Experiment 2: Signature-Based IDS

```
from scapy.all import sniff, IP, TCP, ICMP

def simple_ids(packet):
    if packet.haslayer(IP):
        src = packet[IP].src
        dst = packet[IP].dst
        print(f"Packet: {src} -> {dst}")

        if packet.haslayer(TCP):
            if dst == "192.168.2.2" or src == "192.168.2.2":
                print("** Alert: TCP Packet matched signature **")
        elif packet.haslayer(ICMP):
            if dst == "192.168.2.2" or src == "192.168.2.2":
                print("** Alert: ICMP Packet matched signature **")

print("Sniffing started...")
sniff(prn=simple_ids, store=0, filter="ip")
```

Experiment 3: Anomaly-Based IDS with ML

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

data = pd.read_csv("https://raw.githubusercontent.com/defcom17/NSL_KDD/master/KDDTrain+.txt", header=None)
data = data.iloc[:, :10].dropna()
data = data.apply(LabelEncoder().fit_transform)

X = data.iloc[:, :-1]
y = data.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
print("Model accuracy:", clf.score(X_test, y_test))
```

Experiment 4: IoT Traffic Detection with Scapy

```
from scapy.all import IP, TCP, UDP, Raw, send, RandShort

mqtt = IP(dst="192.168.1.10")/TCP(dport=1883, sport=RandShort())/Raw(load="MQTT")
send(mqtt, count=3)

coap = IP(dst="192.168.1.10")/UDP(dport=5683, sport=RandShort())/Raw(load="GET /")
send(coap, count=3)
```

Experiment 5: Cloud IDS Flask API

```
from flask import Flask, jsonify
```

```
app = Flask(__name__)

@app.route("/")
def index():
    return "Suricata IDS Running..."

@app.route("/alerts")
def alerts():
    with open("/var/log/suricata/fast.log") as f:
        return jsonify(f.readlines())

if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

Experiment 6: IDS + SIEM Integration

```
import logging, time
from logging.handlers import SysLogHandler

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
handler = SysLogHandler(address=('192.168.15.3', 514))
formatter = logging.Formatter('%(asctime)s IDS: %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)

alert_id = 1
while True:
    logger.info(f"Simulated Attack Alert ID: {alert_id}")
    print(f"Alert {alert_id} sent to SIEM")
    alert_id += 1
    time.sleep(5)
```

Experiment 7: Detect ICMP Flood

```
# Add this rule in iot.rules
# alert icmp any any -> any any (msg:"ICMP Flood Detected"; sid:1000002; rev:1;)

# Simulate ICMP Flood
# Run in terminal: hping3 --flood -1 <Target-IP>
```

Experiment 8: Hybrid IDS (Snort + ML)

```
# Signature-based detection rule for Snort
# alert tcp any any -> any 80 (msg:"Possible HTTP Attack"; content:"GET /login"; sid:1000001;)

# Anomaly detection in Python
import pandas as pd
from sklearn.ensemble import IsolationForest

data = pd.read_csv("network_traffic.csv")
features = data[['src_ip', 'dst_ip', 'packet_size', 'duration']]
model = IsolationForest(contamination=0.05)
model.fit(features)
data['anomaly'] = model.predict(features)
```

```
print(data[data['anomaly'] == -1])
```

Experiment 9: Detect APTs

```
# Snort Rule
# alert tcp any any -> any any (msg:"Possible APT activity detected"; content:"malicious.exe"; sid:1000001;
rev:1;)

# ML Part
import pandas as pd
from sklearn.ensemble import IsolationForest

df = pd.read_csv("network_logs.csv")
features = df[['packet_size', 'connection_duration']]
clf = IsolationForest(contamination=0.05)
clf.fit(features)
df['anomaly'] = clf.predict(features)
print(df[df['anomaly'] == -1])
```

Experiment 10: Optimize IDS with Random Forest

```
import numpy as np, pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

np.random.seed(42)
data = pd.DataFrame({
    'packet_size': np.random.randint(50, 1500, 5000),
    'delay': np.random.uniform(0, 2, 5000),
    'flag': np.random.choice([0, 1], 5000, p=[0.8, 0.2])
})

X_train, X_test, y_train, y_test = train_test_split(data.drop(columns='flag'), data['flag'], test_size=0.3)
model = RandomForestClassifier(n_estimators=200)
model.fit(X_train, y_train)
print(classification_report(y_test, model.predict(X_test)))
```

Experiment 11: Adversarial Attack on IDS

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf

X, y = make_classification(n_samples=1000, n_features=20, n_classes=2)
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y)

model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5)

# FGSM Attack
def fgsm(model, x, eps=0.1):
    x_var = tf.Variable(x)
    with tf.GradientTape() as tape:
        tape.watch(x_var)
        loss = tf.keras.losses.binary_crossentropy(y_test, model(x_var))
    grad = tape.gradient(loss, x_var)
    return (x + eps * tf.sign(grad)).numpy()

X_adv = fgsm(model, X_test)
print("Accuracy on adversarial:", model.evaluate(X_adv, y_test)[1])
```

Experiment 12: Self-Healing IDS

```
import pandas as pd, numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

def generate_data():
    f1 = np.random.normal(50, 10, 500)
    f2 = np.random.normal(5, 2, 500)
    f1a = np.random.normal(150, 50, 500)
    f2a = np.random.normal(20, 5, 500)
    X = np.vstack((np.column_stack((f1, f2)), np.column_stack((f1a, f2a))))
    y = np.array([0]*500 + [1]*500)
    return X, y

X, y = generate_data()
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y)

model = RandomForestClassifier()
model.fit(X_train, y_train)
acc = accuracy_score(y_test, model.predict(X_test))
print("Initial Accuracy:", acc)

if acc < 0.95:
    print("Retraining model...")
    model.fit(X_train, y_train)
```