ДОДАТОК А Програмний код

А.1 Реалізація роботи застосунку

Лістниг 1.1 – Клас traffic_analyzer.py

```
import pyshark
from packet_sniffer.sniffer_mode import SnifferMode
from pcap_handler.feature_preparer import Preparer
from idnn.core import Predictor
from idnn.core import Trainer
import argparse

predictor = Predictor()
preparer = Preparer()
trainer = Trainer()
capture: pyshark.capture


def train(filename):
    trainer.train(filename)


def prediction_with_data(packet_data, additional_packet_data):
    predictor.make_prediction(packet_data, additional_packet_data)


def handle_sniffed_data(packet):
    preparer.start_feature_extraction(packet,
callback=prediction_with_data)


def live_traffic_capture(interface=None):
    global previous_packet_timestamp, iat, capture
    if interface is None:
        interface = SnifferMode.online.interface
    capture  =  pyshark.LiveCapture(interface=interface,  use_json=True,
include_raw=True)

    for packet in capture.sniff_continuously():
        handle_sniffed_data(packet)


def pcap_traffic(file=None):
    if file is None:
        file = SnifferMode.offline.interface
    capture = pyshark.FileCapture(file, use_json=True, include_raw=True)

    for packet in capture:
        handle_sniffed_data(packet)


def start_sniffer(mode):
    if mode == SnifferMode.online:
        live_traffic_capture()
    else:
```

```
            pcap_traffic()


    def cli(input_file, input_interface):
        if input_file is not None:
            pcap_traffic(file=input_file)
        else:
            live_traffic_capture(interface=input_interface)


    def main():
        parser = argparse.ArgumentParser()

        input_group = parser.add_mutually_exclusive_group(required=True)
        input_group.add_argument(
            "-i",
            "--interface",
            action="store",
            dest="input_interface",
            help="Cherimoya   will   capture   online   packet   data   from
INPUT_INTERFACE",
        )

        input_group.add_argument(
            "-f",
            "--file",
            action="store",
            dest="input_file",
            help="Cherimoya  will  capture  offline  packet  data  from  .pcap
INPUT_FILE",
        )

        args = parser.parse_args()

        try:
            cli(args.input_file, args.input_interface)
        except KeyboardInterrupt:
            capture.close()


    if __name__ == '__main__':
        main()
```

Лістниг 1.2 – Клас feature_extractor.py

```
    import pyshark
    from pcap_handler.packet_data import PacketData
    from pcap_handler.feature_extractor_base import BaseFeatureExtractor


    class FeatureExtractor(BaseFeatureExtractor):

        def get_data(self) -> PacketData:
            return self._get_packet_data()

        def assign_tcp_flags(self, packet):
            self._extract_tcp_flags(packet)

        def assign_flags_count(self, ack, syn, fin, rst, urg):
            self._extract_flags_count(ack, syn, fin, rst, urg)
```

```
        def extract_feature(self, **kwargs):
            self._extract_features(**kwargs)

        def assign_layer_flags(self, packet):
            self._determine_layer_flag(packet)
```

Лістниг 1.3 – Клас feature_extractor_base.py

```
    from enum import Enum
    from pcap_handler.packet_data import PacketData
    from pcap_handler.dynamic_features_extractor import DynamicLengthFeature
    from pcap_handler.dynamic_features_extractor import
DynamicMeasurementFeature


    class FeatureKwargsKey(Enum):
        proto = 'proto'
        iat = 'iat'
        length = 'length'
        measurements = 'measur'
        duration = 'duration'
        flow_duration = 'flow_duration'
        rate = 'rate'
        srate = 'srate'
        drate = 'drate'
        header_length = 'header_length'
        total_size = 'total_size'
        number = 'number'

    class BaseFeatureExtractor:

        def __init__(self):
            self.packet_data = PacketData()

        def _extract_features(self, **kwargs):
            for key in kwargs.keys():
                match key:
                    case FeatureKwargsKey.proto.value:

self.packet_data.set_protocol_type(kwargs[FeatureKwargsKey.proto.value])
                    case FeatureKwargsKey.iat.value:

self.packet_data.set_iat(kwargs[FeatureKwargsKey.iat.value])
                    case FeatureKwargsKey.length.value:
                        stat = kwargs[FeatureKwargsKey.length.value]
                        self._extract_statistic(stat)
                    case FeatureKwargsKey.measurements.value:
                        stat = kwargs[FeatureKwargsKey.measurements.value]
                        self._extract_measurements(stat)
                    case FeatureKwargsKey.duration.value:

self.packet_data.set_duration(kwargs[FeatureKwargsKey.duration.value])
                    case FeatureKwargsKey.flow_duration.value:

self.packet_data.set_flow_duration(kwargs[FeatureKwargsKey.flow_duration.valu
e])
                    case FeatureKwargsKey.rate.value:

self.packet_data.set_rate(kwargs[FeatureKwargsKey.rate.value])
                    case FeatureKwargsKey.drate.value:

self.packet_data.set_drate(kwargs[FeatureKwargsKey.drate.value])
```

```
                    case FeatureKwargsKey.srate.value:

self.packet_data.set_srate(kwargs[FeatureKwargsKey.srate.value])
                    case FeatureKwargsKey.header_length.value:

self.packet_data.set_header_length(kwargs[FeatureKwargsKey.header_length.valu
e])
                    case FeatureKwargsKey.total_size.value:
                        self.packet_data.total_size                          =
kwargs[FeatureKwargsKey.total_size.value]
                    case FeatureKwargsKey.number.value:

self.packet_data.set_number(kwargs[FeatureKwargsKey.number.value])

        def _extract_tcp_flags(self, packet):
            self.packet_data.set_fin_flag(packet)
            self.packet_data.set_syn_flag(packet)
            self.packet_data.set_rst_flag(packet)
            self.packet_data.set_psh_flag(packet)
            self.packet_data.set_ack_flag(packet)
            self.packet_data.set_ece_flag(packet)
            self.packet_data.set_cwr_flag(packet)

        def _determine_layer_flag(self, packet):
            self.packet_data.set_is_http(packet)
            self.packet_data.set_is_https(packet)
            self.packet_data.set_is_dns(packet)
            self.packet_data.set_is_smtp(packet)
            self.packet_data.set_is_ssh(packet)
            self.packet_data.set_is_irc(packet)
            self.packet_data.set_is_tcp(packet)
            self.packet_data.set_is_udp(packet)
            self.packet_data.set_is_telnet(packet)
            self.packet_data.set_is_dhcp(packet)
            self.packet_data.set_is_arp(packet)
            self.packet_data.set_is_icmp(packet)
            self.packet_data.set_is_ipv(packet)
            self.packet_data.set_is_llc(packet)

        def _extract_statistic(self, stat: DynamicLengthFeature):
            self.packet_data.min = stat.min_packets
            self.packet_data.max = stat.max_packets
            self.packet_data.avg = stat.mean_packets
            self.packet_data.total_sum = stat.sum_packets
            self.packet_data.std = stat.std

        def _extract_measurements(self, stat: DynamicMeasurementFeature):
            self.packet_data.set_magnitude(stat.magnitude)
            self.packet_data.set_radius(stat.radius)
            self.packet_data.set_covariance(stat.covariance)
            self.packet_data.set_variance(stat.var_ratio)
            self.packet_data.set_weight(stat.weight)

        def _extract_flags_count(self, ack, syn, fin, rst, urg):
            self.packet_data.ack_count = ack
            self.packet_data.syn_count = syn
            self.packet_data.fin_count = fin
            self.packet_data.rst_count = rst
            self.packet_data.urg_count = urg

        # GET PacketData
        def _get_packet_data(self) -> PacketData:
            return self.packet_data
```

Лістниг 1.4 – Файл feature_preparer.py

```
import dpkt
from pcap_handler.dynamic_features_extractor import DynamicFeatures
from pcap_handler.qualifier import TransportLayer
from pcap_handler.connectivity_features_extractor import
ConnectivityFeatures
from pcap_handler.flow_extractor import FlowExtractor
from pcap_handler.feature_extractor import FeatureExtractor
from pcap_handler.packet_data import AdditionalPacketData

class Preparer:
    def __init__(self):
        self.first_pack_time = 0
        self.flow_duration = 0
        self.rate, self.srate, self.drate = 0, 0, 0
        self.ethsize = []
        self.src_ports = {}
        self.dst_ports = {}
        self.tcp_flows = {}
        self.udp_flows = {}
        self.src_packet_count = {}  # saving the number of packets per
source IP
        self.dst_packet_count = {}  # saving the number of packets per
destination IP
        self.dst_port_packet_count = {}  # saving the number of packets
per destination port
        self.src_ip_byte, dst_ip_byte = {}, {}
        self.tcp_flow_flags = {}  # saving the number of flags for each
flow
        self.ips = set()  # saving unique IPs
        self.number_of_packets_per_transaction = 0  # saving the number
of packets per transaction
        self.incoming_pack = []
        self.outgoing_pack = []
        self.src_to_dst_pkt, self.dst_to_src_pkt = 0, 0
        self.additional_data = AdditionalPacketData()

    def erase_length(self):
        self.ethsize = []
        self.incoming_pack = []
        self.outgoing_pack = []

    def start_feature_extraction(self, packet, callback):
        ack_count, syn_count, fin_count, urg_count, rst_count = 0, 0, 0,
0, 0
        extractor = FeatureExtractor()
        ts = float(packet.sniff_timestamp)
        raw_data = packet.get_raw_packet()
        packet_frame_size = 0
        dpkt_eth = None
        if hasattr(packet.eth, 'trailer_raw'):
            trailer = packet.eth.trailer_raw[0]
            raw_hex = raw_data.hex().removesuffix(trailer)
            packet_bytes = bytes.fromhex(raw_hex)
            packet_frame_size = len(packet_bytes)
            dpkt_eth = dpkt.ethernet.Ethernet(packet_bytes)
        else:
            packet_frame_size = len(raw_data)
            dpkt_eth = dpkt.ethernet.Ethernet(raw_data)

        extractor.extract_feature(total_size=packet_frame_size)
```

```
            if hasattr(packet, 'ip') or hasattr(packet, 'ipv6') or
hasattr(packet, 'arp'):
                self.ethsize.append(packet_frame_size)
                srcs = {}
                dsts = {}
                if len(self.ethsize) % 20 == 0:
                    packet_length_feature                                =
DynamicFeatures.dynamic_calculation(ethsize=self.ethsize)
                    packet_measur_stats                                  =
DynamicFeatures.dynamic_streams(incoming=self.incoming_pack,

outgoing=self.outgoing_pack)
                    extractor.extract_feature(length=packet_length_feature,
measur=packet_measur_stats)

                    self.erase_length()

                    self.first_pack_time = 0
                    last_pack_time = ts
                    iat = last_pack_time - self.first_pack_time
                    extractor.extract_feature(iat=iat)
                    self.first_pack_time = last_pack_time
                else:
                    packet_length_feature                                =
DynamicFeatures.dynamic_calculation(ethsize=self.ethsize)
                    extractor.extract_feature(length=packet_length_feature)

                    last_pack_time = ts
                    iat = last_pack_time - self.first_pack_time

                    extractor.extract_feature(iat=iat)

                    self.first_pack_time = last_pack_time

                    connectivity       =       ConnectivityFeatures(packet,
dpkt_packet=dpkt_eth)
                    src = connectivity.get_source_ip()
                    dst = connectivity.get_destination_ip()

                    if src in dsts:
                        self.outgoing_pack.append(packet_frame_size)
                    else:
                        dsts[src] = 1
                        self.outgoing_pack.append(packet_frame_size)

                    if dst in srcs:
                        self.incoming_pack.append(packet_frame_size)
                    else:
                        srcs[dst] = 1
                        self.incoming_pack.append(packet_frame_size)

                    packet_measur_stats                                  =
DynamicFeatures.dynamic_streams(incoming=self.incoming_pack,

outgoing=self.outgoing_pack)
                    extractor.extract_feature(measur=packet_measur_stats)

                connectivity         =         ConnectivityFeatures(packet,
dpkt_packet=dpkt_eth)
                proto = connectivity.get_protocol_type()
                self.additional_data.proto = proto
                duration = connectivity.duration()

                extractor.extract_feature(proto=proto, duration=duration)
```

```
                    extractor.assign_layer_flags(packet)

                    src_ip = connectivity.get_source_ip()
                    dst_ip = connectivity.get_destination_ip()

                    self.additional_data.src_ip = src_ip
                    self.additional_data.dst_ip = dst_ip

                    if      connectivity.related_to_tranport(TransportLayer.udp,
packet):
                        src_port = connectivity.get_source_port()
                        dst_port = connectivity.get_destination_port()

                        self.additional_data.src_port = src_port
                        self.additional_data.dst_port = dst_port

                        flow = sorted([(src_ip, src_port), (dst_ip, dst_port)])
                        flow = (flow[0], flow[1])
                        flow_data = {
                            'byte_count': packet_frame_size,
                            'ts': ts
                        }
                        if self.udp_flows.get(flow):
                            self.udp_flows[flow].append(flow_data)
                        else:
                            self.udp_flows[flow] = [flow_data]

                        packets = self.udp_flows[flow]
                        self.number_of_packets_per_transaction = len(packets)
                        flow_byte,      self.flow_duration,      idle_time      =
FlowExtractor.get_flow_duration(self.udp_flows, flow)
                        self.src_to_dst_pkt,        self.dst_to_src_pkt        =
FlowExtractor.get_src_dst_packets(self.udp_flows, flow)


extractor.extract_feature(flow_duration=self.flow_duration,
header_length=flow_byte)

                    elif    connectivity.related_to_tranport(TransportLayer.tcp,
packet):
                        src_port = connectivity.get_source_port()
                        dst_port = connectivity.get_destination_port()

                        self.additional_data.src_port = src_port
                        self.additional_data.dst_port = dst_port

                        flag_values = connectivity.get_flag_values(dpkt_eth)

                        flow = sorted([(src_ip, src_port), (dst_ip, dst_port)])
                        flow = (flow[0], flow[1])
                        flow_data = {
                            'byte_count': packet_frame_size,
                            'ts': ts
                        }

                        if self.tcp_flows.get(flow):
                            self.tcp_flows[flow].append(flow_data)
                            # comparing Flow state based on its flags
                            ack_count, syn_count, fin_count, urg_count, rst_count
= self.tcp_flow_flags[flow]
                            ack_count, syn_count, fin_count, urg_count, rst_count
= FlowExtractor.compare_flow_flags(packet,

ack_count,
```

```
syn_count,

fin_count,

urg_count,

rst_count,

flag_values)
                            self.tcp_flow_flags[flow] = [ack_count, syn_count,
fin_count, urg_count, rst_count]
                        else:
                            self.tcp_flows[flow] = [flow_data]
                            ack_count, syn_count, fin_count, urg_count, rst_count
= FlowExtractor.compare_flow_flags(packet,

ack_count,

syn_count,

fin_count,

urg_count,

rst_count,

flag_values)
                            self.tcp_flow_flags[flow] = [ack_count, syn_count,
fin_count, urg_count, rst_count]

                        extractor.assign_tcp_flags(packet)
                        extractor.assign_flags_count(ack_count,       syn_count,
fin_count, rst_count, urg_count)

                        packets = self.tcp_flows[flow]
                        self.number_of_packets_per_transaction = len(packets)
                        flow_byte,      self.flow_duration,      idle_time      =
FlowExtractor.get_flow_duration(self.tcp_flows, flow)
                        self.src_to_dst_pkt,        self.dst_to_src_pkt        =
FlowExtractor.get_src_dst_packets(self.tcp_flows, flow)


extractor.extract_feature(flow_duration=self.flow_duration,
                                                header_length=flow_byte)

                    if self.flow_duration != 0:
                        self.rate  =  self.number_of_packets_per_transaction  /
self.flow_duration
                        self.srate = self.src_to_dst_pkt / self.flow_duration
                        self.drate = self.dst_to_src_pkt / self.flow_duration

                    extractor.extract_feature(number=len(self.ethsize),
                                              rate=self.rate,
                                              srate=self.srate,
                                              drate=self.drate)

                    packet_data = extractor.get_data()
                    callback(packet_data, self.additional_data)
```

Лістниг 1.5 – Клас core.py

```
import time
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
import pickle
import optuna
from idnn.intermediate_model import TTS
from idnn.constants import Attack, mapping

class Trainer:

    def __init__(self):
        self.model = None
        self.df = pd.DataFrame()
        self.tts: TTS
        self.scaler = StandardScaler()

        self.maximize = 'maximize'
        self.num_trials = 10
        self.study = optuna.create_study(direction=self.maximize)
        self.model_trained = False

    def read_dataset(self, name):
        print('Start dataset allocation')
        start_time = time.time()
        self.df = pd.read_csv(name)
        end_time = time.time()
        execution_time = end_time - start_time
        print(f"Proccessed CSV DataFrame time: {execution_time}")

    def mapper(self):
        print('Mapping label class values')
        self.df['label'] = self.df['label'].map(mapping)

    def preprocess(self):
        print('Preprocess dataframe. Reset indexes, drop duplicates, drop
NA')
        self.df = self.df.dropna()
        self.df = self.df.reset_index(drop=True)
        self.df = self.df.drop_duplicates()

    def features(self):
        print('Sklearn train test process')
        target_column = 'label'
        X = self.df.drop(target_column, axis=1)
        y = self.df[target_column]
        start_time = time.time()
        X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.2, random_state=12)
        self.tts = TTS(X_train, y_train, X_test, y_test)
        end_time = time.time()
        execution_time = end_time - start_time
        print(f"Proccessed TRAIN TEST SPLIT Time: {execution_time}")

    def make_scale(self):
        print('Transform values with Scaler features')
        start_time = time.time()
        self.tts.x_train = self.scaler.fit_transform(self.tts.x_train)
```

```
            self.tts.x_test = self.scaler.transform(self.tts.x_test)
            end_time = time.time()
            execution_time = end_time - start_time
            print(f"Proccessed StandardScaler Time: {execution_time}")

        def hyperparameters(self, trial):
            print('Inject hyperparameters')
            max_depth = trial.suggest_int('max_depth', 2, 32, log=False)
            max_features   =   trial.suggest_int('max_features',   2,   46,
log=False)
            n_estimators   =   trial.suggest_int('n_estimators',   3,   30,
log=False)
            classifier  =  RandomForestClassifier(max_features=max_features,
max_depth=max_depth, n_estimators=n_estimators)
            classifier.fit(self.tts.x_train, self.tts.y_train)
            accuracy = classifier.score(self.tts.x_test, self.tts.y_test)
            return accuracy

        def optunate(self):
            print('Optunate hyperparameters')
            with tqdm(total=self.num_trials) as progress_bar:
                def callback(study, trial):
                    progress_bar.update(1)

                self.study.optimize(self.hyperparameters,
n_trials=self.num_trials, callbacks=[callback])
            print(f'BEST TRIAL: {self.study.best_trial}')

        def start_train(self):
            print('Start train')
            max_features = self.study.best_trial.params['max_features']
            max_depth = self.study.best_trial.params['max_depth']
            n_estimators = self.study.best_trial.params['n_estimators']

            self.model = RandomForestClassifier(max_features=max_features,
                                                max_depth=max_depth,
                                                n_estimators=n_estimators,
                                                verbose=2)
            self.model.fit(self.tts.x_train, self.tts.y_train)

            model_train        =        self.model.score(self.tts.x_train,
self.tts.y_train)
            model_test = self.model.score(self.tts.x_test, self.tts.y_test)
            print(f'Train Score: {model_train}\nTest Score: {model_test}')
            self.model_trained = True

        def show_metrics(self):
            if self.model_trained is False:
                return
            X_test = self.scaler.transform(self.tts.x_test)
            y_pred = self.model.predict(X_test)
            print("Accuracy:",              accuracy_score(self.tts.y_test,
y_pred.round()))
            print("Precision:",            precision_score(self.tts.y_test,
y_pred.round(), average='macro'))
            print("Recall:",  recall_score(self.tts.y_test,  y_pred.round(),
average='macro'))
            print("F1  Score:",  f1_score(self.tts.y_test,  y_pred.round(),
average='macro'))

        def save(self, path):
            with open(path, 'wb') as f:
                pickle.dump(self.model, f)
```

```
def train(self, dataset_name):
    self.read_dataset(dataset_name)
    self.mapper()
    self.preprocess()
    self.features()
    self.make_scale()
    self.optunate()
    self.start_train()
    self.show_metrics()
```

ДОДАТОК Б

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Модель та методи виявлення широкомасштабної атаки в середовищі IoT

Кваліфікаційна робота
Другий (магістерський) рівень

Автор:
ст. гр. СПм-22-6
Великодний І.А.

Керівник:
доц. каф. ЕОМ
Ляшенко О.С

Актуальність

- Ростуча кількість пристроїв, ціна і важливість інформації збільшує ризик кіберзагроз і викраденню інформації в корисних цілях. Виходячи з цього, розробка інтелектуальних методів та систем виявлення вторгнень для пристроїв IoT стає необхідною для їх ефективного захисту.
- Тема безпеки інформаційного середовища стає дедалі актуальною і кібербезпека набуває життєвої важливості, з огляду на те що IoT є драйвером промислової революції та системою для збору живих даних.

Таким чином, система виявлення вторгнень є необхідною для виявлення і захисту мережі та пов’язаних систем від поточних і майбутніх кібератак.

Мета та постановка задачі

Метою даної роботи є запропонування системи виявлення вторгнень в режимі реального часу, яка буде навчена на наборі з великим обсягом даних, за допомогою нейронної мережі з використанням ансамблевого методу машинного навчання.

Задача цієї роботи складається з проведення аналізу існуючитх концепцій та методів виявлення вторгнень в інфраструктурі IoT та розроблення власної системи виявлення вторгнень і створення програмного забезпечення у вигляді CLI інтерфейсу.

3

Методи систем виявлення вторгнень

Системи виявлення вторгнень можуть використовувати різні методи, такі як сигнатурний аналіз, виявлення аномалій, використання інтелектуальних технологій, включаючи машинне навчання та евристичний аналіз.

Зазвичай системи виявлення вторгнень використовують два основні підходи для виявлення потенційних загроз:

- Сигнатурний аналіз – метод який грунтується на використанні визначених сигнатур або патернів для ідентифікаціх або розпізнавання конкретних відомих загроз. Сигнатури можуть представляти з себе конкретні приклади або вирази в шкідливому програмному коді, унікальні характеристики того чи іншого вірусу чи способи вторгнення, які раніше вже були визначені або вивчені. IDS застосовує ці сигнатури для пошуку вхідних даних чи активности в мережі, які відповідають зазначеним сигнатурам.

4

Методи систем виявлення вторгнень

- Виявлення аномалій – метод який базується на аналізі звичайної поведінки мережі, системи, користувачів чи інших об’єктів. Система методу будує модель так званої „норми“ на основі історичних даних, фокусується на виявленні незвичайностей, відхиленню від цієї норми, яке може бути ознакою нових загроз або підозр.

До підходів виявлення можемо віднести: статистичні методи, методи машиного навчання, методи порівння зразків. Виявлення аномалій може проводится на основі патернів мережевого трафіку, неправильних адрес або портів, незвичних об’ємів даних, аналіз лог файлів. Застосування цього методу допомогає виявляти атаки, які можуть бути невідомими та непередбаченими, що робить його ефективним і корисним для захисту від нових атак та загроз.

5

Машинне навчання в IDS

Машинне навчання відіграє важливо роль у покращенні ефективності та адаптивності в системах виявлення вторгнень. Воно дозволяє системам аналізувати дані, навчатися на їх основі та виявляти нові невідомі загрози, класифікувати події як безпечні чи підозрілі. Навчання моделі на основі історичних даних та поведінки допомогає автоматично розпізнати нові атаки чи загрози.

Машинне навчання ефективно працює з великими обсягами даних, що дозволяє виявляти складні патерни та взаємодії, які може бути важко виявити за допомогою традиційних методів. Застосування ML дозволяє створювати інтелектуальні IDS, яеі можуть взаємодіяти та розпізнавати атаки на високому рівні.

6

Порівняння нейромереж для визначення найбільш підхожої

NN	Train Score	Test Score	Accuracy	Precision	Recall	F1 Score
XGBClassifier	0.9999	0.9950	0.9950	0.8501	0.7648	0.7945
VotingClassifier	0.9882	0.9864	0.9864	0.8077	0.7825	0.7641
RandomForest	0.9974	0.9963	0.9963	0.9646	0.8473	0.8903
AdaBoostClassifier	0.9756	0.9756	0.9756	0.7986	0.7437	0.7273
GradientBoost	0.9999	0.9940	0.9940	0.8556	0.8509	0.8528

$$Acc = \frac{\text{кількість правильних передбачень}}{\text{Загальна кількість передбачень}}$$

$$F_1 = \frac{2}{\frac{1}{\text{recall}} \times \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision+recall}}$$

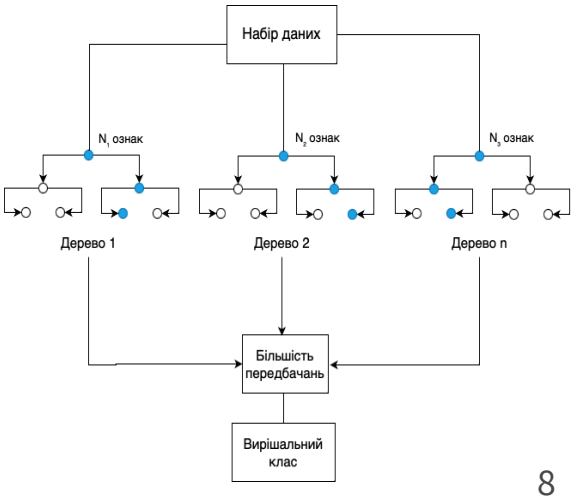
$$Pr = \frac{TP}{TP + FP}$$

7

Архітектура нейронної мережі

В ході досліджень було вирішено використовувати модель ансамблевого методу RandomForest. Він показав найкращі показники з усіх інших.

Це метод машиного навчання, який використовується для класифікації та регресії. Він є типом і відповідає ряду класифікаторів дерева рішень на різних підвибірках набору даних. Використовує техніку випадковості і усереднення для підвищення точності прогнозування, покращення продуктивності та стабільності моделі.



8

Архітектура IDS

9

Розробка програмного забезпечення

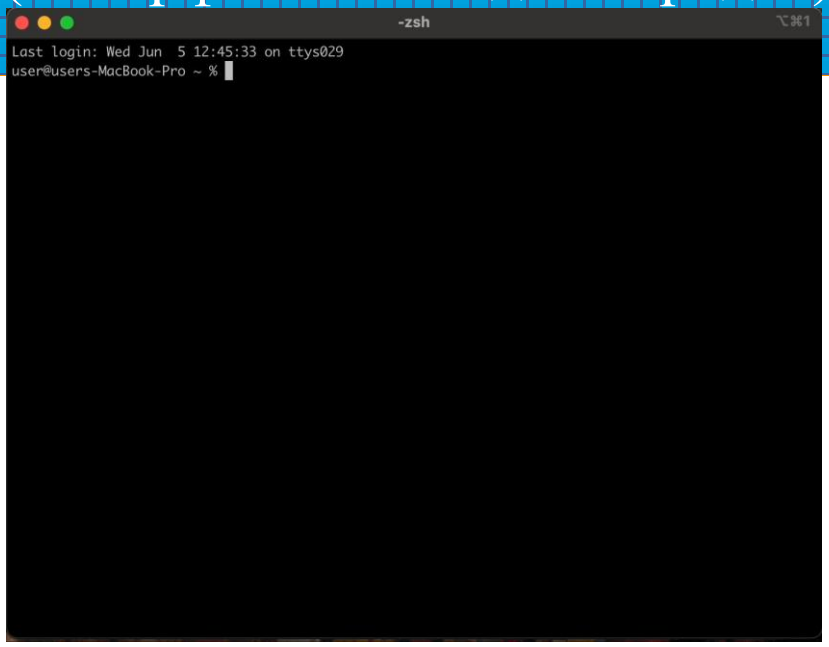







10

CLI (Інтерфейс командного рядка)



11

Висновки

У наслідок проведених досліджень і розглянутих концепцій передових систем та методів виявлення атак в інфраструктурі IoT, було запропоновано власну систему виявлення вторгнень в реальному часі, яка базується на методі виявленні аномалій та працює в комплексі з нейронної мережею ансамблевого методу *RandomForest,* яка була навчена на наборі з великим обсягом даних та має гарні показники: *правильність – 0.996, точність – 0.964, запам'ятовування – 0.847,* гармонічне середнє значення точності та *запам'ятовування – 0.89.* Для зручності використання застосунку системи був розроблений інтерфейс командного рядка, котрий сповіщає користувача або іншу систему про вторгнення чи атаку. В майбутньому запропонована модель може бути використана для систем побудованих в поєднанні з концепцією *туманного обчислення(Fog Computing)* та Інтернету речей, на принципах Fog-IoT архітектури.

12