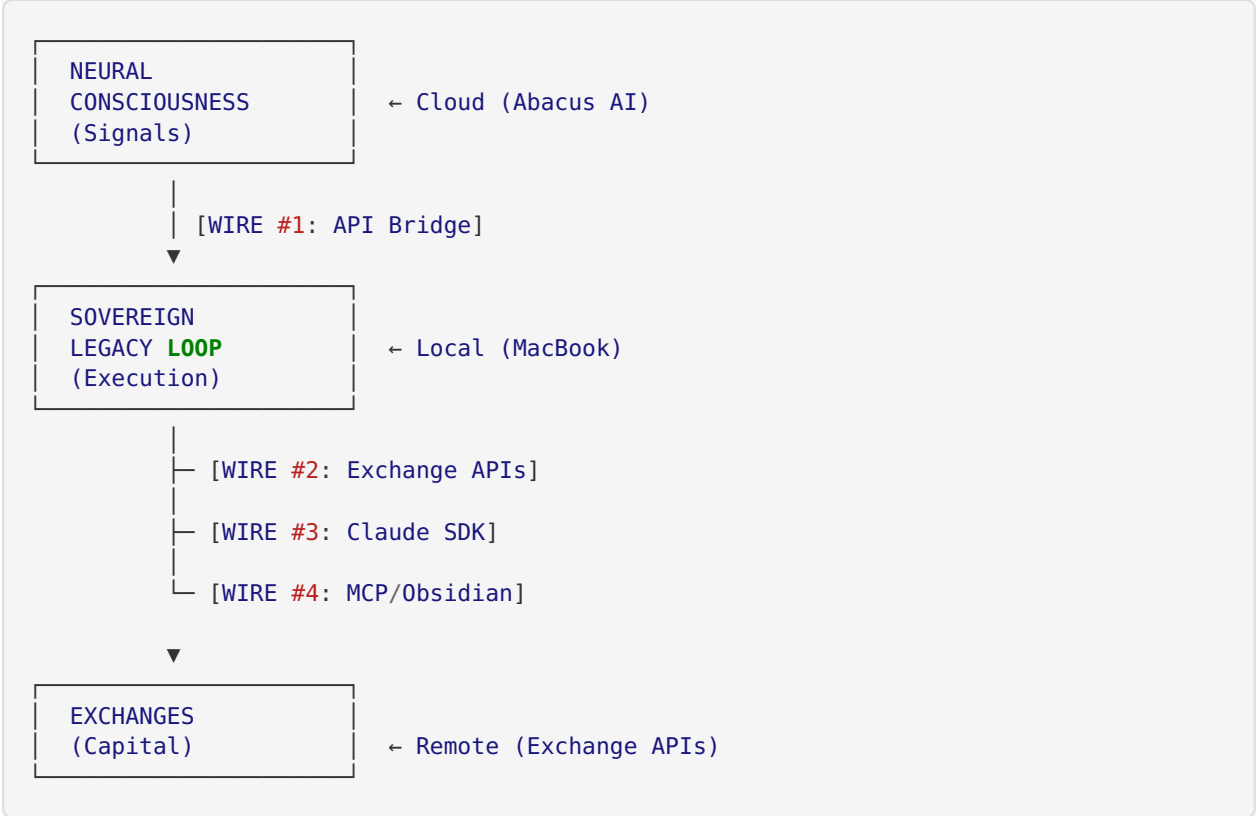
🔌 SOVEREIGN SHADOW - WIRING & INTEGRATION GUIDE

What needs to be connected and how

🎯 INTEGRATION OVERVIEW

Your system has 4 major components that need to be wired together:



🔌 WIRE #1: NEURAL CONSCIOUSNESS → LOCAL EXECUTION

Current State

✅ **Neural Consciousness:** LIVE at https://legacyloopshadowai.abacusai.app
⚠️ **Local Connection:** NOT YET WIRED

What Needs to Happen

The cloud brain needs to send signals to your local execution system.

Implementation Options

Option A: Webhook (Recommended)

```
# In Neural Consciousness (Abacus AI):
# When opportunity detected, POST to local endpoint
import requests

def send_signal_to_local(opportunity):
    webhook_url = "http://your-macbook-ip:8080/api/signals"
    signal = {
        "type": "arbitrage",
        "exchange_1": "coinbase",
        "exchange_2": "okx",
        "asset": "BTC/USDT",
        "spread": 0.027,  # 2.7%
        "confidence": 0.95,
        "timestamp": "2025-10-16T15:30:00Z"
    }
    requests.post(webhook_url, json=signal)
```

```
# In sovereign_shadow_unified.py:
# Listen for incoming signals
from flask import Flask, request

app = Flask(__name__)

@app.route('/api/signals', methods=['POST'])
def receive_signal():
    signal = request.json
    if signal['spread'] >= 0.025:  # 2.5% minimum
        execute_arbitrage(signal)
    return {"status": "received"}

# Start webhook listener in background thread
```

Option B: Polling (Simpler, Higher Latency)

```
# In sovereign_shadow_unified.py:
# Poll Abacus AI API every 30 seconds
import requests
import time

def poll_neural_consciousness():
    while True:
        response = requests.get("https://legacyloopshadowai.abacusai.app/api/signals")
        signals = response.json()

        for signal in signals:
            if signal['spread'] >= 0.025:
                execute_arbitrage(signal)

        time.sleep(30)  # Check every 30 seconds
```

Implementation Steps

1. **Expose API in Neural Consciousness**
   - Create `/api/signals` endpoint

   - Return list of detected opportunities
   - Include confidence scores

2. **Create Signal Receiver in Local System**
   - Add Flask or FastAPI server
   - Listen on port 8080
   - Validate incoming signals

3. **Test Integration**
   - Send mock signal from Abacus AI
   - Verify local system receives it
   - Check signal processing logic

Files to Modify

- `neural_consciousness/api_bridge.py` - Create this file
- `sovereign_shadow_unified.py` - Add signal receiver
- `.env.production` - Add `NEURAL_CONSCIOUSNESS_API_URL`

Configuration

```
# Add to .env.production
NEURAL_CONSCIOUSNESS_API_URL=https://legacyloopshadowai.abacusai.app/api/signals
NEURAL_CONSCIOUSNESS_API_KEY=your_api_key_here
SIGNAL_POLL_INTERVAL=30  # seconds
```

🔌 WIRE #2: LOCAL SYSTEM → EXCHANGE APIs

Current State

✅ **Exchange Connectors:** Code complete
⚠️ **API Keys:** Need to be added
⚠️ **Testing:** Not yet validated with real accounts

What Needs to Happen

Your local system needs to authenticate and trade on exchanges.

Implementation Steps

Step 1: Add API Keys to .env.production

```
# Navigate to system
cd /Volumes/LegacySafe/SovereignShadow/sovereign_legacy_loop

# Copy template
cp .env.production.template .env.production

# Edit file
nano .env.production
```

Add these keys:

```
# Coinbase
COINBASE_API_KEY=your_key_here
COINBASE_API_SECRET=your_secret_here

# OKX
OKX_API_KEY=your_key_here
OKX_API_SECRET=your_secret_here
OKX_API_PASSPHRASE=your_passphrase_here

# Kraken
KRAKEN_API_KEY=your_key_here
KRAKEN_API_SECRET=your_secret_here

# Ledger (READ-ONLY)
LEDGER_API_KEY=your_readonly_key_here
LEDGER_READ_ONLY=true  # MUST ALWAYS BE TRUE
```

Step 2: Get API Keys from Exchanges

Coinbase:
1. Log in to Coinbase
2. Go to Settings → API
3. Create New API Key
4. Permissions: `wallet:accounts:read`, `wallet:buys:create`, `wallet:sells:create`
5. Save key + secret (shown only once!)

OKX:
1. Log in to OKX
2. Profile → API
3. Create API
4. Permissions: Read + Trade (NO withdraw)
5. Set IP whitelist to your MacBook IP
6. Save key + secret + passphrase

Kraken:
1. Log in to Kraken
2. Settings → API
3. Generate New Key
4. Permissions: Query Funds, Create & Modify Orders
5. Save key + secret

Ledger (via Coinbase):
1. Connect Ledger to Coinbase
2. Create READ-ONLY API key
3. Permissions: ONLY `wallet:accounts:read`
4. Save key separately from trading keys

Step 3: Validate Connections

```
python3 scripts/validate_api_connections.py
```

Expected output:

```
Validating Coinbase...
✅ Coinbase: Connected
   Balance: $1,660.00 USD
   API Permissions: Read, Trade

Validating OKX...
✅ OKX: Connected
   Balance: $0.00 USD
   API Permissions: Read, Trade

Validating Kraken...
✅ Kraken: Connected
   Balance: $0.00 USD
   API Permissions: Read, Trade

Validating Ledger...
✅ Ledger: Connected (READ-ONLY)
   Balance: $6,600.00 (BTC + ETH)
   API Permissions: Read ONLY ✅

Summary:
✅ All exchanges connected successfully
✅ Trading exchanges: 3 of 3
✅ Total tradeable capital: $1,660
✅ Cold storage verified: $6,600 (READ-ONLY)
```

Files Involved

- `.env.production` - API credentials (GITIGNORED)
- `exchanges/coinbase/connector.py` - Coinbase API wrapper
- `exchanges/okx/connector.py` - OKX API wrapper
- `exchanges/kraken/connector.py` - Kraken API wrapper
- `scripts/validate_api_connections.py` - Validation script

Security Checklist

- [ ] API keys stored in `.env.production` (gitignored)
- [ ] No keys hardcoded in Python files
- [ ] All keys use `os.getenv()` to load
- [ ] Ledger key is READ-ONLY (verified)
- [ ] Exchange keys have NO WITHDRAW permission
- [ ] 2FA enabled on all exchange accounts
- [ ] IP whitelist configured (if exchange supports)

🔌 WIRE #3: LOCAL SYSTEM → CLAUDE SDK

Current State

✅ **Claude SDK:** 5,000+ files present
⚠️ **Integration:** Needs to be called by trading strategies
⚠️ **Testing:** Not yet validated

What Needs to Happen

Trading strategies should consult Claude SDK for decision support.

Implementation Approach

```
# In trading_systems/arbitrage/claude_arbitrage_trader.py

from claudeSDK import MarketAnalyzer, OpportunityScorer

class ArbitrageTrader:
    def __init__(self):
        self.analyzer = MarketAnalyzer()
        self.scorer = OpportunityScorer()

    def evaluate_opportunity(self, spread_data):
        # Get basic arbitrage metrics
        spread_percent = spread_data['spread']
        volume = spread_data['volume']

        # Consult Claude SDK for deeper analysis
        market_sentiment = self.analyzer.get_sentiment(spread_data['asset'])
        risk_score = self.scorer.calculate_risk(spread_data)

        # Decision logic
        if spread_percent >= 0.025:  # 2.5% minimum
            if market_sentiment > 0.6:  # Bullish sentiment
                if risk_score < 0.3:  # Low risk
                    return "EXECUTE"

        return "SKIP"
```

Integration Points

Market Analysis:

```
from claudeSDK.analysis import MarketSentiment

sentiment = MarketSentiment()
btc_sentiment = sentiment.analyze("BTC/USDT")
# Returns: {"score": 0.75, "signals": ["bullish"], "confidence": 0.82}
```

Pattern Recognition:

```
from claudeSDK.patterns import PatternDetector

detector = PatternDetector()
patterns = detector.find_patterns("BTC/USDT", timeframe="1h")
# Returns: ["ascending_triangle", "volume_surge"]
```

Risk Assessment:

```python
from claudeSDK.risk import RiskCalculator

risk_calc = RiskCalculator()
risk_score = risk_calc.assess_trade({
    "asset": "BTC/USDT",
    "position_size": 250,
    "strategy": "arbitrage",
    "market_conditions": "volatile"
})
# Returns: {"risk_level": "low", "score": 0.23, "recommended_size": 250}
```

Files to Modify

- `trading_systems/arbitrage/claude_arbitrage_trader.py` - Add SDK calls
- `trading_systems/sniping/token_sniper.py` - Add SDK calls
- `trading_systems/scalping/scalp_trader.py` - Add SDK calls

Implementation Steps

1. **Import Claude SDK modules in each strategy**
2. **Add decision support calls before trade execution**
3. **Test with paper trading to verify SDK responses**
4. **Tune confidence thresholds based on results**

🔌 WIRE #4: LOCAL SYSTEM → MCP/OBSIDIAN KEY VAULT

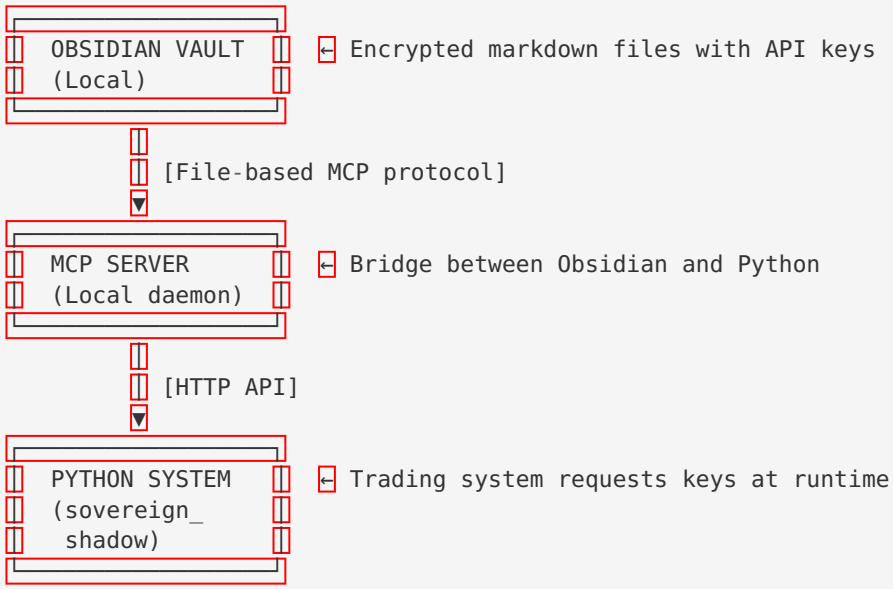
Current State

✅ **Obsidian Vault:** Setup and encrypted
✅ **MCP Server:** Configured
⚠️ **Integration:** Not yet connected to Python system

What Needs to Happen

Python system should retrieve API keys from Obsidian vault via MCP server.

Architecture



Implementation

Step 1: Structure Obsidian Vault

Create files in your Obsidian vault:

File: `API_Keys/Coinbase.md`

```
# Coinbase API Keys

## Production
- API Key: `your_coinbase_key_here`
- API Secret: `your_coinbase_secret_here`
- Permissions: Read, Trade
- Created: 2025-10-16
- Status: Active

## Notes
- Used by Sovereign Shadow trading system
- NO WITHDRAW permission
- Rotated every 90 days
```

File: `API_Keys/OKX.md`

```
# OKX API Keys

## Production
- API Key: `your_okx_key_here`
- API Secret: `your_okx_secret_here`
- Passphrase: `your_okx_passphrase_here`
- Permissions: Read, Trade
- IP Whitelist: Your MacBook IP
- Created: 2025-10-16
- Status: Active
```

Step 2: Start MCP Server

```
# In your Obsidian vault directory
cd ~/Obsidian/SovereignShadow
mcp-server --vault-path . --port 9999
```

Step 3: Create Python MCP Client

File: `security/mcp_client.py`

```python
import requests
import os

class MCPKeyVault:
    def __init__(self):
        self.mcp_url = os.getenv("MCP_SERVER_URL", "http://localhost:9999")

    def get_api_key(self, exchange, key_type="key"):
        """
        Retrieve API key from Obsidian vault via MCP

        Args:
            exchange: "coinbase", "okx", "kraken"
            key_type: "key", "secret", "passphrase"
        """
        response = requests.get(
            f"{self.mcp_url}/api/keys/{exchange}/{key_type}"
        )

        if response.status_code == 200:
            return response.json()['value']
        else:
            raise Exception(f"Failed to retrieve {exchange} {key_type}")

    def rotate_key(self, exchange, new_key, new_secret):
        """
        Rotate API keys in vault
        """
        response = requests.post(
            f"{self.mcp_url}/api/keys/{exchange}/rotate",
            json={"key": new_key, "secret": new_secret}
        )
        return response.status_code == 200
```

Step 4: Use in Trading System

```
# In sovereign_shadow_unified.py
from security.mcp_client import MCPKeyVault

vault = MCPKeyVault()

# Get keys from Obsidian vault instead of .env
coinbase_key = vault.get_api_key("coinbase", "key")
coinbase_secret = vault.get_api_key("coinbase", "secret")

# Use keys for exchange connection
coinbase_client = CoinbaseConnector(
    api_key=coinbase_key,
    api_secret=coinbase_secret
)
```

Configuration

```
# Add to .env.production
MCP_SERVER_URL=http://localhost:9999
MCP_VAULT_PATH=/Users/yourname/Obsidian/SovereignShadow
USE_MCP_VAULT=true  # Set false to use .env keys instead
```

Security Benefits

- **Encrypted Storage:** Obsidian vault can be encrypted
- **Centralized Management:** One place to update all keys
- **Audit Trail:** Obsidian tracks changes to key files
- **Easy Rotation:** Update keys in Obsidian, system picks up automatically
- **No .env Exposure:** Keys never touch git or .env files

🧪 INTEGRATION TESTING PLAN

Phase 1: Individual Component Testing

Test 1: Exchange Connections (30 min)

```
python3 scripts/validate_api_connections.py
```

Pass Criteria:
- [ ] All 3 trading exchanges connect
- [ ] Ledger shows as READ-ONLY
- [ ] Balances displayed correctly

Test 2: Claude SDK Integration (1 hour)

```
python3 -c "
from claudeSDK.analysis import MarketSentiment
s = MarketSentiment()
print(s.analyze('BTC/USDT'))
"
```

Pass Criteria:
- [ ] SDK imports successfully
- [ ] Market analysis returns data
- [ ] No errors in output

Test 3: MCP Key Vault (30 min)

```
python3 -c "
from security.mcp_client import MCPKeyVault
vault = MCPKeyVault()
print(vault.get_api_key('coinbase', 'key'))
"
```

Pass Criteria:
- [ ] MCP server responds
- [ ] Keys retrieved successfully
- [ ] Keys match expected format

Test 4: Neural Consciousness API (30 min)

```
curl https://legacyloopshadowai.abacusai.app/api/signals
```

Pass Criteria:
- [ ] API responds with signals
- [ ] JSON format correct
- [ ] Confidence scores present

Phase 2: Integration Testing

Test 5: End-to-End Signal Flow (Paper Mode)

```
./START_SOVEREIGN_SHADOW.sh paper
```

Test Scenario:
1. Neural Consciousness detects 3% BTC/USDT arbitrage
2. Signal sent to local system
3. Claude SDK analyzes opportunity
4. Risk management validates
5. Paper trade executed and logged

Pass Criteria:
- [ ] Signal received from Neural Consciousness
- [ ] Claude SDK consulted for decision
- [ ] Risk checks passed
- [ ] Trade logged in `data/transactions/`
- [ ] No errors in logs

Test 6: Multi-Strategy Testing (Paper Mode)

Run for 24 hours in paper mode, test all strategies:
- [ ] Arbitrage strategy executes on opportunities
- [ ] Sniping strategy detects new listings
- [ ] Scalping strategy identifies micro-movements

- [ ] Laddering triggers on dips
- [ ] All-in remains disabled (safety check)

Pass Criteria:
- [ ] At least 1 paper trade per strategy
- [ ] All trades logged correctly
- [ ] P&L calculations accurate
- [ ] Risk limits respected
- [ ] No system crashes

Phase 3: Live Testing (Small Capital)

Test 7: Test Mode with $100

```
./START_SOVEREIGN_SHADOW.sh test
```

Run with maximum $100 position size for 1 week.

Pass Criteria:
- [ ] Real trades execute successfully
- [ ] Actual P&L matches predictions
- [ ] Stop losses trigger correctly
- [ ] Circuit breaker works if needed
- [ ] No unexpected behavior

📋 WIRING CHECKLIST

Pre-Wiring Setup

- [ ] All 55,379 files present on external drive
- [ ] Python 3.8+ installed
- [ ] Dependencies installed (`pip3 install -r requirements.txt`)
- [ ] `.env.production` file created
- [ ] Obsidian vault setup with API keys
- [ ] MCP server installed and configured

Wire #1: Neural Consciousness → Local

- [ ] API endpoint exposed in Neural Consciousness
- [ ] Signal receiver implemented in local system
- [ ] Webhook or polling mechanism configured
- [ ] Test signal sent and received successfully
- [ ] Signal processing logic validated

Wire #2: Local → Exchange APIs

- [ ] Coinbase API key + secret added
- [ ] OKX API key + secret + passphrase added
- [ ] Kraken API key + secret added
- [ ] Ledger READ-ONLY key added
- [ ] All connections validated with script

- [ ] Real balances displayed correctly

Wire #3: Local → Claude SDK

- [ ] SDK imported in arbitrage strategy
- [ ] SDK imported in sniping strategy
- [ ] SDK imported in scalping strategy
- [ ] Market sentiment integration tested
- [ ] Risk scoring integration tested
- [ ] Decision support logic validated

Wire #4: Local → MCP/Obsidian

- [ ] MCP server running on localhost:9999
- [ ] API keys stored in Obsidian vault
- [ ] Python MCP client created
- [ ] Key retrieval tested and working
- [ ] Fallback to .env configured
- [ ] Key rotation mechanism tested

Integration Testing

- [ ] Individual components tested
- [ ] End-to-end signal flow tested (paper)
- [ ] Multi-strategy testing completed (paper)
- [ ] Test mode validated with real $100
- [ ] 7-day test period completed successfully
- [ ] Ready for full production launch

🚀 LAUNCH SEQUENCE

Once all wiring complete:

Day 1-2: Paper Trading

```
./START_SOVEREIGN_SHADOW.sh paper
```

Run for 48 hours, monitor closely.

Day 3-4: Review & Adjust

- Analyze paper trading results
- Tune confidence thresholds
- Adjust risk parameters if needed

Day 5-7: Test Mode

```
./START_SOVEREIGN_SHADOW.sh test
```

Run with $100 max position, real money.

Day 8-14: Extended Test

Continue test mode for 1 week, validate:
- Win rate ≥ 50%
- Average win > average loss
- No critical errors
- System stability

Day 15: GO/NO-GO Decision

If all tests pass:

```
./START_SOVEREIGN_SHADOW.sh live
```

Full production launch with $1,660 capital.

🔧 TROUBLESHOOTING INTEGRATION ISSUES

Issue: Neural Consciousness not sending signals

Diagnosis:

```
curl https://legacyloopshadowai.abacusai.app/api/health
```

Solutions:
- Check if Abacus AI deployment is running
- Verify API endpoint URL is correct
- Check firewall/network blocking signals

Issue: Exchange API authentication fails

Diagnosis:

```
python3 scripts/validate_api_connections.py --verbose
```

Solutions:
- Verify API keys copied correctly (no extra spaces)
- Check if API permissions include Trade
- Confirm IP whitelist includes your MacBook
- Try regenerating API keys

Issue: Claude SDK import errors

Diagnosis:

```
python3 -c "import claudeSDK; print(claudeSDK.__version__)"
```

Solutions:
- Check if claudeSDK directory exists
- Verify `__init__.py` files present

- Install any missing SDK dependencies
- Check Python path includes SDK directory

Issue: MCP server not responding

Diagnosis:

```
curl http://localhost:9999/health
```

Solutions:
- Check if MCP server is running (`ps aux | grep mcp`)
- Verify port 9999 not in use by another process
- Restart MCP server
- Check Obsidian vault path is correct

Issue: Trades not executing

Diagnosis:

```
tail -100 logs/sovereign_shadow_latest.log
```

Solutions:
- Check if risk management blocking trades
- Verify sufficient balance in hot wallet
- Check if stop-loss would be below exchange minimum
- Review opportunity detection thresholds

📊 INTEGRATION SUCCESS METRICS

After wiring complete, you should see:

Neural Consciousness:
- ✅ Signals generated every 5-30 minutes
- ✅ API responding within 2 seconds
- ✅ Confidence scores 0.6-0.95 range

Exchange APIs:
- ✅ Order placement latency < 500ms
- ✅ Balance updates within 5 seconds
- ✅ No authentication errors

Claude SDK:
- ✅ Analysis completed within 3 seconds
- ✅ Recommendations consistent with market conditions
- ✅ Risk scores correlate with volatility

MCP Vault:
- ✅ Key retrieval < 100ms
- ✅ No failed key lookups
- ✅ Audit trail maintained

Overall System:
- ✅ Signal → Trade execution < 10 seconds
- ✅ 99%+ uptime over 24 hours
- ✅ All strategies operational
- ✅ Risk limits respected 100% of time

🎯 FINAL INTEGRATION CHECKLIST

Before declaring "SYSTEM OPERATIONAL":

- [ ] All 4 wires connected and tested
- [ ] Paper trading successful (48+ hours)
- [ ] Test mode successful (7+ days)
- [ ] Real profitable trade executed
- [ ] Stop loss confirmed working
- [ ] Circuit breaker confirmed working
- [ ] All logs clean (no critical errors)
- [ ] Neural consciousness operational
- [ ] Emergency shutdown procedures tested
- [ ] Documentation complete and reviewed

When all boxes checked: SYSTEM IS FULLY OPERATIONAL

🏴 SOVEREIGN SHADOW - WIRED & READY 🏴

Version: 1.0
Date: October 16, 2025
Status: Integration Guide

🚀 WIRE IT. TEST IT. LAUNCH IT. 🚀