KARADENİZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
AĞ GÜVENLİĞİ

IDS (Intrusion Detection System)

Sızma Tespit Sistemi Dönem Ödevi

(KITSUNE)

Numara	Ad Soyad
401058	Sefa Subaşı
394795	Derya ÇOBAN
394762	Talha YAY

ÖZET

‘Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection' makalesi temel alınarak ağ güvenliği alanında kullanılan İhlal Tespit Sistemleri (IDS) üzerine iyileştirmeler yapılmıştır. Kitsune, ağ trafiğini gerçek zamanlı olarak analiz edebilen ve anormallikleri tespit eden bir yapay zekâ modelidir. Yapılan iyileştirmeler, sistemin algılama hassasiyetini ve yanıt sürelerini artırmak üzerine odaklanmıştır. Bu rapor Kitsune modelinin mevcut yapılandırmasını ele almakta ve yapılan değişiklikler ve elde edilen sonuçlar detaylı bir şekilde incelenmektedir.

GİRİŞ

Son yıllarda ağ güvenliği önemli bir bilgi işlem alanı olarak öne çıkmıştır. Siber saldırıların artan sıklığı ve karmaşıklığı, güvenlik sistemlerinin sürekli olarak geliştirilmesini zorunlu kılmaktadır. Bu bağlamda, Intrusion Detection Systems (IDS) yani İhlal Tespit Sistemleri ağ trafiğindeki şüpheli aktiviteleri tespit etme ve bunlara müdahale etme kapasitesiyle kritik bir rol oynamaktadır. Ancak, mevcut sistemlerin sınırlamaları ve eksiklikleri bu sistemlerin daha da iyileştirilmesi gerektiğini göstermektedir. Bu projede IDS teknolojilerini geliştirmek için mevcut bir çalışma temel alınarak, 'Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection' isimli çalışma üzerinde iyileştirmeler yapılmıştır. Kitsune, ağ trafiği üzerinde gerçek zamanlı olarak anormallik tespiti yapabilen otoenkoder tabanlı bir sistemdir. Bu raporda, bahsi geçen çalışmanın detayları açıklanacak ve yapılan iyileştirmeler ile elde edilen sonuçlar sunulacaktır.

Kitsune nedir?

Kitsune, Ben-Gurion Üniversitesi'nden Yisroel Mirsky ve ekibi tarafından geliştirilen çevrimiçi ağ saldırılarını tespit edebilen bir ağ saldırı tespit sistemidir. Kitsune'nin temel algoritması olan KitNET; autoencoder adı verilen sinir ağlarından oluşan bir ansambl kullanarak normal ve anormal trafik desenlerini ayırt eder. Ağ güvenliği alanında yenilikçi ve etkili bir çözüm sunarak, ağ trafiğini sürekli olarak izleyebilir ve anormallikleri tespit ederek olası saldırılara karşı koruma sağlar. Bu özellikleri sayesinde farklı ağ ortamlarında kullanılabilecek esnek ve güçlü bir NIDS çözümüdür.

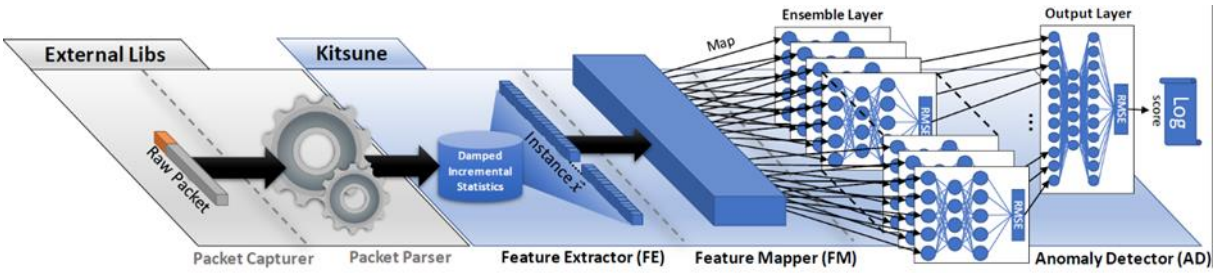
Kitsune'nin Temel Özellikleri:

+ **Otoenkoder Ansambli**: Kitsune çok sayıda otoenkoderin bir araya gelmesiyle oluşturulan bir ansambl yapı kullanır. Her bir otoenkoder ağ trafiğinin normal davranış modellerini öğrenir ve böylece anormallikler daha etkili bir şekilde tespit edilebilir.
+ **Hafif ve Hızlı**: Algoritma düşük işlem gücü gerektiren bir yapıya sahiptir ve bu sayede gerçek zamanlı tespitlerde yüksek performans gösterir. Bu özellik özellikle büyük ve hızlı veri akışlarının olduğu ortamlarda avantaj sağlar.
+ **Otomatik Özellik Öğrenme**: Kitsune ağ trafiği verilerinden otomatik olarak özellikler çıkarabilir bu da manuel özellik mühendisliğine olan ihtiyacı azaltır ve modelin çeşitli ağ ortamlarına kolayca uyarlanmasını sağlar.
+ **Geniş Uygulama Alanı**: IoT cihazlarından kurumsal ağlara kadar geniş bir kullanım yelpazesine sahiptir.
+ **Anormallik Tespiti**: Kitsune ağ trafiğindeki beklenmeyen davranışları saptayarak güvenlik ihlallerinin erken uyarılarını sağlar.

Kitsune Tarafından Tespit Edilebilen Saldırı Türleri

+ **DDoS (Dağıtık Hizmet Reddi) Saldırıları**: Büyük miktarda zararlı trafik ile hedeflenen bir sistemi veya ağı felç etme amacı taşıyan saldırılar.
+ **Malware Aktiviteleri**: Zararlı yazılımların ağa sızma veya ağ üzerinde hareket etme aktiviteleri.
+ **İçeriden Yapılan Saldırılar**: Meşru kullanıcı hesaplarını taklit ederek veya kötüye kullanarak yapılan saldırılar.
+ **Tarama Saldırıları**: Ağdaki zayıf noktaları veya açıkları bulmak için yapılan sistematik taramalar.

Kitsune Mimarisi



- **External Libs**: Kitsune ağ paketlerini yakalamak ve işlemek için dış kütüphanelere (libs) bağımlıdır. Bu kütüphaneler paket yakalama ve analiz etme işlevlerini sağlar.
- **Packet Capturer**: Ağ üzerinden geçen paketleri yakalar. Bu modül ağ trafiğini dinler ve ileri işlem için gerekli olan ham paket verilerini toplar.
- **Packet Parser**: Yakalanan paketlerden gerekli bilgileri çıkarır. Bu aşama paketlerin içeriğini anlamak ve kullanılabilir verilere dönüştürmek için kritik öneme sahiptir.
- **Feature Extractor (FE)**: Paket parser tarafından işlenen verilerden özellikler çıkarır. Özellik çıkarma verilerin model tarafından işlenmesi için uygun hale getirilmesini sağlar.
- **Feature Mapper (FM)**: Çıkarılan özellikleri otoenkoderler tarafından işlenebilecek formata dönüştürür. Bu modül özelliklerin uygun bir şekilde haritalanmasını ve gruplandırılmasını sağlar.
- **Ensemble Layer**: Çoklu otoenkoderlerin yer aldığı katmandır. Her bir otoenkoder belirli özellik grupları üzerinde çalışarak anomali tespiti yapar. Bu katman çeşitli otoenkoderlerin bir araya gelmesiyle oluşur ve her biri farklı türden veri anomalilerini tespit etmek için özelleştirilmiştir.
- **Output Layer**: Ensemble layer'dan gelen bilgileri alır ve son anomali tespit kararını verir. Bu katman tüm otoenkoderlerden gelen çıktıları birleştirerek nihai bir skor veya alarm üretir.
- **ILog**: Anomali tespit sonuçlarını kaydeder. Bu loglar tespit edilen olayların incelenmesi, raporlanması ve arşivlenmesi için kullanılır.

Kitsune'nin Avantajları

- **Gerçek Zamanlı Tespit**: Kitsune ağ trafiğini gerçek zamanlı olarak izleyerek anormallikleri hızlı bir şekilde tespit edebilir. Bu özellik potansiyel tehditlere karşı hızlı müdahale imkanı sunar.
- **Düşük Hesaplama Gereksinimi**: Sistem düşük işlem gücü ile çalışacak şekilde tasarlanmıştır bu da onu kaynak kısıtlı ortamlarda bile uygun hale getirir.

+ **Otomatik Özellik Öğrenme**: Kitsune manuel özellik mühendisliği gerektirmeden ağ trafiğinden otomatik olarak özellikler çıkarabilir. Bu modelin farklı ağ yapılarına kolayca uyum sağlamasını sağlar.
+ **Yüksek Doğruluk ve Düşük Yanlış Alarm Oranı**: Çoklu otoenkoder ansamblini kullanarak Kitsune yüksek tespit doğruluğu ve düşük yanlış alarm oranı sunar.
+ **Esneklik ve Geniş Uygulama Alanı**: IoT cihazlarından kurumsal ağ yapılarına kadar geniş bir uygulama yelpazesine sahiptir ve bu da farklı tipteki ağlar için etkili bir çözüm sunar.
+ **Kolay Entegrasyon ve Ölçeklenebilirlik**: Kitsune mevcut ağ altyapılarına kolayca entegre edilebilir ve ihtiyaçlara göre ölçeklendirilebilir.
+ **Kapsamlı Anomali Tespiti**: DDoS saldırıları malware aktiviteleri ve içeriden yapılan saldırılar gibi çeşitli tehdit türlerini tespit edebilme kapasitesine sahiptir.
+ **Açık Kaynak Avantajı**: Kitsune geliştiricilerin ve araştırmacıların algoritmayı özelleştirebilmesi ve geliştirebilmesi için açık kaynak kodlu bir yapıdadır.

Yapılan Çalışma

Kitsune'nin ağ trafiğini kontrol eden NIDS tabanlı bir sistemdir. Bu sistemin amacı ağ trafiğindeki zararlı atakları (DOS, DDOS) izleyebilme ve buna bağlı RMSE değerleri döndüren anomali tespit sistemidir. Sistemi örnek kodla çalıştırdığımız zaman çok yavaş, hantal olduğunu fark ettik. Bunu üzerine neler yapabileceğimiz araştırdık. Hızlandırma için paralel işleme yöntemlerine başvurduk. Burada grafik işlemcide paralel koşarak daha hızlı yapabileceğimizi öğrendik. Bunu ışığında CUDA'yı(cupy) kütüphanesini kullandık. Daha sonra daha hızlı işlediğini fark ettik. Bunu üzerine bunu nasıl kendi Wifi ağ trafiğimi izleyebilirim sorusu aklımıza geldi. Bunu ışığında araştırmalara başladık. WireShark ile trafiği izleyebileceğimiz bu metrikleri programımızda işleyebileceğimizi öğrendik. Daha sonra bunu için Kitsune.py'de bunu için değişiklikler yaptık. Canlı trafiği izlemek için PyQt5'i kullanabileceğimizi bulduk ve programımıza ekledik. Aşağıda kodlarla alakalı detay bilgi verilmiştir.

(Kitsune.py) adı verilen dosyada gerçek zamanlı bir RMSE (Root Mean Square Error) grafiği çizen bir uygulamayı başlatır. Bu sistem, ağ trafiğini izleyerek her paketten anormallik skorlarını hesaplar. `FeatureExtractor` ve `KitNET` modülleri kullanılarak paketlerden özellikler çıkarılır ve anormallikler saptanır. Kod, PyQt5 ve pyqtgraph kütüphanelerini kullanarak kullanıcı arayüzü oluşturur ve verileri görselleştirir. Uygulama, `QTimer` kullanarak belirli aralıklarla `update` fonksiyonunu çağırır; bu fonksiyon yeni paketler işlendiğinde ve belirli koşullar altında (örneğin, her 1000 pakette bir ve paket sayısı 54000'den fazla olduğunda) grafiği günceller. Kod, 75,000 paket işledikten sonra durur veya bir hata ile karşılaşılırsa işlemi sonlandırır. Bu, ağ güvenliği ve performans izleme uygulamaları için pratik ve etkili bir araç sunar.

```python
import cupy as np
import time
from FeatureExtractor import FE
from KitNET.KitNET import KitNET
import pyqtgraph as pg
from PyQt5.QtWidgets import QApplication
from PyQt5.QtCore import QTimer

class Kitsune:
    def __init__(self, interface, limit, max_autoencoder_size=10, FM_grace_period=None, AD_grace_period=10000, learning_rate=0.1, hidden_ratio=0.75):
        self.FE = FE(interface, limit)
        self.AnomDetector = KitNET(self.FE.get_num_features(), max_autoencoder_size, FM_grace_period, AD_grace_period, learning_rate, hidden_ratio)

    def proc_next_packet(self):
        x = self.FE.get_next_vector()
        if len(x) == 0:
            return -1  # Error or no packets left
        return self.AnomDetector.process(x)  # Train during grace periods, then execute

if __name__ == "__main__":
    interface = "Wi-Fi"  # For Windows usually "Wi-Fi" or "Ethernet"
    packet_limit = np.Inf

    maxAE = 10
    FMgrace = 5000
    ADgrace = 50000

    K = Kitsune(interface, packet_limit, maxAE, FMgrace, ADgrace)

    print("Running Kitsune:")
    RMSEs = []
    i = 0
    start_time = time.time()
    max_duration = 5 * 60  # 5 minutes

    app = QApplication([])  # Correct use of QApplication
    win = pg.GraphicsLayoutWidget(title="Anomaly Scores from Kitsune's Execution Phase")  # Updated widget usage
    plot = win.addPlot(title="Real-Time RMSE Plot")
    curve = plot.plot(pen='y')

    timer = QTimer()
```

```python
    def update():
        global RMSEs, curve, plot, i, start_time

        if i >= 75000:  # Stop after processing 75,000 packets
            timer.stop()
            app.quit()
            duration = time.time() - start_time
            print(f"Complete. Processed {i} packets. Time elapsed: {duration:.2f} seconds")
            return
        rmse = K.proc_next_packet()
        if rmse == -1:
            print("No more packets or error in processing.")
            timer.stop()
            app.quit()
            duration = time.time() - start_time
            print(f"Error encountered after {i} packets. Time elapsed: {duration:.2f} seconds")
            return
        RMSEs.append(rmse)
        if i % 1000 == 0 and i > 54000:  # Update the graph every 1000 packets
            curve.setData(RMSEs)
        i += 1

    timer.timeout.connect(update)
    timer.start(0)  # Immediately start the timer without delay

    win.show()  # Show window
    app.exec_()  # Start application
```

Özellik Çıkarıcı (Feature Extractor - FE): Bu bileşen, ham ağ trafiği verilerinden özellikler çıkarmak için kullanılır. FE sınıfı, belirtilen bir dosya yolu (`file_path`) üzerinden ağ paketlerini okur ve her bir paket için bir özellik vektörü oluşturur. Bu

vektörler, ağ trafiğinin karakteristiklerini temsil eder ve anormallik tespiti için gerekli bilgileri sağlar.

```python
import os
import subprocess
import csv
import cupy as np
from scapy.all import *
import platform
import netStat as ns  # netStat modülünü ekliyoruz

class FE:
    def __init__(self, interface="wlan0", limit=np.inf):
        self.interface = interface
        self.limit = limit
        self.curPacketIndx = 0
        self.process = None
        self.tsvin = None
        self._start_tshark()

        # netStat nesnesini tanımlıyoruz
        self.nstat = ns.netStat(HostLimit=10000, HostSimplexLimit=1000)

    def _start_tshark(self):
        tshark_path = self._get_tshark_path()
        if not tshark_path:
            raise FileNotFoundError("tshark executable not found. Please ensure Wireshark is installed.")
        cmd = [
            tshark_path,
            "-i", self.interface,
            "-T", "fields",
            "-e", "frame.time_epoch",
            "-e", "frame.len",
            "-e", "eth.src",
            "-e", "eth.dst",
            "-e", "ip.src",
            "-e", "ip.dst",
            "-e", "tcp.srcport",
            "-e", "tcp.dstport",
            "-e", "udp.srcport",
            "-e", "udp.dstport",
            "-e", "icmp.type",
            "-e", "icmp.code",
            "-e", "arp.opcode",
            "-e", "arp.src.hw_mac",
```

Anormallik Tespit Modülü (KitNET): Bu bileşen, `KitNET` sınıfı tarafından gerçekleştirilir. `KitNET`, özellik vektörlerini kullanarak eğitim sürecini yönetir ve anormallik tespiti yapar. Anormallik tespiti, otoenkoder tabanlı bir model kullanılarak gerçekleştirilir. Bu model, başlangıçta belirlenen "FM_grace_period" ve "AD_grace_period" süreleri boyunca eğitilir. Bu süreler, modelin yeterli miktarda veri ile eğitilmesini sağlamak için kullanılır. Eğitim tamamlandıktan sonra, model gelen paketler üzerinde anormallik tespiti yapmaya başlar.

Ağ trafiğindeki potansiyel güvenlik tehditlerini veya anormal davranışları otomatik olarak tespit etmek amacıyla kullanılan gelişmiş bir ağ güvenlik çözümüdür. İyi bir ağ güvenliği uygulaması olarak, gerçek zamanlı veri analizi ve anormallik tespiti sağlar, böylece ağ üzerindeki şüpheli faaliyetler hızlı bir şekilde belirlenip müdahale edilebilir.

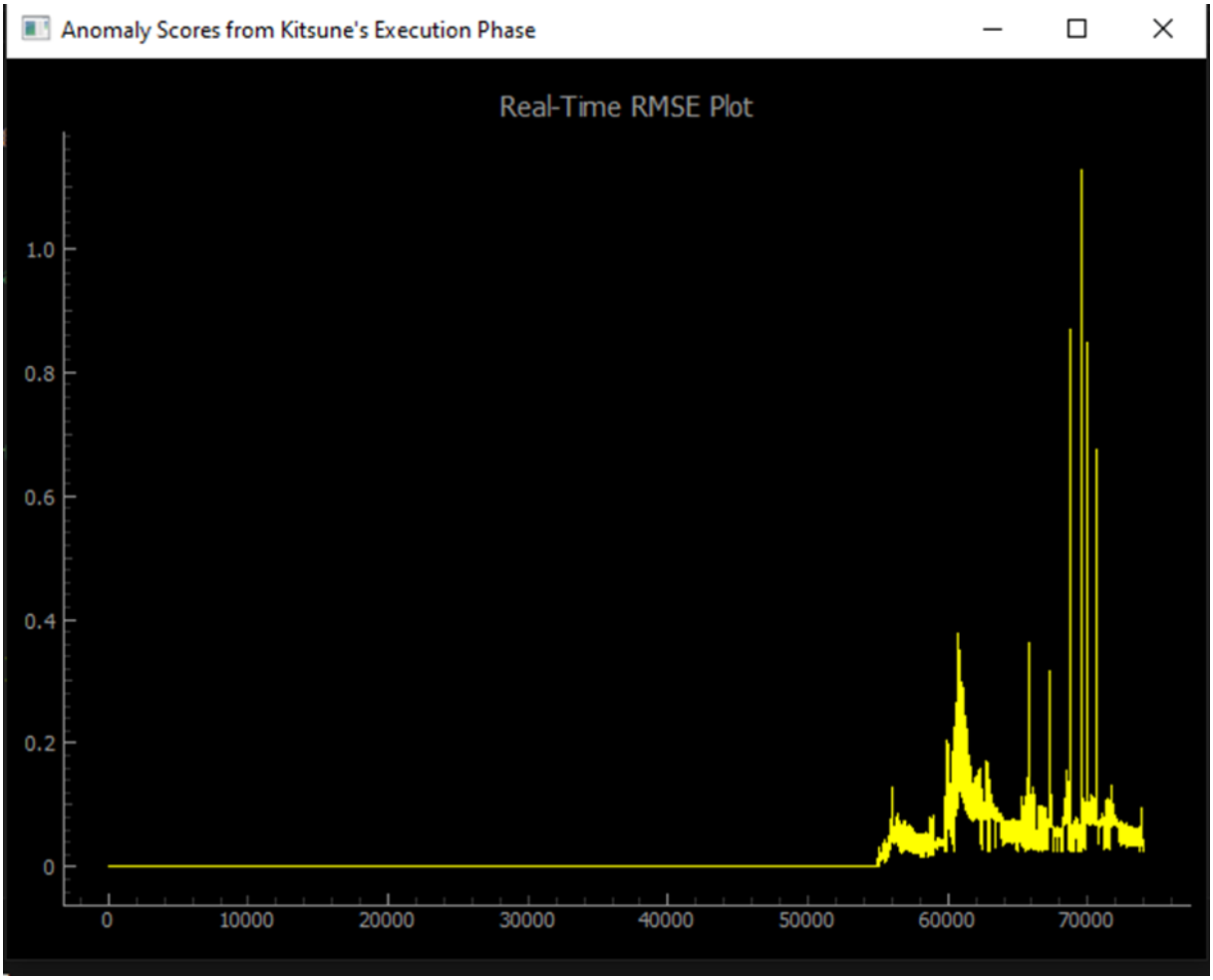
Kitsune.py M | FeatureExtractor.py M | KitNET.py | corClust.py M

Kitsune-py > KitNET > KitNET.py

```python
import numpy as np
import KitNET.dA as AE
import KitNET.corClust as CC

# This class represents a KitNET machine learner.
# KitNET is a lightweight online anomaly detection algorithm based on an ensemble of autoencoders.
# For more information and citation, please see our NDSS'18 paper: Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection
# For licensing information, see the end of this document

class KitNET:
    #n: the number of features in your input dataset (i.e., x \in R^n)
    #m: the maximum size of any autoencoder in the ensemble layer
    #AD_grace_period: the number of instances the network will learn from before producing anomaly scores
    #FM_grace_period: the number of instances which will be taken to learn the feature mapping. If 'None', then FM_grace_period=AM_grace_period
    #learning_rate: the default stochastic gradient descent learning rate for all autoencoders in the KitNET instance.
    #hidden_ratio: the default ratio of hidden to visible neurons. E.g., 0.75 will cause roughly a 25% compression in the hidden layer.
    #feature_map: One may optionally provide a feature map instead of learning one. The map must be a list,
    #           where the i-th entry contains a list of the feature indices to be assingned to the i-th autoencoder in the ensemble.
    #           For example, [[2,5,3],[4,0,1],[6,7]]
    def __init__(self,n,max_autoencoder_size=10,FM_grace_period=None,AD_grace_period=10000,learning_rate=0.1,hidden_ratio=0.75, feature_map = None):
        # Parameters:
        self.AD_grace_period = AD_grace_period
        if FM_grace_period is None:
            self.FM_grace_period = AD_grace_period
        else:
            self.FM_grace_period = FM_grace_period
        if max_autoencoder_size <= 0:
            self.m = 1
        else:
            self.m = max_autoencoder_size
        self.lr = learning_rate
        self.hr = hidden_ratio
        self.n = n

        # Variables
        self.n_trained = 0 # the number of training instances so far
        self.n_executed = 0 # the number of executed instances so far
        self.v = feature_map
        if self.v is None:
            print("Feature-Mapper: train-mode, Anomaly-Detector: off-mode")
        else:
            self.__createAD__()
            print("Feature-Mapper: execute-mode, Anomaly-Detector: train-mode")
```

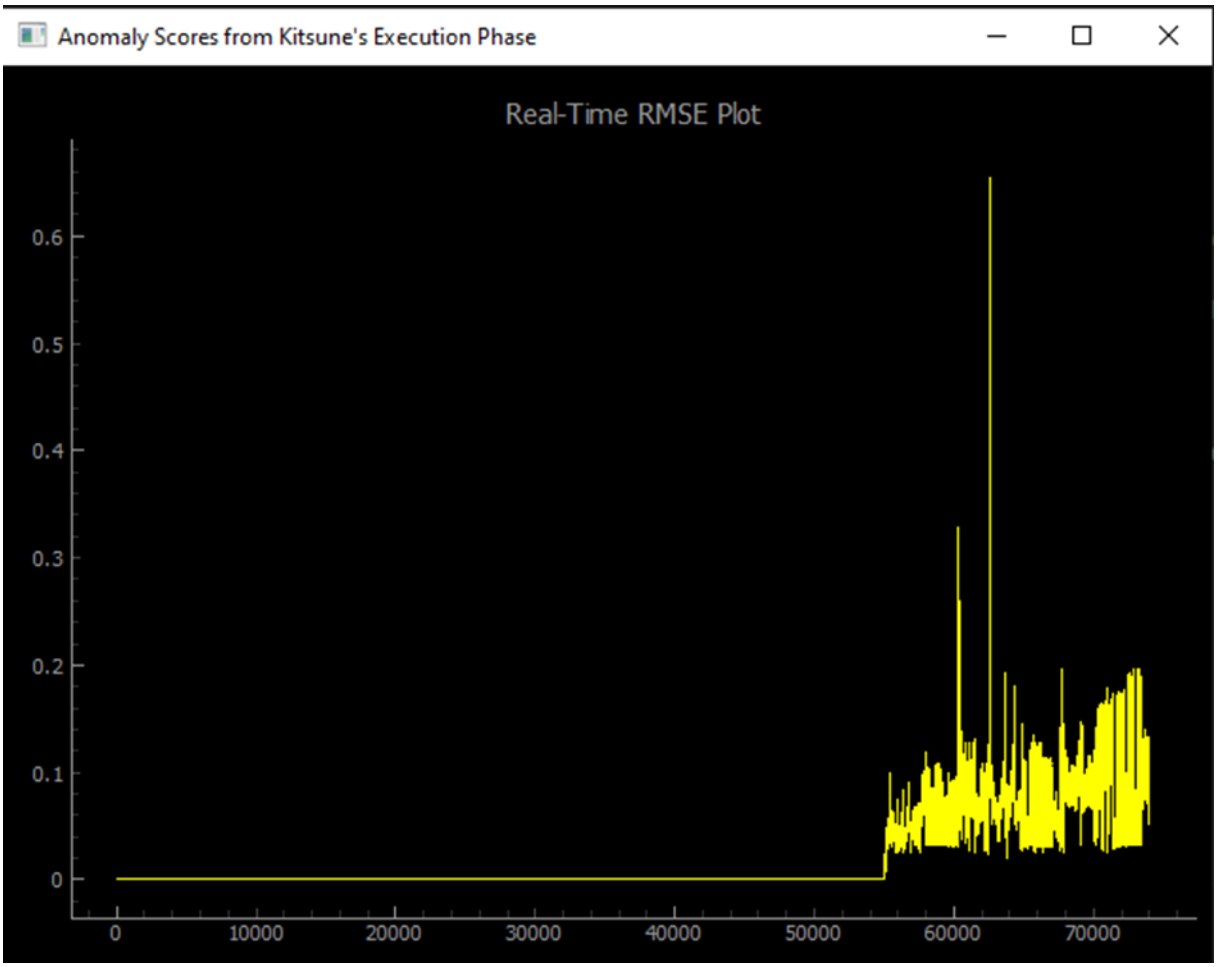
Çıktılar

Numpy.py




```
PS C:\Users\SEFA\Desktop\kitsune\Kitsune-py> python Kitsune.py
Capturing on 'Wi-Fi'
4 Feature-Mapper: train-mode, Anomaly-Detector: off-mode
Running Kitsune:
4997 The Feature-Mapper found a mapping: 100 features to 15 autoencoders.
Feature-Mapper: execute-mode, Anomaly-Detector: train-mode
55048 Feature-Mapper: execute-mode, Anomaly-Detector: execute-mode
74752 Complete. Processed 75000 packets. Time elapsed: 222.28 seconds
PS C:\Users\SEFA\Desktop\kitsune\Kitsune-py>
```

Cupy.py




```
PS C:\Users\SEFA\Desktop\kitsune\Kitsune-py> python Kitsune.py
Capturing on 'Wi-Fi'
Feature-Mapper: train-mode, Anomaly-Detector: off-mode
Running Kitsune:
4932 The Feature-Mapper found a mapping: 100 features to 12 autoencoders.
Feature-Mapper: execute-mode, Anomaly-Detector: train-mode
54850 Feature-Mapper: execute-mode, Anomaly-Detector: execute-mode
74977 Complete. Processed 75000 packets. Time elapsed: 131.48 seconds
PS C:\Users\SEFA\Desktop\kitsune\Kitsune-py>
```

KAYNAKÇA

https://github.com/ymirsky/KitNET-py (Kitsune'nin temel algoritmalarını ve kullanım örneklerini içerir.)

https://paperswithcode.com/paper/kitsune-an-ensemble-of-autoencoders-for (KitNET'in detaylı bir açıklamasını ve nasıl çalıştığını içerir. Autoencoder ansamblı kullanarak ağ trafiği desenlerini öğrenme ve anormallikleri tespit etme sürecini ele alır.)

https://arxiv.org/abs/1802.09089 (Kitsune'nin temel prensiplerini ve işleyişini açıklar.)

https://github.com/barnardp/Intrusion_Detection_XAI

https://github.com/Western-OC2-Lab/Intrusion-Detection-System-Using-Machine-Learning/blob/main/LCCDE_IDS_GlobeCom22.ipynb

https://cybersecurity.springeropen.com/articles/10.1186/s42400-019-0038-7

https://github.com/DamonMohammadbagher/Meterpreter_Payload_Detection

https://github.com/wolfSSL/wolfsentry

https://github.com/leo-arch/sids

https://ar5iv.labs.arxiv.org/html/1805.06070

https://github.com/ymirsky/Kitsune-py?tab=readme-ov-file

https://github.com/Nav3h/NetFortress

https://github.com/ossec/ossec-hids

https://github.com/rahulvigneswaran/Intrusion-Detection-Systems

https://github.com/AnikaitOO7/Display_Intrusion_Detection_System

https://github.com/rezacsedu/Intrusion-Detection-Spark-Conv-LSTM

https://github.com/Asbatel/ContainerHIDS