Complete Study Guide - Part 2: Code Explained (Layman Terms)

Understanding Each File

File 1: `threat_fusion_engine.py` (The Brain)

What it does: Combines network risk and user risk to make final decision

The Code:

```python
def combine_risks(network_risk, user_risk):
    final_risk = (0.6 * network_risk) + (0.4 * user_risk)

    if final_risk > 0.8:
        level = "CRITICAL"
    elif final_risk > 0.6:
        level = "HIGH"
    elif final_risk > 0.4:
        level = "MEDIUM"
    else:
        level = "LOW"

    return final_risk, level
```

In Simple Terms: - **Line 2:** Multiply network risk by 0.6 (60%), user risk by 0.4 (40%), add them - **Lines 4-11:** Check the final number and assign a label - **Line 13:** Send back both the number and the label

Example:

```
Input: network_risk = 0.95, user_risk = 0.10
Calculation: (0.6 × 0.95) + (0.4 × 0.10) = 0.57 + 0.04 = 0.61
Check: 0.61 > 0.6? Yes! So level = "HIGH"
Output: final_risk = 0.61, level = "HIGH"
```

Why This Matters: This is YOUR NOVEL CONTRIBUTION! The 60/40 weighting is unique to your research.

File 2: `ids_engine.py` (Network Watcher)

What it does: Watches network traffic and calculates network risk

Key Parts Explained:

Part A: Getting Metrics from AWS

```python
def get_metric(self, metric_name):
    end_time = datetime.utcnow()
    start_time = end_time - timedelta(minutes=5)

    response = self.cloudwatch.get_metric_statistics(
        Namespace='AWS/EC2',
        MetricName=metric_name,
        Dimensions=[{'Name': 'InstanceId', 'Value': INSTANCE_ID}],
        StartTime=start_time,
        EndTime=end_time,
        Period=60,
        Statistics=['Sum']
    )
```

In Simple Terms: - **Line 2-3:** Get current time and 5 minutes ago - **Line 5-13:** Ask AWS CloudWatch: "Give me the network data for the last 5 minutes" - **Namespace='AWS/EC2':** We're asking about EC2 (the server) - **MetricName:** Either "NetworkIn" (bytes) or "NetworkPacketsIn" (packets) - **Period=60:** Give me data in 60-second chunks

Analogy: Like asking your fitness tracker: "How many steps did I take in the last 5 minutes?"

Part B: Calculating Risk

```
def detect(self):
    network_in = self.get_metric("NetworkIn")
    packets_in = self.get_metric("NetworkPacketsIn")

    if network_in > 8_000_000 or packets_in > 15_000:
        risk = 0.95  # CRITICAL
    elif network_in > 4_000_000 or packets_in > 8_000:
        risk = 0.85  # HIGH
    elif network_in > 1_500_000 or packets_in > 3_000:
        risk = 0.60  # MEDIUM
    else:
        risk = 0.05  # LOW

    return [{"ip": "EC2_INSTANCE", "network_risk": risk}]
```

In Simple Terms: - **Line 2-3:** Get the current network traffic numbers - **Line 5-12:** Check if traffic is above certain thresholds - If VERY high (>8M bytes OR >15K packets) → 95% risk - If high (>4M bytes OR >8K packets) → 85% risk - If medium (>1.5M bytes OR >3K packets) → 60% risk - Otherwise → 5% risk (normal) - **Line 14:** Return the risk score

The Thresholds Explained:

```
Normal traffic: ~1,000 bytes, ~10 packets → 5% risk
Medium spike: 1.5M bytes, 3K packets → 60% risk
```

```
High spike: 4M bytes, 8K packets → 85% risk
Critical spike: 8M+ bytes, 15K+ packets → 95% risk
```

Why OR not AND? We use OR because EITHER high bytes OR high packets indicates an attack. You don't need both.

File 3: `ueba_engine.py` (Behavior Watcher)

What it does: Analyzes user behavior from CloudTrail logs

Key Parts Explained:

Part A: Fetching Logs

```python
def fetch_logs(self):
    logs = []
    today = datetime.datetime.utcnow()
    prefix = f"AWSLogs/{ACCOUNT_ID}/CloudTrail/{REGION}/{today.year}/{today.month:02d}/{toda

    response = self.s3.list_objects_v2(
        Bucket=BUCKET,
        Prefix=prefix,
        MaxKeys=5
    )
```

In Simple Terms: - **Line 3-4:** Get today's date and build a path to today's logs - **Line 6-10:** Ask AWS S3: "Give me the 5 most recent log files from today" - **MaxKeys=5:** Only get 5 files (for speed - we don't need all logs)

Why only today? CloudTrail logs are huge. Getting all logs would take minutes. We only need recent activity for real-time detection.

Analogy: Like checking today's security camera footage instead of watching the entire archive.

Part B: Feature Engineering

```python
def engineer_features(self, df):
    df["time"] = pd.to_datetime(df["time"])
    df["hour"] = df["time"].dt.hour
    df["day"] = df["time"].dt.dayofweek

    df["activity_volume"] = df.groupby("user")["event"].transform("count")
    df["service_diversity"] = df.groupby("user")["service"].transform("nunique")

    features = df[["hour", "day", "activity_volume", "service_diversity"]]
    df["anomaly_score"] = self.model.decision_function(features)
```

In Simple Terms: - **Line 2-4:** Extract time information (what hour? what day?) - **Line 6:** Count how many actions each user did - **Line 7:** Count how many different AWS services each user accessed - **Line 9:** Put these 4 features together - **Line 10:** Use machine learning model to calculate anomaly score

The 4 Features: 1. **Hour:** Is it 3 AM (suspicious) or 2 PM (normal)? 2. **Day:** Is it weekend (unusual) or weekday (normal)? 3. **Activity Volume:** Did user do 5 actions (normal) or 500 (suspicious)? 4. **Service Diversity:** Did user access 2 services (normal) or 20 (suspicious)?

Machine Learning Part: The Isolation Forest model was trained to recognize normal patterns. If someone's behavior is different from normal, it gives a high anomaly score.

Analogy: Like a teacher who knows how students normally behave. If a quiet student suddenly becomes very active, the teacher notices.

File 4: `enhanced_main.py` (The Orchestrator)

What it does: Runs everything together in a loop

The Main Loop:

```python
def run_detection_cycle(self):
    print("===== Hybrid Threat Detection Cycle =====")

    # Step 1: Run IDS
    print("Running IDS...")
    network_results = self.ids.detect()
    print("IDS Done")

    # Step 2: Run UEBA
    print("Running UEBA...")
    user_results = self.ueba.detect()
    print("UEBA Done")

    # Step 3: Process results
    for net in network_results:
        ip = net["ip"]
        network_risk = net["network_risk"]

        # Find matching user
        matched_user = next(
            (u for u in user_results if u["ip"] == ip),
            None
        )
        user_risk = matched_user["user_risk"] if matched_user else 0.1
        # Step 4: Combine risks
        final_risk, level = combine_risks(network_risk, user_risk)

        # Step 5: Create alert if needed
        if final_risk > 0.3:
            self.alert_system.create_alert(...)
```

In Simple Terms: 1. **Lines 5-7:** Ask IDS: "What's the network risk?" 2. **Lines 10-12:** Ask UEBA: "What's the user risk?" 3. **Lines 15-24:** Match network and user data by IP address 4. **Line 27:** Combine the risks using fusion engine 5. **Lines 30-31:** If risk is above 30%, create an alert

The Loop:

```
while True:
    self.run_detection_cycle()
    time.sleep(10)
```

This runs forever, checking every 10 seconds.

Analogy: Like a security guard who walks around the building every 10 minutes checking everything.

File 5: `alert_system.py` (The Alarm)

What it does: Creates and sends alerts when threats are detected

Key Parts:

Part A: Creating Alerts

```
def create_alert(self, threat_level, final_risk, network_risk, user_risk, ...):
    # Generate message
    if threat_level == "CRITICAL":
        message = "  CRITICAL THREAT DETECTED! Immediate action required."
    elif threat_level == "HIGH":
        message = "  HIGH THREAT detected. Investigation needed."
    elif threat_level == "MEDIUM":
        message = "  MEDIUM threat detected. Monitor closely."
    else:
        message = "  LOW threat level. Normal monitoring."

    alert = Alert(
        timestamp=datetime.now(),
        threat_level=threat_level,
        final_risk=final_risk,
        network_risk=network_risk,
        user_risk=user_risk,
        message=message
    )

    self.process_alert(alert)
```

In Simple Terms: - **Lines 3-10:** Choose the right message based on threat level - **Lines 12-19:** Package all the information into an Alert object - **Line 21:** Send the alert for processing

Part B: Processing Alerts

```python
def process_alert(self, alert):
    # Console notification
    self.console_notification(alert)

    # Email notification
    if alert.final_risk >= 0.6:  # HIGH or CRITICAL
        self.send_email_alert(alert)

    # Save to file
    self.save_alert_to_file(alert)
```

In Simple Terms: - **Line 3:** Show alert on screen (always) - **Lines 6-7:** Send email if risk is HIGH or CRITICAL - **Line 10:** Save to log file (always)

Multi-Channel Alerts: 1. **Console:** Always shown (you see it immediately) 2. **Email:** Only for serious threats (prevents spam) 3. **File:** Always saved (for later analysis)

How Everything Works Together

The Complete Flow:

```
1. MAIN LOOP starts
   ↓
2. IDS Engine checks CloudWatch
   → Gets: NetworkIn = 1,751,904 bytes
   → Gets: NetworkPacketsIn = 21,189 packets
   → Calculates: network_risk = 0.95 (HIGH!)
   ↓
3. UEBA Engine checks CloudTrail
   → Gets: 13 user activities
   → Analyzes: hour, day, volume, diversity
   → Calculates: user_risk = 0.10 (normal)
   ↓
4. Threat Fusion Engine combines
```

```
   → Input: network_risk = 0.95, user_risk = 0.10
   → Calculation: (0.6 × 0.95) + (0.4 × 0.10) = 0.61
   → Output: final_risk = 0.61, level = "HIGH"
   ↓
5. Alert System triggers
   → Console: Shows red alert with details
   → Email: Sends to admin (because risk > 0.6)
   → File: Saves to threat_alerts.log
   ↓
6. Wait 10 seconds
   ↓
7. Go back to step 1
```

Key Code Concepts to Understand

1. Why 60/40 Weighting?

```
final_risk = (0.6 * network_risk) + (0.4 * user_risk)
```

Explanation: - Network attacks (DDoS) cause immediate damage → Higher priority (60%) - User behavior changes are slower → Lower priority (40%) - This weighting is YOUR RESEARCH CONTRIBUTION

Example to explain: "Imagine a fire alarm (network) and a smoke detector (user behavior). The fire alarm is more urgent (60%) but the smoke detector provides context (40%). Together they give you the full picture."

2. Why Thresholds?

```
if network_in > 8_000_000 or packets_in > 15_000:
    risk = 0.95
```

Explanation: - Based on normal traffic patterns - Normal: ~1,000 bytes, ~10 packets - Attack: 1,000,000+ bytes, 10,000+ packets - Thresholds set at 8M bytes or 15K packets for CRITICAL

How thresholds were chosen: 1. Observed normal traffic: ~1,000 bytes 2. Observed attack traffic: ~1,750,000 bytes 3. Set thresholds at different levels to catch varying attack intensities

3. Why Machine Learning for UEBA?

```
df["anomaly_score"] = self.model.decision_function(features)
```

Explanation: - User behavior is complex (can't use simple thresholds) - ML model learns what's "normal" for each user - Detects deviations from normal patterns

Analogy: "Like how Netflix knows what you normally watch. If you suddenly start watching completely different content, it's unusual. The ML model does the same for user behavior."

4. Why Real-Time (10 seconds)?

```
time.sleep(10)
```

Explanation: - Literature: 30-60 seconds - Your system: 10 seconds - **You're 2-3x faster!**

Why it matters: "In cybersecurity, every second counts. A DDoS attack can take down a server in minutes. Detecting it 20 seconds faster means you can respond before major damage occurs."

Continue to Part 3 for demo preparation and Q&A...