ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Модель загроз ІС на основі ШІ

Кваліфікаційна робота
Другий (магістерський) рівень

Автор:
Проценко А.С.,
студ. гр. КСМм-23-1

Керівник:
Федорченко В.М.,
доц. каф. ЕОМ

АКТУАЛЬНІСТЬ

- Високі вимоги до інформаційної безпеки та зростання кількості кібератак на інформаційні системи
- Традиційні методи кібербезпеки, такі як брандмауери, антивірусне програмне забезпечення та системи виявлення загроз, хоча й залишаються важливими, не здатні повною мірою відповідати новим викликам
- Підвищення ступеню захисту при використанні ШІ у системах виявлення вторгнень

МЕТА І ЗАДАЧІ РОБОТИ

Метою цієї роботи є аналіз існуючих підходів до побудови моделей загроз на основі ШІ та розробка власної моделі для підвищення рівня безпеки ІС.

Задачею даної кваліфікаційної роботи є створення прототипу нейронної мережі здатної аналізувати інформацію, що до неї надходить, та попереджати користувача про небезпеку.

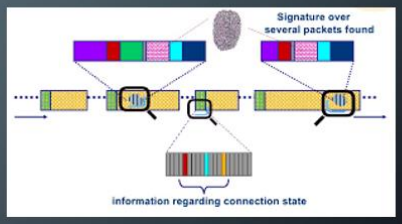
Завдання:

- обрати тип ШІ
- виділити певні підходи реалізації прототипу
- описати систему із якою працює нейронна мережа
- розробити прототип моделі загроз

3

ІСНУЮЧИ МЕТОДИ ВИЯВЛЕННЯ ЗАГРОЗ

- Сигнатурний аналіз
- Контроль доступу та управління правами
- Сканування вразливостей
- Аналіз поведінки
- Аудит журналів



4

МОЖЛИВІ РІШЕННЯ ПРОБЛЕМ

- Використання нейронних мереж для навчання відомим загрозам, та їх автоматизації із самонавчанням.
- Постійне сканування системи із негайним попередженням користувача про наявність слабкого місця чи можливість атаки.
- Автоматичний аналіз нових записів у журналах, після спрацювань правил системи

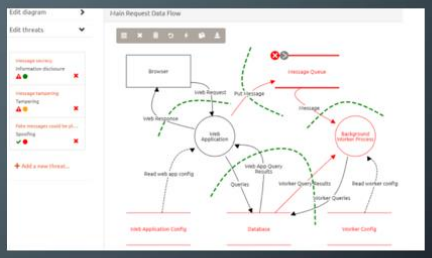
5

ІСНУЮЧІ МОДЕЛІ ЗАГРОЗ

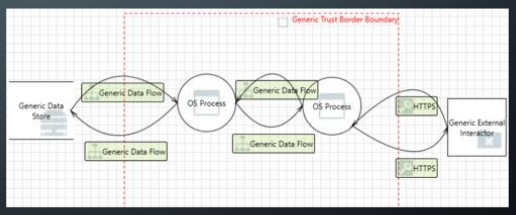
6

АНАЛІЗ ПРОГРАМНИХ ПРОДУКТІВ ДЛЯ МОДЕЛЮВАННЯ ЗАГРОЗ

- OWASP Threat Dragon



- Microsoft Threat Modeling Tool



7

ВИКОРИСТАННЯ ШІ ДЛЯ ПОШУКУ ЗАГРОЗ

Переваги

- Автоматизація
- Самонавчання
- Швидкість
- Точність

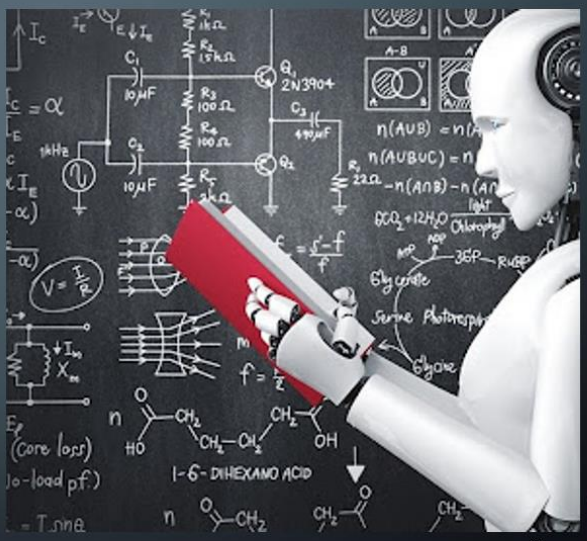
Недоліки

- Неякісні дані
- Складність інтерпретації
- Значні обчислювальні ресурси

8

КАТЕГОРІЇ ШІ

- Машинне навчання
- Глибинне навчання
  - Згорткові
  - Рекурентні
- Обробка природної мови



9

БІБЛІОТЕКИ ДЛЯ НАВЧАННЯ ШІ

- TensorFlow
- PyTorch
- Keras

10

СИСТЕМИ ВИЯВЛЕННЯ ВТОРГНЕНЬ

- Snort

- Suricata

11

ТЕХНІЧНІ ХАРАКТЕРИСТИКИ СЕРЕДОВИЩА ДЛЯ РОЗРОБКИ МОДЕЛІ ЗАГРОЗ

- GPU: NVIDIA Geforce RTX 2050
- CPU: Intel Core i5-12450H

- Тип ШІ: багатошарова персептронна нейронна мережа
- Бібліотеки: Keras та TensorFlow
- IDS: Snort

12

РОЗРОБКА МОДЕЛІ ЗАГРОЗ

СИСТЕМА ВИЯВЛЕННЯ ЗАГРОЗ

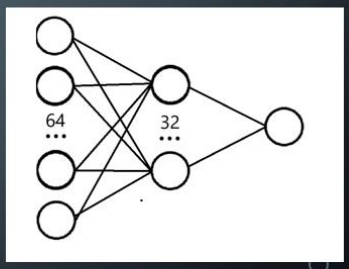
Першим кроком була підготовка і налаштування системи Snort. Так як система Snort була розроблена під дистрибутиви Linux, знадобилося встановити бібліотеку NPcap, що надає Snort можливість працювати із пакетами Windows. Після цього було виконано налаштування базової конфігурації. У конфігураційному файлі snort.conf були задані ключові параметри, такі як адреси локальної мережі, шляхи до журналів та правил.

```
100
101   # Path to your rules files (this can be a relative path)
102   # Note for Windows users:  You are advised to make this an absolute path,
103   # such as:  c:\snort\rules
104   var RULE_PATH c:\Snort\rules
105   var SO_RULE_PATH ../so_rules
106   var PREPROC_RULE_PATH c:\Snort\preproc_rules
107
108   # If you are using reputation preprocessor set these
109   # Currently there is a bug with relative paths, they are relative to where snort is
110   # not relative to snort.conf like the above variables
111   # This is completely inconsistent with how other vars work, BUG 89986
112   # Set the absolute path appropriately
113   var WHITE_LIST_PATH c:\Snort\rules
114   var BLACK_LIST_PATH c:\Snort\rules
115
```

13

СТВОРЕННЯ ШІ

Після налаштування Snort потрібно було створити нейронну мережу для аналізу загроз. Для побудови моделі були використані бібліотеки TensorFlow та Keras, яка, у результаті, отримувала вхідні дані у вигляді мережевих характеристик, таких як IP-адреси, порти чи протоколи. Архітектура мережі включала 3 шари: 2 вхідні шари із функціями активації ReLU та Dropout для запобігання перенавчанню, а також вихідний шар із сигмоідальною активацією для класифікації, чи є трафік загрозливим



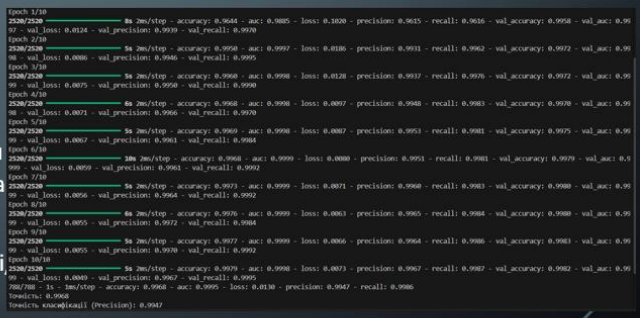
14

НАВЧАННЯ ШІ

Навчання проводилося на датасеті NSL-KDD, який містив 123 записи про нормальні та небезпечні дії у мережі.

Попередня обробка даних передбачала масштабування числових значень, кодування категоріальних ознак і розділення вибірки на навчальну та тестову

Завдяки великій кількості ознак при навчанні ця модель загроз працює водночас на транспортному та мережевому рівні, що дозволяє охопити великий спектр використання ІС.



СТВОРЕННЯ МОДЕЛІ ЗАГРОЗ

Інтеграцію Snort із нейронною мережею було організовано через аналіз журналів Snort у реальному часі. Для цього розроблено скрипт на Python, який читає файл alerts.txt, обробляє нові записи та витягує ключові ознаки.

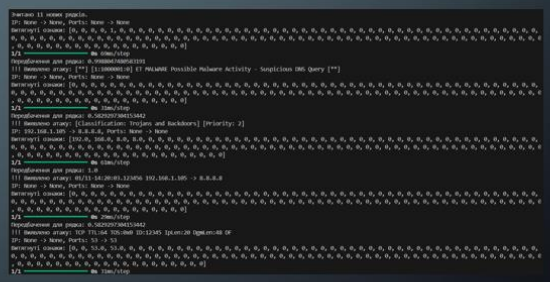
```
[**] [1:1000001:0] ET MALWARE Possible Malware Activity - Suspicious DNS Query [**]
[Classification: Trojans and Backdoors] [Priority: 2]
01/11-14:20:03.123456 192.168.1.105 -> 8.8.8.8
TCP TTL:64 TOS:0x0 ID:12345 IpLen:20 DgmLen:48 DF
Proto:UDP SrcPort:53 DstPort:53 Len:28

[**] [1:1000001:1] ICMP ping request [**]
[Classification: Attempted Information Leak] [Priority: 3]
12/30-14:23:45.456789 192.168.1.5 -> 192.168.1.10
ICMP TTL:64 TOS:0x0 ID:56789 IpLen:20 DgmLen:56
Type:8  Code:0  ID:100  Seq:1  ECHO
```

ТЕСТУВАННЯ МОДЕЛІ ЗАГРОЗ

Завантажена модель аналізує кожен запис і передбачає, чи є він загрозою. У разі виявлення атаки система виводить попередження, що дозволяє швидко реагувати на загрози

На завершальному етапі модель проходила тестування на реальних даних. Досліджувалася її здатність виявляти як відомі, так і нові загрози. Приклади готових записів про шкідливі атаки, так само як й дата сет NSL-KDD, були взяті з порталу Kaggle. За результатами тестування вдосконалювалася як модель нейронної мережі, так і правила Snort для зменшення помилкових спрацьовувань



17

МОЖЛИВІ ПОКРАЩЕННЯ

Модель не є ідеальною, тому завжди може бути покращена. Із можливих покращень можна виділити:

- покращення дружнього інтерфейсу, замість консолей та текстових файлів;
- додавання самонавчання до моделі ШІ.

18

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено модель захисту для виявлення та попередження вторгнень.

У результаті було виконано наступні дії:

- було проаналізовано можливі рішення для створення ефективнішої моделі загроз;
- було написано програму для навчання нейронної мережі на основі датасету NSL-KDD із 123 записами про нормальні та небезпечні дії у мережі;
- було написано скрипт для інтеграції ШІ у систему Snort для аналізу записів про дії у мережі, виявлення та передбачення вторгнень.

Розроблена модель загроз передбачає постійну роботу зі сканування записів, створених системою Snort та, у разі виникнення підозрілих дій у мережі, швидкого реагування із попередженням користувача про небезпеку.

19

АПРОБАЦІЯ

Публікації:

Тези: Проценко А.С., Федорченко В.М., «Using artificial intelligence to analyze security threats in information systems». Тези доповідей сьомої міжнародної науково-технічної конференції COMPUTER AND INFORMATIONAL SYSTEMS AND TECHNOLOGIES вересень 26-27, с. 43-45, 2024.

Стаття: Проценко А.С., Федорченко В.М., «MOBILE APPLICATION SECURITY ANALYSIS MODEL BASED ON ARTIFICIAL INTELLIGENCE»
Буде викладена у 1-ому томі 79-ого випуску «Системи управління, навігації та зв'язку»

20

ДОДАТОК Б

Програмний код кваліфікаційної роботи

Лістинг Б.1 – Скрипт для навчання нейронної мережі

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

def create_model(input_dim):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_dim,)),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])

    metrics = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        tf.keras.metrics.AUC(name='auc')
    ]

    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='binary_crossentropy',
        metrics=metrics
    )

    return model

def load_and_prepare_data():
    url =
"https://raw.githubusercontent.com/defcom17/NSL_KDD/master/KDDTr
ain+.txt"
    columns = ["duration", "protocol_type", "service", "flag",
"src_bytes", "dst_bytes", "land", "wrong_fragment",
               "urgent", "hot", "num_failed_logins",
"logged_in", "num_compromised", "root_shell", "su_attempted",
               "num_root", "num_file_creations", "num_shells",
"num_access_files", "num_outbound_cmds",
               "is_host_login", "is_guest_login", "count",
"srv_count", "serror_rate", "srv_serror_rate",
```

```
                "rerror_rate", "srv_rerror_rate",
"same_srv_rate", "diff_srv_rate", "srv_diff_host_rate",
                "dst_host_count", "dst_host_srv_count",
"dst_host_same_srv_rate", "dst_host_diff_srv_rate",
                "dst_host_same_src_port_rate",
"dst_host_srv_diff_host_rate", "dst_host_serror_rate",
                "dst_host_srv_serror_rate",
"dst_host_rerror_rate", "dst_host_srv_rerror_rate", "attack",
"last_flag"]

    data = pd.read_csv(url, names=columns)

    data['attack'] = data['attack'].apply(lambda x: 0 if x ==
'normal' else 1)

    categorical_columns = ['protocol_type', 'service', 'flag']
    data = pd.get_dummies(data, columns=categorical_columns)

    X = data.drop('attack', axis=1)
    y = data['attack']

    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test

if __name__ == "__main__":
    X_train, X_test, y_train, y_test = load_and_prepare_data()

    model = create_model(input_dim=X_train.shape[1])

    history = model.fit(
        X_train, y_train,
        epochs=10,
        batch_size=32,
        validation_split=0.2
    )

    loss, accuracy, precision, recall, auc =
model.evaluate(X_test, y_test, verbose=2)
    print(f"Точність: {accuracy:.4f}")
    print(f"Точність класифікації (Precision): {precision:.4f}")
    print(f"Повнота класифікації (Recall): {recall:.4f}")
    print(f"AUC: {auc:.4f}")

    model.save("attack_detection_model.h5")
    print("Модель успішно навчена та збережена!")
```

Лістинг Б.2 – Скрипт для аналізу записів про загрози

```
import time
import os
import tensorflow as tf
import numpy as np
import re

model = tf.keras.models.load_model("attack_detection_model.h5")

model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

def process_alert(line):
    features = extract_features(line)
    if features is None:
        return None

    input_data = np.array([features])
    prediction = model.predict(input_data)

    print(f"Передбачення для рядка: {prediction[0][0]}")

    is_attack = prediction[0][0] > 0.5
    return is_attack

def extract_features(line):
    try:
        ip_regex = r"(\d+\.\d+\.\d+\.\d+)"
        port_regex = r"SrcPort:(\d+) DstPort:(\d+)"

        ip_match = re.findall(ip_regex, line)

        port_match = re.findall(port_regex, line)

        if ip_match:
            src_ip = ip_match[0]
            dst_ip = ip_match[1] if len(ip_match) > 1 else None
        else:
            src_ip = dst_ip = None

        if port_match:
            src_port, dst_port = map(int, port_match[0])
        else:
            src_port = dst_port = None

        print(f"IP: {src_ip} -> {dst_ip}, Ports: {src_port} ->
{dst_port}")
```

```
        features = []

        if src_ip:
            features.append(float(src_ip.split('.')[0]))
            features.append(float(src_ip.split('.')[1]))
        else:
            features.append(0)

        if dst_ip:
            features.append(float(dst_ip.split('.')[0]))
            features.append(float(dst_ip.split('.')[1]))
        else:
            features.append(0)

        if src_port:
            features.append(float(src_port))
        else:
            features.append(0)

        if dst_port:
            features.append(float(dst_port))
        else:
            features.append(0)

        if "ET MALWARE" in line:
            features.append(1)
        else:
            features.append(0)

        while len(features) < 123:
            features.append(0)

        print(f"Витягнуті ознаки: {features}")

        return features
    except Exception as e:
        print(f"Error parsing line: {e}")
        return None

def monitor_alerts(file_path):
    print("Запуск моніторингу журналів...")
    file_position = 0

    while True:
        if not os.path.exists(file_path):
            print(f"Файл {file_path} не знайдено!")
            time.sleep(5)
            continue

        with open(file_path, "r") as file:
            file.seek(file_position)
            new_lines = file.readlines()
            file_position = file.tell()
```

```
            if new_lines:
                print(f"Зчитано {len(new_lines)} нових рядків.")

            for line in new_lines:
                is_attack = process_alert(line)
                if is_attack:
                    print(f"!!! Виявлено атаку: {line.strip()}")

            time.sleep(2)

if __name__ == "__main__":
    monitor_alerts("C:\\Snort\\log\\alerts.txt")
```