
18                         # Đường dẫn output cho package_name/version
19                         package_output_dir = output_dir / package_name / version
20                         package_output_dir.mkdir(parents=True, exist_ok=True)
21
22                         # Trích xuất đặc tính và lưu thành change-features.csv
23                         features = extract_package_features(version_dir)
24                         change_features_path = package_output_dir / "change-features.csv"
25                         with open(change_features_path, "w", newline="") as f:
26                             writer = csv.writer(f)
27                             writer.writerow(["feature", "value"]) # Ghi tiêu đề cột
28                             for feature, value in features.items():
29                                 writer.writerow([feature, value]) # Ghi từng cặp feature-value
30
31                         print(f"Processed {package_name}/{version} -> {change_features_path}")
32
33                     except Exception as e:
34                         print(f"Error processing {package_name}/{version}: {e}")
35
36     print(f"All packages organized and features extracted to {output_dir}")
37
38 if __name__ == "__main__":
39     import argparse
40
41     parser = argparse.ArgumentParser(description="Organize NPM packages and extract features using Tree-sitter.")
42     parser.add_argument("npm_packages_dir", help="Directory containing unzipped NPM packages.")
43     parser.add_argument("output_dir", help="Directory to organize packages and save features.")
44     args = parser.parse_args()
45

```

- Kết quả chạy:

```

$ python3 organize.py basic_corpus training Sets
Processed evocateur-libmpmpublish/99.3.5 -> Training_sets/evocateur-libmpmpublish/99.3.5/change-features.csv
Processed xstate-viz/8.5.1 -> Training_sets/xstate-viz/8.5.1/change-features.csv
Processed signed-off/0.9.0 -> Training_sets/signed-off/0.9.0/change-features.csv
Processed ng-ul-library/1.0.12 -> Training_sets/ng-ul-library/1.0.12/change-features.csv
Processed ng-ul-library/1.0.970 -> Training_sets/ng-ul-library/1.0.970/change-features.csv
Processed ng-ul-library/1.0.984 -> Training_sets/ng-ul-library/1.0.984/change-features.csv
Processed ng-ul-library/1.0.956 -> Training_sets/ng-ul-library/1.0.956/change-features.csv
Processed ng-ul-library/1.0.993 -> Training_sets/ng-ul-library/1.0.993/change-features.csv
Processed ng-ul-library/1.1.33 -> Training_sets/ng-ul-library/1.1.33/change-features.csv
Processed ng-ul-library/1.0.933 -> Training_sets/ng-ul-library/1.0.933/change-features.csv
Processed ng-ul-library/1.1.31 -> Training_sets/ng-ul-library/1.1.31/change-features.csv
Processed ng-ul-library/1.0.941 -> Training_sets/ng-ul-library/1.0.941/change-features.csv
Processed ng-ul-library/1.0.905 -> Training_sets/ng-ul-library/1.0.905/change-features.csv
Processed ng-ul-library/1.0.958 -> Training_sets/ng-ul-library/1.0.958/change-features.csv
Processed ng-ul-library/1.0.965 -> Training_sets/ng-ul-library/1.0.965/change-features.csv
Processed ng-ul-library/1.0.948 -> Training_sets/ng-ul-library/1.0.948/change-features.csv
Processed ng-ul-library/1.1.10 -> Training_sets/ng-ul-library/1.1.10/change-features.csv

```

A.3.3. Predictor

- Sau khi đã tạo được các mô hình phân loại gói npm từ chương trình train_classifier. Chúng em tạo 1 chương trình để sử dụng các mô hình đó để thực hiện dự đoán danh sách các gói npm

Đồ án môn học – NT521 - Báo cáo tổng kết

```
prediction.py
4
5 def classify_packages_from_csv(npm_packages_dir, classifier_file, output_csv_file):
6     # Tải mô hình đã được lưu
7     with open(classifier_file, "rb") as f:
8         model_data = pickle.load(f)
9
10    feature_names = model_data["feature_names"]
11    booleanize = model_data["booleanize"]
12    classifier = model_data["classifier"]
13
14    # Mở tệp CSV để ghi kết quả
15    with open(output_csv_file, "w", newline="", encoding="utf-8") as csv_file:
16        csv_writer = csv.writer(csv_file)
17        # Ghi tiêu đề cột
18        csv_writer.writerow(["package", "version", "label"])
19
20        # Duyệt qua tất cả các gói trong thư mục
21        for package_dir in os.listdir(npm_packages_dir):
22            package_path = os.path.join(npm_packages_dir, package_dir)
23            if os.path.isdir(package_path):
24                for version_dir in os.listdir(package_path):
25                    version_path = os.path.join(package_path, version_dir)
26                    if os.path.isdir(version_path):
27                        change_features_path = os.path.join(version_path, "change-features.csv")
28                        if os.path.exists(change_features_path):
29                            try:
30                                # Đọc các đặc tính từ file change-features.csv
31                                raw_features = {}
32                                with open(change_features_path, "r", encoding="utf-8") as f:
33                                    reader = csv.reader(f)
34                                    next(reader)
35                                    for row in reader:
36                                        feature, value = row
37                                        raw_features[feature] = float(value) # Hoặc float(value) nếu giá trị không phải số nguyên
38
39                                # Đảm bảo tính nhất quán với các đặc tính đã sử dụng khi huấn luyện
40                                feature_vector = [raw_features.get(feature, 0) for feature in feature_names]
41
42                                if booleanize:
43                                    feature_vector = [1 if x else 0 for x in feature_vector]
44
45                                raw_features = {}
46                                with open(change_features_path, "r", encoding="utf-8") as f:
47                                    reader = csv.reader(f)
48                                    next(reader)
49                                    for row in reader:
50                                        feature, value = row
51                                        raw_features[feature] = float(value) # Hoặc float(value) nếu giá trị không phải số nguyên
52
53                                # Đảm bảo tính nhất quán với các đặc tính đã sử dụng khi huấn luyện
54                                feature_vector = [raw_features.get(feature, 0) for feature in feature_names]
55
56                                if booleanize:
57                                    feature_vector = [1 if x else 0 for x in feature_vector]
58
59                                # Dự đoán nhãn (0 = lành tính, 1 = độc hại)
60                                prediction = classifier.predict([feature_vector])[0]
61                                label = "Malicious" if prediction == 1 else "Benign"
62
63                                # Ghi kết quả vào CSV
64                                csv_writer.writerow([package_dir, version_dir, label])
65
66                                print(f"Processed {package_dir}@{version_dir}: {label}")
67
68                                except Exception as e:
69                                    print(f"Error processing {package_dir}@{version_dir}: {e}")
70
71    print("Classification completed. Results saved to {output_csv_file}")

72
73 if __name__ == "__main__":
74     import argparse
75
76     parser = argparse.ArgumentParser(description="Classify NPM packages using pre-extracted features.")
77     parser.add_argument("--npm_packages_dir", help="Directory containing NPM packages with change-features.csv files.")
78     parser.add_argument("--classifier_file", help="Trained classifier file (e.g., classifier.pkl).")
79     parser.add_argument("--output_csv_file", help="File to save classification results in CSV format.")
80     args = parser.parse_args()
81
82     classify_packages_from_csv(args.npm_packages_dir, args.classifier_file, args.output_csv_file)
```

- Kết quả chạy:

```
output_dt.csv
135  @prisma@fetch-engine,6.0.1,Benign
136  pyodide,0.26.4,Benign
137  @pnpm@package-bins,1000.0.0,Benign
138  @smithy@core,2.5.5,Benign
139  @lwc@template-compiler,8.12.0,Benign
140  @smithy@middleware-serde,3.0.11,Benign
141  @monorepo-utils@package-utils,2.10.4,Malicious
142  radicjs,0,Benign
143  radicjs,2,Benign
144  @react-spring@types,9.7.5,Benign
145  @semantic-release@npm,12.0.1,Malicious
146  spinal-service-ticket,9.2.9,Malicious
147  elkeid,9.3.5,Benign
148  cdntodister,20.0.0,Benign
149  airwallex-platform-onboarding-sdk-demo-react,1.0.0,Malicious
150  com.microsoft.mrtk.graphics-tools.unity,0.4.0,Benign
151  com.microsoft.mrtk.graphics-tools.unity,4.0.0,Benign
152  @volcenjine@veoplayer,89.3.5,Malicious
153  configure-pages,4.3.0,Benign
154  trufel,5.11.5,Benign
155  @smithy@hash-blob-browser,3.1.10,Malicious
156  seatalk-openapi,6.5.8,Benign
157  sap-bucket,0.0.0,Benign
158  mona-plugin-events,99.11.18,Benign
159  spid-growth-notifier-config,9.5.2,Benign
160  custom-ui-extension-template,1.0.1,Benign
161  debugr1,3.1.1,Benign
162  com.unity.entities,9.4.5,Benign
163  com.unity.entities,7.1.2,Benign
164  bloxupgrade,1.0.0,Benign
165  vue2-tiktok,89.3.5,Malicious
166  react-native-markdown-package,1.8.2,Malicious
167  nk_element_1_0_0,Benign

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
Processed pyramid-proportion@1.0.1: Benign
Processed pyramid-proportion@1.0.2: Benign
Processed pyramid-proportion@1.0.4: Benign
Classification completed. Results saved to output_dt.csv

└─(testenv)─(kali㉿kali)-[~/Documents/training]
└─$ python3 prediction.py training_n1 trained_dt.pkl output_dt.csv
```

A.3.4. Clone detector

- Với mục tiêu tự động hóa quy trình phát hiện các gói độc hại, chúng tôi nâng cấp từ việc tính toán hash của từng gói (hash package) và so sánh thủ công, thành một hệ thống phát hiện sao chép (clone detector).
- Cơ chế hoạt động:
 - + Đọc tệp csv chứa thông tin các gói độc hại và lành tính đã biết (basic corpus), lọc ra hash của các gói được đánh dấu là malicious.

- + Duyệt qua các gói và phiên bản trong thư mục new_packages_dir và tính toán hash của chúng.
- + Kiểm tra từng hash của các gói mới xem có nằm trong danh sách hash độc hại không. Nếu có, clone_detected = “yes”.
- + Ghi thông tin của tất cả gói (tên, phiên bản, hash, kết quả phát hiện clone) vào tệp csv output.

```
def hash_versions(directory):
    """
    Traverse all packages and versions in the given directory, and compute hash for each version.
    Returns a dictionary: {hash: (package_name, version)}.
    """
    hashes = {}
    for package in os.listdir(directory):
        package_path = os.path.join(directory, package)
        if os.path.isdir(package_path):
            for version in os.listdir(package_path):
                version_path = os.path.join(package_path, version)
                if os.path.isdir(version_path):
                    package_hash = hash_package(version_path)
                    hashes[package_hash] = (package, version)
    return hashes
```

```
def detect_clones(basic_corpus_csv, new_packages_dir, output_csv="results.csv"):
    # Đọc basic_corpus.csv và lấy các hash malicious
    print("Reading basic corpus...")
    basic_corpus = pd.read_csv(basic_corpus_csv)
    malicious_hashes = basic_corpus[basic_corpus['analysis'] == 'malicious']['hash'].tolist()

    print(f"Found {len(malicious_hashes)} malicious hashes in the basic corpus.")

    # Hash tất cả các gói trong new packages
    print("Hashing new packages...")
    new_hashes = hash_versions(new_packages_dir)

    print("\nDetecting clones...")
    results = []

    for new_hash, (package, version) in new_hashes.items():
        if new_hash in malicious_hashes:
            print(f"Clone detected: {package}@{version} matches a malicious package in the basic corpus.")
            clone_detected = "yes"
            results.append({
                "package": package,
                "version": version,
                "hash": new_hash,
                "clone_detect": clone_detected
            })

    # Write results to CSV
    with open(output_csv, mode="w", newline="") as csv_file:
        fieldnames = ["package", "version", "hash", "clone_detect"]
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
        for row in results:
            writer.writerow(row)

    print(f"\nResults have been written to {output_csv}")
```

- Kết quả chạy:

Đồ án môn học – NT521 - Báo cáo tổng kết

```
1 package,version,hash,clone_detect
2 airpay-js-sdk,7.1.9,d8d709d7608d97b587976732c7a2112a,yes
3 adminconsole,1.0.0,b8a57af21d7e349e30cc37ea88710ff2,yes
4 @alaska-its@design-tokens,1.0.1,e682d2b82a3b3b636628801efb5a8ff8,yes
5 aem-core-react-components,6.9.9,9d68aa33d198a9fc56aa5ed3fd1c0b5b,yes
6 aem-core-react-components,6.8.9,b0ad3380e2ec1997b843a6112b1033ef,yes
7 @oxengine@xmlrpc,1.3.18,001f76da5b0c9aab6fec868bcc8103a,yes
8 aa-bundler,0.0.2,2fb74214bac1730db0bec219ef778ee2,yes
9 anumanu2324,1.0.1,2c9046bd7f462dfaf97f081e827bbaa9,yes
10 apc-admin-utils,6.1.3,566fd8399572d2051a9c1e25632944af,yes
11 @afound@common,10.10.12,83e274ed0c8b0c7444c9b20b8997d7f2,yes
12 apa-components,6.1.3,0eeddeb8e66e1036f3e31d5ac021a08e,yes
13 @al-ui@useappinsights,1.0.1,9921bd715fc77b3ea4790ea4854762,yes
14 elkeid,9.3.5,15a8121e0a4d9e2258c4abe304cffald,yes
15 cdntodister,20.0.0,f22f1d422531c01b3e88589c205b19ab,yes
16 airwallex-platform-onboarding-sdk-demo-react,1.0.0,fc187300157b260f4db04c3229ce17d3,yes
17 com.microsoft.mrtk.graphics.unity,0.4.0,199a65993c36a8b08ca0f6eca067162a,yes
18 com.microsoft.mrtk.graphics.unity,4.0.0,0a4e55ffd0b6145464670c5f2aaa8e9e,yes
19 configure-pages,4.3.0,d56fe84743af886455c768fa87d99784,yes
20 mona-plugin-events,99.11.18,4390efd11f39b5c21133ed3dbe2dabbe,yes
21 custom-ui-extension-template,1.0.1,d7c450ada463984fe7d7038518a13adc,yes
22 debugr1,3.1.1,0c50970b60f091635f89076bb6293faf,yes
23 com.unity.entities,9.4.5,4e22d520cafca73113b2648cd90d39c6,yes
24 com.unity.entities,7.1.2,537f48c4a778c13c58bff728fdc2b834,yes
25 bloxupgrade,1.0.0,44295da4604f0889b184c3484d09b8dd,yes
26 cdp-wallet-manager,1.0.0,be8513b2ff9d908d2e48045f5a58764b,yes
27 cdp-wallet-manager,0.1.0,7fc2b88ffffdd33d5a23d3e06379105c,yes
28 appetize-cli,1.2.0,9fbb60610b249bb6d709f7eeb963b287,yes
29 appetize-cli,1.3.0,a589ce78f6a6fe9764a00fec3b44d0a1,yes
30 peritter,1.0.0,14858cbae3206bdda353e241c2aeb9fa,yes
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
└─(testenv)─(kali㉿kali)─[~/Documents/training]
$ python3 clone_detect.py label.csv N1 clone-detect.csv
Reading basic corpus...
Found 662 malicious hashes in the basic corpus.
Hashing new packages...

Detecting clones...
Clone detected: airpay-js-sdk@7.1.9 matches a malicious package in the basic corpus.
```

A.3.5. Reproducer

- Sau khi tổng hợp các gói npm được dự đoán là độc hại ở cả mục predictor và clone detector thành 1 danh sách, chúng em sẽ thực hiện tái tạo gói npm để tìm ra gói npm nào tái tạo được, gói nào không, nếu gói nào không tái tạo được sẽ ghi lại vào 1 file csv, file này sẽ là kết quả dự đoán cuối cùng.

Đồ án môn học – NT521 - Báo cáo tổng kết

```
def run_reproduce(package, version, output_dir):
    working_dir = "working"
    if os.path.exists(working_dir):
        print(f"Cleaning up: Removing folder '{working_dir}'...")
        shutil.rmtree(working_dir, ignore_errors=True)
    try:
        command = ["bash", "reproduce-package.sh", f"{package}@{version}", output_dir]
        result = subprocess.run(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, check=True, text=True)
        print(f"Successfully reproduced {package}@{version}: {result.stdout.strip()}")
        return True
    except subprocess.CalledProcessError as e:
        # Ghi lỗi vào log
        return False

def normalize_package_name(package_name):
    if package_name.count('@') > 1:
        parts = package_name.split('@')
        return f'{parts[0]}@{parts[1]}/{parts[2]}'
    return package_name

def process_packages(input_csv, output_csv, output_dir):
    failed_packages = []

    # Đọc file CSV đầu vào
    with open(input_csv, "r") as infile:
        reader = csv.DictReader(infile)

        # Duyệt qua từng hàng trong file CSV
        for row in reader:
            original_package = row["package"]
            package = normalize_package_name(original_package)
            version = row["version"]

            print(f"Processing {package}@{version}...")

            # Chạy tập lệnh reproduce-package.sh
            success = run_reproduce(package, version, output_dir)

            if not success:
                print(f"Failed to reproduce {package}@{version}.")
                failed_packages.append({"package": package, "version": version})

    # Ghi các gói thất bại vào file CSV đầu ra
    with open(output_csv, "w", newline="") as outfile:
        writer = csv.DictWriter(outfile, fieldnames=["package", "version"])
        writer.writeheader()
        writer.writerows(failed_packages)

    print(f"Done! Failed reproductions are saved to {output_csv}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Reproduce npm packages and log failures.")
    parser.add_argument("input_csv", help="Input CSV file containing 'package' and 'version' columns.")
    parser.add_argument("output_dir", help="Directory to save reproduction results.")
    parser.add_argument("output_csv", help="Output CSV file to log failed reproductions.")

Ln 15, Col 1  Spaces: 4  UTF-8  LF  Python  Q
```

- Kết quả:

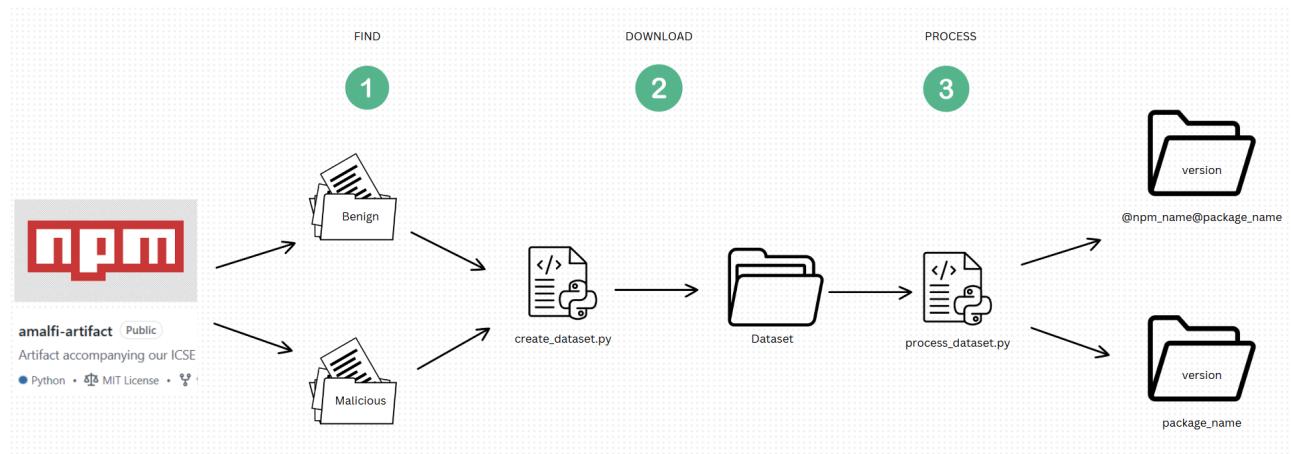
```
■ output.csv
1 package,version
2 airwallex-platform-onboarding-sdk-demo-react,1.0.0
3 adminconsole,1.0.0
4 action-deploy,2.0.1
5 pm-controls,0.0.40
6 acribus-core,6.1.3
7 apa-components,6.1.3
8 pm-controls,0.0.38
9 @abdallaeg/sap_access,0.0.0
10 anumanu2324,1.0.1
11 apache-airflow-ui,99.9.9
12 acp-docs,6.1.3
13 ap-components-react,3.15.9
14 pm-controls,0.0.46
15 ap-components-react,3.9.9
16 @oxengine/meow,1.2.7
17 pm-controls,0.0.39
18 @add-wallet-exchange/type-api-data,1.0.0
19 ap-components-react,3.16.9
20 ads-common-utils,7.1.9
21 airpay-js-sdk,7.1.9
22 acces-react,99.3.5
23 @al-ui/useappinsights,1.0.1
24 acm-nano-logger-fe,1.0.1
25 pm-controls,0.0.48
26 apache-airflow-ui,9.9.9
27 @oxengine/xmlrpc,1.3.18
28 @add-wallet-exchange/reset,1.0.0
29 actions-languageservices,2.0.9
30 pm-controls,0.0.41
31 @aftersale/react-eva,1.1.1
32 apc-admin-utils,6.1.3
33 pm-controls,0.0.42

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

└── (testenv)─(kali㉿kali)-[~/Documents/training]
$ python3 reproduce.py output-total.csv output output.csv
Processing airwallex-platform-onboarding-sdk-demo-react@1.0.0...
Failed to reproduce airwallex-platform-onboarding-sdk-demo-react@1.0.0.
Processing adminconsole@1.0.0...
```

B. Chi tiết cài đặt, hiện thực

B.1. Chuẩn bị dữ liệu



Giai đoạn 1: Tổng hợp tập dữ liệu cơ bản gồm các gói npm lành tính và độc hại

- Xét tổng số lượng gói ta có 2840 gói, cụ thể 2270:570 (Benign:Malicious)

```

kali@kali: ~/Downloads/Kho_Basic_Corpus
$ find Benign -mindepth 2 -maxdepth 2 -type d | wc -l
2270

$ find Malicious -mindepth 2 -maxdepth 2 -type d | wc -l
570
  
```

- **create_csv.py** tổng hợp dữ liệu bằng cách phân loại và tính toán giá trị băm các gói npm từ 2 thư mục **Benign** và **Malicious**, sau đó gộp lại ở dạng file .csv, cụ thể như sau:
 - + **hash-package()**: tính toán giá trị băm cho từng gói npm. Đầu tiên, ta tạo một đối tượng md5 của thư viện hashlib. Duyệt qua các tệp trong thư mục root và các thư mục con của nó. Đối với mỗi tệp, thêm đường dẫn tương ứng và một ký tự xuống dòng vào giá trị hash. Nếu trong thư mục có file **package.json**, ta mở và đọc nội dung của file, sau đó sẽ bỏ trường **name** và **version** trước khi thêm vào giá trị hash; đối với các thư mục còn lại hàm đọc và thêm nội dung của nó vào giá trị hash mà không thực hiện các thao tác khác.

```
def hash_package(root):
    """
    Tính hash giống với logic trong đoạn code clone-detect.
    `package.json` sẽ bị loại bỏ trường `name` và `version`.
    """
    m = hashlib.md5()
    for dirname, dirnames, filenames in os.walk(root):
        dirnames.sort()
        for filename in sorted(filenames):
            path = os.path.join(dirname, filename)
            m.update(f"{os.path.relpath(path, root)}\n".encode("utf-8")) # Đường dẫn tương đối với xuống dòng
            if filename == "package.json":
                with open(path, "r", encoding="utf-8") as f:
                    pkg = json.load(f)
                    pkg["name"] = ""
                    pkg["version"] = ""
                    m.update(json.dumps(pkg, sort_keys=True).encode("utf-8"))
            else:
                with open(path, "rb") as f:
                    m.update(f.read()) # Hash nội dung file
    return m.hexdigest()
```

- + **process-package()**: sau khi đã có hàm tính toán hash cho từng gói, ta dùng hàm này cho duyệt qua từng gói trong các thư mục **Benign** hoặc **Malicious** để tính toán giá trị băm cho từng gói. Tạo một mảng data để lưu kết quả sau khi duyệt mỗi gói. Kiểm tra nếu gói là một thư mục, tiếp tục duyệt qua các phiên bản của gói đó. Tiếp tục kiểm tra nếu phiên bản là một thư mục, tính giá trị hash của phiên bản đó bằng cách gọi hàm **hash-package()** ở trên. Data cuối cùng sẽ bao gồm tên gói, phiên bản, giá trị băm và nhãn (lành tinh hay độc hại).

```
def process_packages(base_dir, analysis_label):
    """
    Duyệt qua các gói và tính hash cho từng version.
    """
    data = []
    for package in os.listdir(base_dir):
        package_path = os.path.join(base_dir, package)
        if os.path.isdir(package_path): # Kiểm tra package là thư mục
            for version in os.listdir(package_path):
                version_path = os.path.join(package_path, version)
                if os.path.isdir(version_path): # Kiểm tra version là thư mục
                    package_hash = hash_package(version_path)
                    data.append([package, version, package_hash, analysis_label])
    return data
```

- + **main()**: ở đây ta khai báo 2 thư mục cần tổng hợp dữ liệu là **Malicious** và **Benign** sau đó lưu kết quả vào file **basic_corpus.csv**. Ta sẽ thực hiện tính giá trị hash ở từng thư mục trước, sau khi có kết quả ở từng thư mục ta tạo header như các danh mục để lưu dữ liệu và gộp chúng lại thành một file duy nhất.

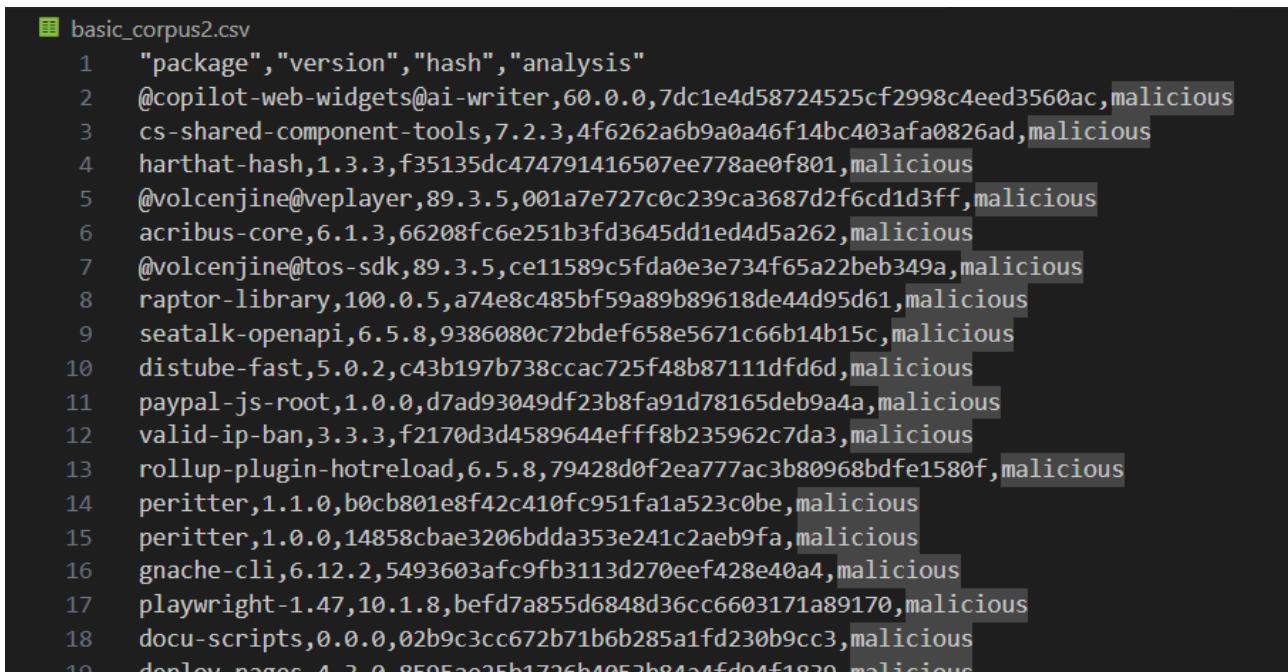
```
def main():
    # Đường dẫn đến thư mục malicious và benign
    malicious_dir = "Malicious"
    benign_dir = "Benign"
    output_file = "basic_corpus.csv"

    # Xử lý từng thư mục và tính hash
    malicious_data = process_packages(malicious_dir, "malicious")
    benign_data = process_packages(benign_dir, "benign")

    # Ghi dữ liệu vào file CSV
    with open(output_file, mode="w", newline="", encoding="utf-8") as file:
        writer = csv.writer(file)
        writer.writerow(["package", "version", "hash", "analysis"]) # Ghi header
        writer.writerows(malicious_data + benign_data) # Ghi dữ liệu

    print(f"File CSV đã được tạo: '{output_file}'")
```

- Kết quả:
 - + Basic corpus có 1789 gói (1219:570 - Benign:Malicious).
 - + Các gói còn lại dùng cho thí nghiệm.



```
basic_corpus2.csv
1 "package", "version", "hash", "analysis"
2 @copilot-web-widgets@ai-writer,60.0.0,7dc1e4d58724525cf2998c4eed3560ac, malicious
3 cs-shared-component-tools,7.2.3,4f6262a6b9a0a46f14bc403afa0826ad, malicious
4 harthat-hash,1.3.3,f35135dc474791416507ee778ae0f801, malicious
5 @volcenjine@veplayer,89.3.5,001a7e727c0c239ca3687d2f6cd1d3ff, malicious
6 acribus-core,6.1.3,66208fc6e251b3fd3645dd1ed4d5a262, malicious
7 @volcenjine@tos-sdk,89.3.5,ce11589c5fda0e3e734f65a22beb349a, malicious
8 raptor-library,100.0.5,a74e8c485bf59a89b89618de44d95d61, malicious
9 seatalk-openapi,6.5.8,9386080c72bdef658e5671c66b14b15c, malicious
10 distube-fast,5.0.2,c43b197b738ccac725f48b87111dfd6d, malicious
11 paypal-js-root,1.0.0,d7ad93049df23b8fa91d78165deb9a4a, malicious
12 valid-ip-ban,3.3.3,f2170d3d4589644efff8b235962c7da3, malicious
13 rollup-plugin-hotreload,6.5.8,79428d0f2ea777ac3b80968bdfe1580f, malicious
14 peritter,1.1.0,b0cb801e8f42c410fc951fa1a523c0be, malicious
15 peritter,1.0.0,14858cbae3206bdda353e241c2aeb9fa, malicious
16 gnache-cli,6.12.2,5493603afc9fb3113d270eef428e40a4, malicious
17 playwright-1.47,10.1.8,befd7a855d6848d36cc6603171a89170, malicious
18 docu-scripts,0.0.0,02b9c3cc672b71b6b285a1fd230b9cc3, malicious
19 deploy-pages,4.3.0,8595ae25b1726b4053b84a4fd94f1839, malicious
```

Giai đoạn 2: Xử lý thông tin gói

- Gói npm đầu vào cho file tổng hợp .csv của các mô hình phải ở 2 dạng sau:
 - + @scope@package_name
 - version
 - package.json
 - + package_name
 - version
 - package.json

- **process_data.py** tái cấu trúc các thư mục con trong thư mục hiện tại và di chuyển chúng vào một thư mục đầu ra (output). Sau đó, ta duyệt qua tất cả các thư mục cha trong thư mục hiện tại. Nếu tên của thư mục cha bắt đầu bằng ký tự "@", ta sẽ duyệt qua các thư mục con của nó, tạo một tên thư mục mới bằng cách kết hợp tên thư mục cha và tên thư mục con, rồi di chuyển nội dung của các thư mục con vào thư mục mới trong thư mục output, sau đó xóa thư mục con cũ. Ngược lại, nếu tên thư mục cha không bắt đầu bằng ký tự "@", mã sẽ di chuyển toàn bộ nội dung của thư mục cha vào thư mục output mà không thay đổi tên.

```

import os
import shutil

# Tạo thư mục output nếu chưa có
output_dir = "output"
os.makedirs(output_dir, exist_ok=True)

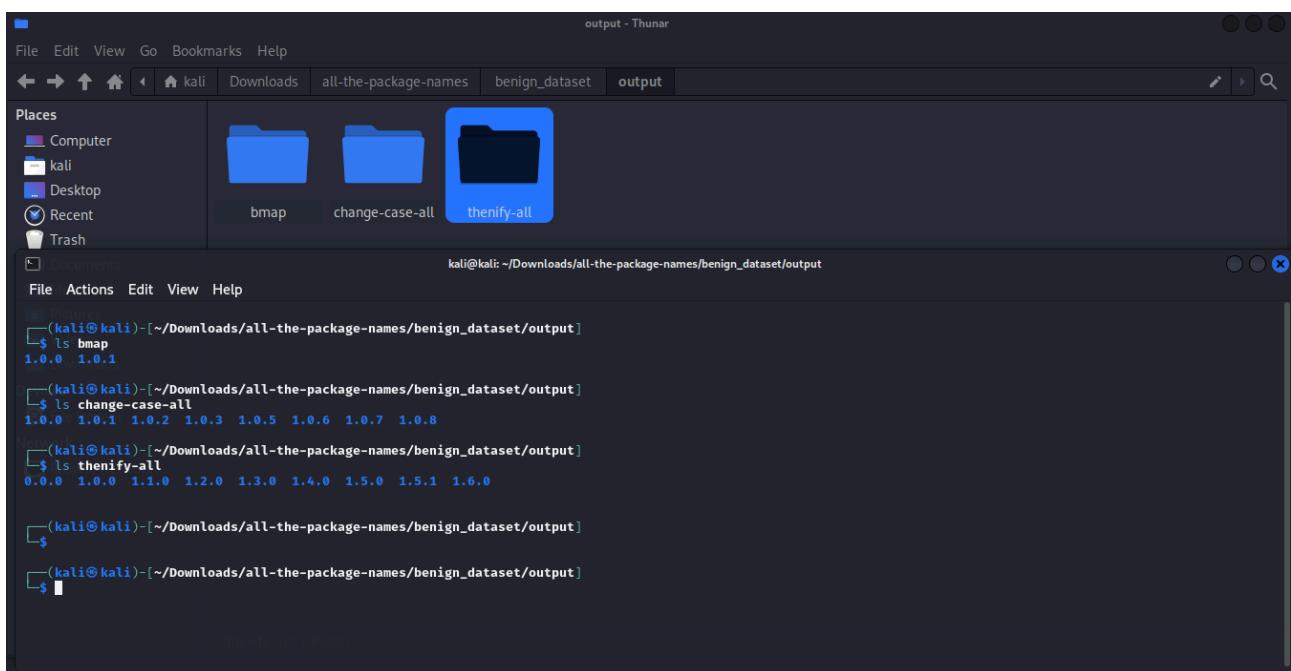
# Duyệt qua tất cả các thư mục cha trong thư mục hiện tại
for parent_dir in os.listdir('.'):
    if os.path.isdir(parent_dir):
        # Lấy tên thư mục cha
        parent_name = os.path.basename(parent_dir)
        # Nếu tên thư mục cha bắt đầu bằng @@
        if parent_name.startswith('@'):
            # Duyệt qua các thư mục con trong thư mục cha
            for dir in os.listdir(parent_dir):
                dir_path = os.path.join(parent_dir, dir)
                if os.path.isdir(dir_path):
                    # Tạo tên thư mục mới bằng cách kết hợp tên thư mục cha và thư mục con
                    new_name = f"{parent_name}@{dir}"
                    new_dir_path = os.path.join(output_dir, new_name)
                    # Tạo thư mục con trong thư mục output
                    os.makedirs(new_dir_path, exist_ok=True)
                    # Di chuyển nội dung từ thư mục con cũ sang thư mục mới trong output
                    for filename in os.listdir(dir_path):
                        shutil.move(os.path.join(dir_path, filename), new_dir_path)
                    # Xóa thư mục con cũ sau khi đã di chuyển
                    os.rmdir(dir_path)
                    print(f"Đã tạo thư mục mới và di chuyển nội dung: {dir_path} -> {new_dir_path}")
        else:
            # Nếu thư mục cha không bắt đầu bằng "@", di chuyển nguyên thư mục con vào thư mục output
            parent_output_dir = os.path.join(output_dir, parent_name)
            os.makedirs(parent_output_dir, exist_ok=True)
            for filename in os.listdir(parent_dir):
                shutil.move(os.path.join(parent_dir, filename), parent_output_dir)
            print(f"Giữ nguyên thư mục con trong thư mục cha: {parent_dir} -> {parent_output_dir}")

```

```

kali㉿kali:[~/Downloads/all-the-package-names/benign_dataset]
└─$ python3 process_data.py
Giữ nguyên thư mục con trong thư mục cha: change-case-all → output/change-case-all
Giữ nguyên thư mục con trong thư mục cha: output → output/output
Giữ nguyên thư mục con trong thư mục cha: thenify-all → output/thenify-all
Giữ nguyên thư mục con trong thư mục cha: bmap → output/bmap

```



B.2. Thí nghiệm 1: Phân loại các gói mới xuất bản

B.2.1. Ngữ cảnh

- Thí nghiệm này mô phỏng thực tế trong việc phát hiện mã độc tự động trên npm registry, nơi hàng trăm gói phần mềm mới được công bố mỗi ngày. Mục tiêu của thí nghiệm là đánh giá hiệu quả của Amalfi.

B.2.2. Kịch bản

- Ngày đầu tiên:
 - + Các bộ phân loại được huấn luyện trên dữ liệu cơ bản (basic corpus) bao gồm các gói độc hại và lành tính đã biết.
 - + Áp dụng các bộ phân loại và clone detector trên tập hợp N1, bao gồm các phiên bản gói mới được công bố trong ngày mà ta tập hợp được.
 - + Phân tích kết quả để tạo ra tập P1, chứa các gói bị gắn cờ bởi ít nhất một công cụ.
 - + Tiến hành loại bỏ dương tính giả thông qua: reproducer và kiểm tra thủ công, tạo thành 1 tập TP1
- Ngày thứ hai và các ngày tiếp theo:
 - + Các bộ phân loại được huấn luyện lại trên dữ liệu bao gồm:
 - Basic corpus.
 - Tập N1 (đã được phân loại ngày trước).
 - TP1: gói mã độc đã xác nhận.
 - + Giả định rằng các gói không bị gắn cờ bởi bất kỳ công cụ nào là lành tính.
 - + Lặp lại quy trình:
 - Áp dụng trên tập hợp N2, gồm các gói công bố mới trong ngày.
 - Phân loại tập P2 (các gói bị gắn cờ).

- Kiểm tra thủ công và loại các gói dương tính giả bằng reproducer vào TP2.
- + Quy trình này tiếp tục qua từng ngày, với dữ liệu mới từ các tập N1, N2,... liên tục được bổ sung vào quá trình huấn luyện.

B.2.3. Triển khai

- Thu thập và xử lý dữ liệu:
 - + Tắt cả các gói công khai trên npm registry trong một tuần được thu thập và tổ chức theo ngày.
 - + Dữ liệu cơ bản (basic corpus) được chuẩn bị bao gồm danh sách các gói mã độc đã biết.
- Áp dụng công cụ phát hiện:
 - + Bộ phân loại (classifiers): Ba bộ phân loại được huấn luyện trên dữ liệu cơ bản và dữ liệu từ các ngày trước đó.
 - + Công cụ phát hiện sao chép (clone detector): Áp dụng trên tập dữ liệu mỗi ngày để tìm các bản sao của gói mã độc.
- Phân tích và kiểm tra:
 - + Loại bỏ dương tính giả: Sử dụng công cụ tái tạo tự động để giảm số lượng gói cần kiểm tra thủ công.
 - + Mỗi gói mã độc được xác nhận bởi một tác giả và được đổi chiêu bởi tác giả còn lại.
- Cải thiện: Quy trình lặp lại qua từng ngày, với dữ liệu được cập nhật để cải thiện độ chính xác của các công cụ.
- Kết quả chạy ở N1:
 - + Tạo training set của basic corpus

```

$ python3 organize.py basic_corpus training_sets
Processed evocateur-libnmpublish/99.3.5 -> Training_sets/evocateur-libnmpublish/99.3.5/change-features.csv
Processed xstate-viz/8.5.1 -> training_sets/xstate-viz/8.5.1/change-features.csv
Processed sctc-processor/0.0.0 -> training_sets/sctc-processor/0.0.0/change-features.csv
Processed ng-ul-library/1.0.12 -> training_sets/ng-ul-library/1.0.12/change-features.csv
Processed ng-ul-library/1.0.970 -> training_sets/ng-ul-library/1.0.970/change-features.csv
Processed ng-ul-library/1.0.984 -> training_sets/ng-ul-library/1.0.984/change-features.csv
Processed ng-ul-library/1.0.956 -> training_sets/ng-ul-library/1.0.956/change-features.csv
Processed ng-ul-library/1.0.959 -> training_sets/ng-ul-library/1.0.959/change-features.csv
Processed ng-ul-library/1.0.933 -> training_sets/ng-ul-library/1.0.933/change-features.csv
Processed ng-ul-library/1.1.13 -> training_sets/ng-ul-library/1.1.13/change-features.csv
Processed ng-ul-library/1.1.31 -> training_sets/ng-ul-library/1.1.31/change-features.csv
Processed ng-ul-library/1.0.905 -> training_sets/ng-ul-library/1.0.905/change-features.csv
Processed ng-ul-library/1.0.958 -> training_sets/ng-ul-library/1.0.958/change-features.csv
Processed ng-ul-library/1.0.965 -> training_sets/ng-ul-library/1.0.965/change-features.csv
Processed ng-ul-library/1.0.948 -> training_sets/ng-ul-library/1.0.948/change-features.csv
Processed ng-ul-library/1.1.10 -> training_sets/ng-ul-library/1.1.10/change-features.csv

```

- + Tạo các mô hình phân loại:

Đồ án môn học – NT521 - Báo cáo tổng kết

The terminal window shows the following file structure and command:

```
(testenv) (kali㉿kali)-[~/Documents/training]$ python3 train_classifier.py svm basic_corpus.csv training_sets -o trained svm.pkl
```

- + Dự đoán các mô hình để dự đoán gói npm ở ngày 1:

The terminal window shows the output of the prediction script and a command to run it again:

```
(testenv) (kali㉿kali)-[~/Documents/training]$ python3 prediction.py training_n1 trained_dt.pkl output_dt.csv
```

Đồ án môn học – NT521 - Báo cáo tổng kết

```
■ output_nb.csv
333 build-onchain-apps,1.0.0,Malicious
334 daun124wdsa8,23.6.1,Benign
335 journey-client-reactor,10.9.9,Benign
336 eslint-plugin-fulfillment-lint,7.1.9,Benign
337 eslint-plugin-fulfillment-lint,7.3.9,Benign
338 puppeteer-harr,1.1.2,Malicious
339 sovryn,0.0.0,Benign
340 seller-core,6.5.8,Benign
341 ssc-ui-vue,7.3.10,Benign
342 prettier-plugin-kimi-i18next,6.5.8,Malicious
343 web-eth,4.10.0,Benign
344 seller-base,6.5.8,Malicious
345 sap-auth,0.0.0,Benign
346 @mosfe@beam-git-util,1.0.0,Benign
347 ganach-cli,6.12.2,Malicious
348 ethers-multcall,0.2.3,Malicious
349 eslint-config-sipplint,7.1.9,Malicious
350 @fullcalendar@core,6.1.15,Benign
351 metro-core,0.81.0,Benign
352 angular-location-update,0,Benign
353 apc-protobuf,6.1.3,Malicious
354 api-cache-worker,6.1.3,Benign
355 metamodel-editor,99.9.9,Benign
356 metamodel-editor,99.10.1,Malicious
357 @dp-bpsc-tiktok@ranking,1.0.0,Benign
358 aem-core-react-components,6.9.9,Benign
359 aem-core-react-components,6.8.9,Benign
360 string-process-mate,1.0.0,Benign
361 @copilot-web-widgets@ai-writer,60.0.0,Malicious
362 etherjs-util,7.1.5,Benign
363 puppeteerscroll-down,2.0.0,Malicious
364 marketing-jest-cli,6.5.8,Benign
365 web3-toekn,1.0.6,Malicious
366 wynn-and-encore,100.0.5,Malicious
367 web-bzz,1.10.3,Benign

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

└─(testenv)─(kali㉿kali)-[~/Documents/training]
$ python3 prediction.py training_n1 trained_nb.pkl output_nb.csv
```

```
output_svm.csv
685 react-datepicker-plus,2.3.9,Benign
686 react-datepicker-plus,2.2.7,Benign
687 react-datepicker-plus,2.1.0,Benign
688 react-datepicker-plus,2.3.0,Benign
689 react-datepicker-plus,2.1.3,Benign
690 cargo-hub-ui-api-internal,420.6.10,Malicious
691 pnpm,9.15.0,Malicious
692 @pnpm@manifest-utils,1000.0.1,Malicious
693 eslint-plugin-mkt-bff,6.5.8,Malicious
694 package-manager-manager,0.2.0,Malicious
695 stripe-testfb-v3,1.0.0,Malicious
696 @pkgr@core,0.1.1,Malicious
697 bootstrap-srio,4.0.0,Benign
698 @platform-ui-kit@components-library-react,9.9.1,Malicious
699 web-live-sdk,400.10.1,Malicious
700 @maps-bc@i18n,1.0.0,Malicious
701 puppeteer-capture,1.1.1,Malicious
702 babel-plugin-method-version,5.3.5,Malicious
703 commitlint-config-marketing,5.3.5,Malicious
704 raptor-library,100.0.5,Malicious
705 puppeteerrecordr,1.0.7,Malicious
706 scoped-regex,3.0.0,Malicious
707 vue-midata,9.5.2,Malicious
708 ssc-upload-react,5.1.9,Malicious
709 sap-b,0.0.0,Malicious
710 getsolara,1.0.0,Malicious
711 getsolara,1.0.1,Malicious
712 precode.js,0.1.2,Malicious
713 precode.js,1.0.0,Malicious
714 precode.js,1.0.1,Malicious
715 precode.js,1.1.0,Malicious
716 precode.js,1.0.2,Malicious

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Processed pyramid-proportion@1.0.1: Malicious
Processed pyramid-proportion@1.0.2: Malicious
Processed pyramid-proportion@1.0.4: Malicious
Classification completed. Results saved to output_svm.csv

(testenv)-(kali㉿kali)-[~/Documents/training]
$
```

- + Chạy chương trình clone detector

Đồ án môn học – NT521 - Báo cáo tổng kết

```
1 package,version,hash,clone_detect
2 airpay-js-sdk,7.1.9,d8d709d7608d97b587976732c7a2112a,yes
3 adminconsole,1.0.0,b8a57af21d7e349e30cc37ea88710ff2,yes
4 @alaska-its@design-tokens,1.0.1,e682d2b82a3b3b636628801efb5a8ff8,yes
5 aem-core-react-components,6.9.9,9d68aa33d198a9fc56aa5ed3fd1c0b5b,yes
6 aem-core-react-components,6.8.9,b0ad3380e2ec1997b843a6112b1033ef,yes
7 @oxengine@xmlrpc,1.3.18,001f76da5b0c9aab6fec868bcc8103a,yes
8 aa-bundler,0.0.2,2fb74214bac1730db0bec219ef778ee2,yes
9 anumanu2324,1.0.1,2c9046bd7f462dfaf97f081e827bbaa9,yes
10 apc-admin-utils,6.1.3,566fd8399572d2051a9c1e25632944af,yes
11 @afound@common,10.10.12,83e274ed0c8b0c7444c9b20b8997d7f2,yes
12 apa-components,6.1.3,0eeddeb8e66e1036f3e31d5ac021a08e,yes
13 @al-ui@useappinsights,1.0.1,9921bd715fc77b3ea4790ea4854762,yes
14 elkeid,9.3.5,15a8121e0a4d9e2258c4abe304cffald,yes
15 cdntodister,20.0.0,f22f1d422531c01b3e88589c205b19ab,yes
16 airwallex-platform-onboarding-sdk-demo-react,1.0.0,fc187300157b260f4db04c3229ce17d3,yes
17 com.microsoft.mrtk.graphics.tools.unity,0.4.0,199a65993c36a8b08ca0f6eca067162a,yes
18 com.microsoft.mrtk.graphics.tools.unity,4.0.0,0a4e55ffd0b6145464670c5f2aaa8e9e,yes
19 configure-pages,4.3.0,d56fe84743af886455c768fa87d99784,yes
20 mona-plugin-events,99.11.18,4390efd11f39b5c21133ed3dbe2dabbe,yes
21 custom-ui-extension-template,1.0.1,d7c450ada463984fe7d7038518a13adc,yes
22 debugr1,3.1.1,0c50970b60f091635f89076bb6293faf,yes
23 com.unity.entities,9.4.5,4e22d520cafca73113b2648cd90d39c6,yes
24 com.unity.entities,7.1.2,537f48c4a778c13c58bff728fdc2b834,yes
25 bloxupgrade,1.0.0,44295da4604f0889b184c3484d09b8dd,yes
26 cdp-wallet-manager,1.0.0,be8513b2ff9d908d2e48045f5a58764b,yes
27 cdp-wallet-manager,0.1.0,7fc2b88ffffdd33d5a23d3e06379105c,yes
28 appetize-cli,1.2.0,9fbb60610b249bb6d709f7eeb963b287,yes
29 appetize-cli,1.3.0,a589ce78f6a6fe9764a00fec3b44d0a1,yes
30 peritter,1.0.0,14858cbae3206bdda353e241c2aeb9fa,yes
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
└─(testenv)─(kali㉿kali)─[~/Documents/training]
$ python3 clone_detect.py label.csv N1 clone-detect.csv
Reading basic corpus...
Found 662 malicious hashes in the basic corpus.
Hashing new packages...

Detecting clones...
Clone detected: airpay-js-sdk@7.1.9 matches a malicious package in the basic corpus.
```

- + Chạy chương trình reproducer để tạo ra file csv dự đoán cuối cùng

The screenshot shows a terminal window with the following content:

```
output.csv
1 package,version
2 airwallex-platform-onboarding-sdk-demo-react,1.0.0
3 adminconsole,1.0.0
4 action-deploy,2.0.1
5 pm-controls,0.0.40
6 acribus-core,6.1.3
7 apa-components,6.1.3
8 pm-controls,0.0.38
9 @abdallaeg/sap_access,0.0.0
10 anumanu2324,1.0.1
11 apache-airflow-ui,99.9.9
12 acp-docs,6.1.3
13 ap-components-react,3.15.9
14 pm-controls,0.0.46
15 ap-components-react,3.9.9
16 @0xengine/meow,1.2.7
17 pm-controls,0.0.39
18 @add-wallet-exchange/type-api-data,1.0.0
19 ap-components-react,3.16.9
20 ads-common-utils,7.1.9
21 airpay-js-sdk,7.1.9
22 acces-react,99.3.5
23 @al-ui/useappinsights,1.0.1
24 acm-nano-logger-fe,1.0.1
25 pm-controls,0.0.48
26 apache-airflow-ui,9.9.9
27 @0xengine/xmlrpc,1.3.18
28 @add-wallet-exchange/reset,1.0.0
29 actions-languageservices,2.0.9
30 pm-controls,0.0.41
31 @aftersale/react-eva,1.1.1
32 apc-admin-utils,6.1.3
33 pm-controls,0.0.42
```

Below the code listing, there is a navigation bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS.

```
(testenv)-(kali㉿kali)-[~/Documents/training]
$ python3 reproduce.py output-total.csv output output.csv
Processing airwallex-platform-onboarding-sdk-demo-react@1.0.0...
Failed to reproduce airwallex-platform-onboarding-sdk-demo-react@1.0.0.
Processing adminconsole@1.0.0...
```

B.3. Thí nghiệm 2: Phân loại dữ liệu đã được dán nhãn

B.3.1. Ngữ cảnh

- Trong thí nghiệm đầu tiên cung cấp cái nhìn sâu sắc về hiệu suất của phương pháp tiếp cận trong điều kiện thực tế, đặc biệt là tỷ lệ dương tính giả, nhưng mô hình không cho chúng ta biết nhiều về các trường hợp âm tính giả. Vì vậy thí nghiệm 2 này sẽ đo lường độ chính xác (**precision**) và độ nhạy (**recall**) trên tập dữ liệu cơ bản.

B.3.2. Kịch bản

- Tổng hợp một tập dữ liệu gồm các gói npm lành tính và độc hại (**malicious hay benign**).
- Thực hiện phương pháp **10-fold cross-validation**.
- Đánh giá 3 modules trên các tập dữ liệu đã chia
- Đánh giá hiệu suất dựa trên **Precision** và **Recall**.

B.3.3. Triển khai

- Thực hiện phương pháp **10-fold cross-validation**.
 - + **Chia dữ liệu** thành 10 phần (folds) có kích thước gần bằng nhau.

Tạo 1 thư mục chứa các fold sau khi thực hiện 10-fold cross-validation.

```
# 8. Tao thu muc luu ket qua
output_dir = 'cross-validation'
os.makedirs(output_dir, exist_ok=True)
```

```
# 9. Vòng lặp cross-validation
for fold, (train_index, test_index) in enumerate(kf.split(X_encoded, y)):
    print(f"Processing Fold {fold + 1}")

    # Chia tập train và test
    X_train, X_test = X_encoded.iloc[train_index], X_encoded.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Khớp đúng tập test gốc
    X_test_original = X.iloc[test_index].reset_index(drop=True)

    # Tạo thư mục cho từng fold
    fold_dir = os.path.join(output_dir, f'fold_{fold + 1}')
    os.makedirs(fold_dir, exist_ok=True)

    for model_name, model in models.items():
        # Huấn luyện mô hình
        model.fit(X_train, y_train)

        # Dự đoán nhãn
        y_pred = model.predict(X_test)

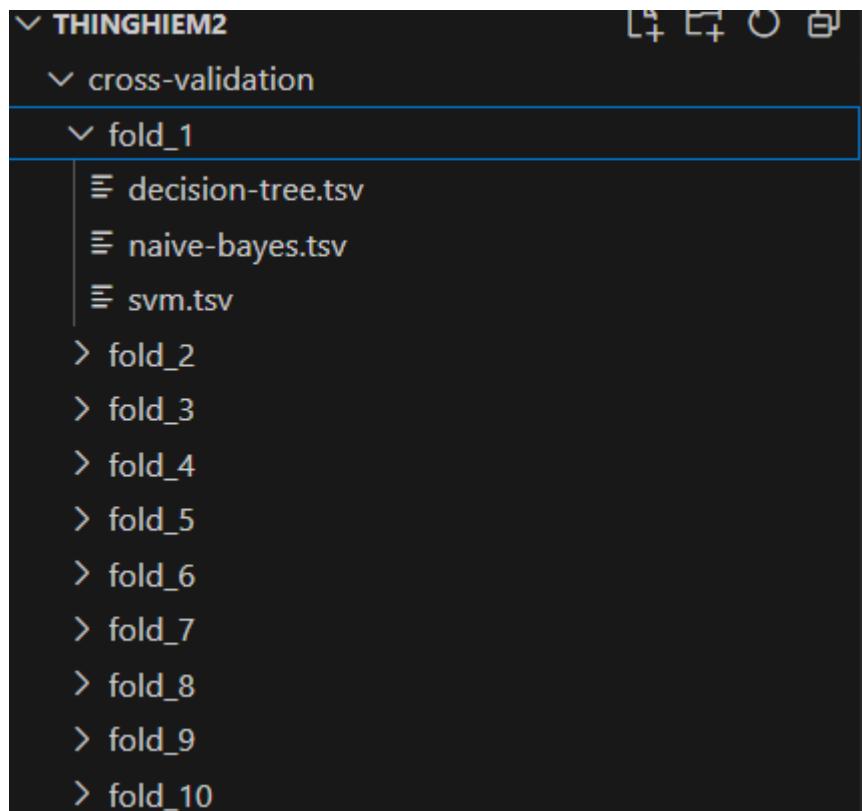
        # Tính độ chính xác
        accuracy = accuracy_score(y_test, y_pred)
        print(f'{model_name} Accuracy in Fold {fold + 1}: {accuracy:.4f}')

        # Kết hợp nhãn dự đoán với tập test gốc
        results = X_test_original.copy()
        results['analysis'] = pd.Series(y_pred).map({1: 'benign', 0: 'malicious'})

        # Ghi kết quả ra file .tsv
        results_file = os.path.join(fold_dir, f'{model_name}.tsv')
        results.to_csv(results_file, sep='\t', index=False, header=True)

print("Hoàn tất quá trình huấn luyện và lưu kết quả.")
```

- Vòng lặp này sẽ phân chia dữ liệu thành các fold. Mỗi fold chứa chỉ số mẫu cho tập huấn luyện và chỉ số mẫu cho tập kiểm tra.
- Trong quá trình đó, đồng thời chia tập train và test. Sau đó tạo thư mục cho từng fold có cấu trúc.
- Vòng lặp tiếp theo sẽ dùng cho huấn luyện và đánh giá từng mô hình.
- Kết quả của chương trình sẽ cho kết quả như sau:



+ Huấn luyện và kiểm tra:

- Thực hiện lặp lại 10 lần (tương ứng với số fold): Mỗi lần lặp chọn 1 fold làm tập kiểm tra (test set). Sau đó sử dụng 9 fold còn lại làm tập huấn luyện (training test).

```
# Khởi tạo các mô hình
models = [
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Naive Bayes': GaussianNB(),
    'SVM': SVC(kernel='rbf', random_state=42, class_weight='balanced', C=1.0, gamma='scale') # Đổi kernel sang RBF
]

# Sử dụng StratifiedKFold cho 10-fold cross-validation
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

- Huấn luyện mô hình trên tập huấn luyện và đánh giá trên tập kiểm tra.

```
# Sử dụng StratifiedKFold cho 10-fold cross-validation |
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Hàm đánh giá mô hình với precision và recall
for model_name, model in models.items():
    scores = cross_validate(model, X, y, cv=kf,
                           scoring={'precision': make_scorer(precision_score),
                                     'recall': make_scorer(recall_score)})

# In kết quả cho từng mô hình
print(f'{model_name}')
print(f'Mean Precision: {scores["test_precision"].mean():.2f}')
print(f'Mean Recall: {scores["test_recall"].mean():.2f}\n')
```

- Sử dụng **StratifiedKFold** Chia dữ liệu thành 10 fold, đảm bảo tỷ lệ giữa các lớp trong nhãn (malicious và benign) được giữ nguyên trong mỗi fold (stratification).
- Lặp qua từng mô hình và thực hiện cross-validation cho mô hình.
- Đánh giá bằng các thước đo:
 - + **Precision (precision_score)**: Tỉ lệ mẫu dự đoán đúng là **malicious** so với tất cả các mẫu dự đoán là **malicious**.
 - + **Recall (recall_score)**: Tỉ lệ mẫu dự đoán đúng là **malicious** so với tất cả các mẫu thực sự là **malicious**.

=> Kết quả của thí nghiệm 2:

```
PS D:\HK1_2025\Laptrinhantoan\doan\Thiinghiem2> python .\danhgia.py
[Decision Tree]
Mean Precision: 0.86
Mean Recall: 0.81

[Naive Bayes]
Mean Precision: 0.63
Mean Recall: 0.48

[SVM]
Mean Precision: 0.66
Mean Recall: 0.68
```

Đánh giá:

- **Decision Tree** hoạt động tốt nhất trong cả Precision và Recall, cân bằng tốt giữa hai yếu tố. Đây là tốt nhất nếu muốn phát hiện mẫu **malicious** với độ chính xác và độ nhạy cao.

- **Naive Bayes** kém hiệu quả so với các mô hình khác. Mức Precision và Recall đều thấp, không đủ tin cậy để áp dụng cho bài toán yêu cầu phát hiện **malicious** với độ chính xác cao.
- **SVM** có kết quả trung bình, phù hợp nếu cần một mô hình thay thế nhưng không đạt hiệu quả như Decision Tree. Precision và Recall tương đối cân bằng, nhưng vẫn chưa tối ưu.

C. Kết quả thực nghiệm

C.1. Kết quả thực nghiệm thí nghiệm 1

- Ở ngày 1, tiến hành kiểm tra tập hợp N1 gồm 1136 gói npm:
 - + Thuật toán decision tree kiểm tra được 537 gói npm độc
 - + Thuật toán naive bayes kiểm tra được 278 gói npm độc
 - + Thuật toán svm kiểm tra được 635 gói độc
 - + Clone detector kiểm tra được 67 gói npm là bản sao
 - + Sau khi chạy reproducer, còn lại được 488 gói npm độc
 - + Thực tế có khoảng 398 gói npm độc hại
- Ở ngày 2, tiến hành kiểm tra tập hợp N2 gồm 982 gói npm:
 - + Thuật toán decision tree kiểm tra được 361 gói npm độc
 - + Thuật toán naive bayes kiểm tra được 98 gói npm độc
 - + Thuật toán svm kiểm tra được 381 gói độc
 - + Clone detector kiểm tra được 32 gói npm là bản sao
 - + Sau khi chạy reproducer, còn lại được 301 gói npm độc
 - + Thực tế có khoảng 275 gói npm độc hại
- Nhận xét:
 - + Có thể thấy thuật toán decision tree cho ra kết quả dự đoán là tốt nhất trong ba thuật toán.
 - + Mô hình dự đoán qua các ngày sau khi đã được cải thiện đã cho ra kết quả dự đoán càng ngày càng tốt hơn.
 - + Ưu điểm là thời gian train mô hình cũng như dự đoán là khá nhanh, nhưng ngược lại thì thời gian để chương trình phát hiện bản sao, vài tái tạo gói npm là khá lâu, nhất là thời gian của chương trình tái tạo gói npm là rất lớn nếu số lượng gói npm cần tái tạo là nhiều, đặc biệt là nếu số lượng gói npm dương tính giả cao.

C.2. Kết quả thực nghiệm thí nghiệm 2

```
PS D:\HK1_2025\Laptrinhantoan\doan\Thiinghiem2> python .\danhgia.py
[Decision Tree]
Mean Precision: 0.86
Mean Recall: 0.81

[Naive Bayes]
Mean Precision: 0.63
Mean Recall: 0.48

[SVM]
Mean Precision: 0.66
Mean Recall: 0.68
```

- Hiệu quả được đo lường thông qua các chỉ số như **Precision**, **Recall**, và **Accuracy**. Trong thí nghiệm này, các mô hình học máy như **Decision Tree**, **Naive Bayes**, và **SVM** được sử dụng:
 - + **Decision Tree** đạt Precision ~86% và Recall ~81%.
 - + **Naive Bayes** đạt Precision ~63% và Recall ~48%.
 - + **SVM** thường vượt trội nếu có nhiều đặc trưng phức tạp hơn (dữ liệu cần chuẩn hóa tốt).
- Các phép đo về độ chính xác (precision) và recall từ thí nghiệm kiểm tra chéo 10 lần (10-fold cross-validation) trên tập dữ liệu cơ bản, được tính trung bình qua tất cả mươi lần chạy. Tất cả các mô hình đều đạt độ chính xác rất cao, nhưng độ recall của Naive Bayes và SVM thì hơi thấp. Điều này là do mức ưu tiên (prior) thấp đối với các gói độc hại.
- Thí nghiệm thứ hai cho thấy rằng có thể tồn tại một số lượng lớn các âm tính giả, nhưng ít nhất các số liệu của cây quyết định (decision tree) có triển vọng, phù hợp với bài toán phát hiện gói npm độc hại.

D. Hướng phát triển

D.1. Tự động hóa toàn bộ quy trình

- Để tăng tính hiệu quả và giảm tải cho con người, một hướng phát triển quan trọng là tự động hóa toàn bộ các bước, từ thu thập dữ liệu, phát hiện mã độc, đến xác thực và báo cáo.
- Tính ứng dụng:
 - + Tiết kiệm thời gian và công sức kiểm tra thủ công.
 - + Giảm thiểu sai sót do yếu tố con người.
 - + Đảm bảo quy trình được thực hiện nhất quán trên quy mô lớn.

D.2. Đề xuất gói lành tính thay thế

- Đối với các gói bị phát hiện là độc hại, một hướng phát triển hữu ích là đề xuất các gói lành tính thay thế tương tự. Giúp người dùng nhanh chóng tìm được giải pháp thay thế.

- Tính ứng dụng:
 - + Cải thiện trải nghiệm người dùng bằng cách cung cấp lựa chọn thay thế ngay lập tức.
 - + Nâng cao giá trị của công cụ trong việc hỗ trợ cộng đồng lập trình.

D.3. Phát triển thành công cụ hoặc GUI

- Hiện tại, quy trình chủ yếu được triển khai qua dòng lệnh và kịch bản mã nguồn. Việc phát triển thành công cụ độc lập hoặc GUI thân thiện sẽ giúp người dùng dễ tiếp cận hơn.
- Tính ứng dụng:
 - + Mở rộng đối tượng sử dụng, bao gồm cả các nhà phát triển không quen thuộc với công cụ dòng lệnh.
 - + Dễ dàng tích hợp vào quy trình làm việc hiện có của tổ chức hoặc cá nhân.
 - + Tăng cường khả năng tương tác, trực quan hóa thông tin và báo cáo.

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File **.PDF**. Tập trung vào nội dung, không mô tả lý thuyết.
- Đặt tên theo định dạng: [Mã lớp]-Project_Final_NhomX_Madetai. (trong đó X và Madetai là mã số thứ tự nhóm và Mã đề tài trong danh sách đăng ký nhóm đồ án).
Ví dụ: [NT521.N11.ANTT]-Project_Final_Nhom03_CK01.
- Nếu báo cáo có nhiều file, nén tất cả file vào file **.ZIP** với cùng tên file báo cáo.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

HẾT