In [ ]:
```
#TASK 1: THE MULTI VECTOR FIREWALL

def analyze_traffic(source_name, *ips, **metadata):

    ip_count = len(ips)

    tags = ", ".join(metadata.keys())

    return f"Source: {source_name} | Analyzed {ip_count} IPs | Security
Tags: {tags}"

if __name__ == "__main__":

    # Test Case 1: Standard Input (Mastery)
    print(analyze_traffic("Mainframe_A", "192.168.1.1", "10.0.0.5",
"172.16.0.1",
                          threat_level="High", protocol="SSH"))
```

```
Source: Mainframe_A | Analyzed 3 IPs | Security Tags:
threat_level, protocol
```

In [2]:
```
# Test Case 2: Edge Case (Zero IPs)

print(analyze_traffic("Server_B", threat_level="Low"))
```

```
Source: Server_B | Analyzed 0 IPs | Security Tags: threat_level
```

In [3]:
```
# Test Case 3: Edge Case (Empty Tags)

print(analyze_traffic("Gateway_C", "1.1.1.1"))
```

```
Source: Gateway_C | Analyzed 1 IPs | Security Tags:
```

In [4]:
```
#TASK 2: THE INSTANT PAYLOAD DECODER (LAMBDA AND MAP)

def decode_payload(payload):

    decoded = list(map(lambda x: x * 2 if x % 2 == 0 else x + 5,
payload))

    return decoded

if __name__ == "__main__":

    # Test Case 1: Standard Input (Mastery)

    payload = [10, 15, 20, 25]
    print(f"Original: {payload}")
    print(f"Decoded:  {decode_payload(payload)}") # Expected: [20, 20,
40, 30]
```

```
Original: [10, 15, 20, 25]
Decoded:  [20, 20, 40, 30]
```

In [5]:
```
 # Test Case 2: Edge Case (Empty Payload)

print(f"Empty:    {decode_payload([])}")
```

```
Empty:    []
```

In [6]:
```
# Test Case 3: Edge Case (Negative Numbers)

print(f"Negative: {decode_payload([-2, -3])}")
```

```
Negative: [-4, 2]
```

In [7]:
```
#TASK 3:THE DARKNET TRAFFIC PURGE (FILTER AND SCOPE)

BLACKLIST_RANGE = range(1000, 2000)

def filter_ports(ports):

    BLACKLIST_RANGE = range(0, 500)

    global_blacklist = globals()['BLACKLIST_RANGE']

    filtered_results = list(filter(lambda p: p not in global_blacklist,
ports))

    return filtered_results

if __name__ == "__main__":

    # Test Case 1: Standard Input (Mastery)

    ports = [80, 443, 1050, 21, 1500]
    print(f"Original Ports: {ports}")
    print(f"Purged Traffic: {filter_ports(ports)}")
```

```
Original Ports: [80, 443, 1050, 21, 1500]
Purged Traffic: [80, 443, 21]
```

In [8]:
```
# Test Case 2: Edge Case (Empty List)

print(f"Empty List:     {filter_ports([])}")
```

```
Empty List:     []
```

In [9]:
```
# Test Case 3: Edge Case (All Blacklisted)

print(f"All Blocked:    {filter_ports([1100, 1200, 1999])}")
```

```
All Blocked:    []
```

In [10]:
```
#TASK 4: THE SIGNATURE HASH AGGREGATOR

from functools import reduce

def aggregate_signatures(tokens):

    master_hash = reduce(
        lambda acc, x: acc * x if (acc * x) < 1000000 else x,
        tokens
    )

    return master_hash

if __name__ == "__main__":

    # Test Case 1: Standard Input (Mastery)

    tokens = [10, 100, 5, 200, 2]
    print(f"Tokens: {tokens}")
    print(f"Master Hash: {aggregate_signatures(tokens)}")
```

```
Tokens: [10, 100, 5, 200, 2]
Master Hash: 400
```

In [11]:
```
# Test Case 2: Edge Case (Small Tokens - No Reset)

print(f"Small Tokens: {aggregate_signatures([2, 3, 4])}")
```

```
Small Tokens: 24
```

In [12]:
```
# Test Case 3: Edge Case (Immediate Overflow)

print(f"Immediate Reset: {aggregate_signatures([2000000, 5])}")
```

```
Immediate Reset: 5
```

In [13]:
```
#TASK 5: MODULAR THREAT DISPATCHER (HIGHER ORDER FUNCTIONS)

def security_engine(data_stream, strategy):

    return strategy(data_stream)

def detection_strategy(stream):

    boosted = map(lambda x: x * 1.5, stream)

    cleaned = filter(lambda x: x > 100, boosted)

    return list(cleaned)

if __name__ == "__main__":

    # Test Case 1: Standard Input (Mastery)

    raw_stream = [50, 80, 120, 200]
    final_output = security_engine(raw_stream, detection_strategy)

    print(f"Raw Stream: {raw_stream}")
    print(f"Processed:  {final_output}") # Expected: [120.0, 180.0,
300.0]
```

```
Raw Stream: [50, 80, 120, 200]
Processed:  [120.0, 180.0, 300.0]
```

In [14]:
```
# Test Case 2: Edge Case (Empty Stream)

print(f"Empty:      {security_engine([], detection_strategy)}")
```

```
Empty:      []
```