Binary Parser

```
import struct
import socket
import datetime
from collections import defaultdict, Counter

def parse_binary_logs(file_path):
    """Parse binary log file according to the specified format."""
    login_attempts = []

    with open(file_path, 'rb') as f:
        data = f.read()

    offset = 0
    while offset < len(data):
        # Read username length (4-byte integer, big-endian)
        username_length = struct.unpack('>I', data[offset:offset+4])[0]
        offset += 4

        # Read username (variable length string)
        username = data[offset:offset+username_length].decode('utf-8')
        offset += username_length

        # Read IPv4 address (4 bytes)
        ip_bytes = data[offset:offset+4]
        ip_address = socket.inet_ntoa(ip_bytes)
        offset += 4

        # Read timestamp (4-byte Unix timestamp)
        timestamp = struct.unpack('>I', data[offset:offset+4])[0]
        datetime_obj = datetime.datetime.fromtimestamp(timestamp, tz=datetime.timezone.utc)
        offset += 4

        # Read success flag (1-byte boolean)
        success = bool(data[offset])
        offset += 1
```

```
        # Store the parsed login attempt
        login_attempts.append({
            'username': username,
            'ip_address': ip_address,
            'timestamp': timestamp,
            'datetime': datetime_obj,
            'success': success
        })

    return login_attempts

def analyze_logs(login_attempts):
    """Basic analysis of the log data to extract key metrics."""
    # Get the earliest timestamp (start date of the log)
    earliest_timestamp = min(login_attempts, key=lambda x: x['timestamp'])['timestamp']
    start_date_utc = datetime.datetime.fromtimestamp(earliest_timestamp,
tz=datetime.timezone.utc)

    # Count unique usernames
    unique_usernames = set(attempt['username'] for attempt in login_attempts)

    # Count unique IP addresses
    unique_ips = set(attempt['ip_address'] for attempt in login_attempts)

    # Count total login attempts
    total_attempts = len(login_attempts)

    return {
        'start_date_utc': start_date_utc,
        'total_attempts': total_attempts,
        'unique_usernames': len(unique_usernames),
        'unique_ips': len(unique_ips),
        'usernames': unique_usernames,
        'ip_addresses': unique_ips
    }

def analyze_login_patterns(login_attempts):
    """Analyze login patterns to identify potentially compromised users."""
    # Track login data per user
    user_data = defaultdict(lambda: {
        'ips': set(),
```

```
    'successful_logins': 0,
    'failed_logins': 0,
    'login_times': [],
    'success_after_failure': 0,
    'unique_ips_count': 0
})

# Group login attempts by user
user_attempts = defaultdict(list)
for attempt in login_attempts:
    user = attempt['username']
    user_attempts[user].append(attempt)

    # Track IPs used
    user_data[user]['ips'].add(attempt['ip_address'])

    # Track login success/failure
    if attempt['success']:
        user_data[user]['successful_logins'] += 1
    else:
        user_data[user]['failed_logins'] += 1

    # Track login times
    user_data[user]['login_times'].append(attempt['timestamp'])

# Calculate additional metrics
for user, attempts in user_attempts.items():
    # Sort attempts by timestamp
    sorted_attempts = sorted(attempts, key=lambda x: x['timestamp'])

    # Check for successful login after failures
    for i in range(1, len(sorted_attempts)):
        if (not sorted_attempts[i-1]['success'] and
            sorted_attempts[i]['success'] and
            sorted_attempts[i]['ip_address'] != sorted_attempts[i-1]['ip_address']):
            user_data[user]['success_after_failure'] += 1

    # Count unique IPs
    user_data[user]['unique_ips_count'] = len(user_data[user]['ips'])

# Identify suspicious users based on multiple criteria
```

```
    suspicious_users = []
    for user, data in user_data.items():
        suspicion_score = 0

        # Multiple IPs used (especially if significantly more than other users)
        if data['unique_ips_count'] > 3:
            suspicion_score += data['unique_ips_count']

        # High number of failed logins followed by successful ones from different IPs
        if data['success_after_failure'] > 0:
            suspicion_score += data['success_after_failure'] * 10

        # High ratio of failed to successful logins
        if data['successful_logins'] > 0:
            failure_ratio = data['failed_logins'] / data['successful_logins']
            if failure_ratio > 3:
                suspicion_score += failure_ratio

        if suspicion_score > 10:
            suspicious_users.append((user, suspicion_score, data))

    # Sort by suspicion score
    suspicious_users.sort(key=lambda x: x[1], reverse=True)

    return suspicious_users, user_data

def main():
    file_path = input("Enter the path to the binary log file: ")

    try:
        login_attempts = parse_binary_logs(file_path)
        print(f"Successfully parsed {len(login_attempts)} login attempts.")

        # Basic log analysis for the requested metrics
        log_metrics = analyze_logs(login_attempts)

        print("\n===== LOG METRICS =====")
        print(f"Start date of the log (UTC): {log_metrics['start_date_utc'].strftime('%Y-%m-%d
%H:%M:%S UTC')}")
        print(f"Total login attempts recorded: {log_metrics['total_attempts']}")
        print(f"Number of unique usernames: {log_metrics['unique_usernames']}")
```

```
        print(f"Number of unique IP addresses: {log_metrics['unique_ips']}")

        # Advanced pattern analysis for suspicious activity
        suspicious_users, user_data = analyze_login_patterns(login_attempts)

        print("\n===== SUSPICIOUS ACTIVITY ANALYSIS =====")
        print(f"Total users analyzed: {len(user_data)}")

        if suspicious_users:
            print("\nPotentially compromised users (sorted by suspicion score):")
            for user, score, data in suspicious_users:
                print(f"\nUsername: {user} (Suspicion Score: {score:.2f})")
                print(f"  Unique IPs: {data['unique_ips_count']}")
                print(f"  Successful logins: {data['successful_logins']}")
                print(f"  Failed logins: {data['failed_logins']}")
                print(f"  Successful logins after failures from different IPs:
{data['success_after_failure']}")
                print(f"  IP addresses used: {', '.join(data['ips'])}")
        else:
            print("\nNo suspicious users identified.")

    except Exception as e:
        print(f"Error processing the file: {e}")

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

Revision #1

Created 2025-11-25 18:02:16 UTC by David Rizzo

Updated 2025-11-25 18:03:17 UTC by David Rizzo