Tutorial: Auto-Hold High-Risk Orders Over $5,000 with Webhooks

Overview

Audience: Power users (Product Operations, Data Analysts)
Time to complete: 20 minutes
Prerequisites: - AUTRANEX sandbox account with Sourcing Manager role - Node.js 18+ installed - Basic familiarity with REST APIs

Why Auto-Hold High-Risk Orders?

High-value orders from risky customers need review before fulfillment. Common risk indicators include:

- New customer account (< 30 days old)
- Ship-to ≠ bill-to country or OFAC-listed regions
- Sudden 10× jump in average order value
- Multiple payment methods on file, first-time AMEX usage

Compliance benefit: Automated holds create an audit trail in AUTRANEX, giving compliance teams a single source of truth for all risk decisions.

Learning Objectives

By the end of this tutorial, you'll be able to:

- ✅ Configure an Adaptive Sourcing Block to auto-hold high-value orders
- ✅ Set up a webhook listener to receive order events
- ✅ Test the complete workflow with mock payloads
- ✅ Handle webhook security and retries properly

Assets & Resources

- **Postman Collection**: Run in Postman
- **Tutorial Location**: `/tutorials/auto-hold-high-risk-orders.mdx`
- **Complete Code**: Available in the code blocks below

Tutorial Structure

Part 1: Configure the Sourcing Block (5 min)

1. **Create the High-Risk Order Rule**
    - Navigate to Sourcing > Blocks
    - Configure condition: `order_value > 5000`
    - Set action: `hold`
    - Add metadata for audit trail
2. **Activate and Verify**
    - Submit the rule
    - Check activation status
    - Review the generated `block_id`

Part 2: Set Up Webhook Subscription (5 min)

1. **Create Webhook Endpoint**
    - Navigate to Settings > Webhooks
    - Configure endpoint URL
    - Select `order.held` event type
    - Copy webhook secret for HMAC verification
2. **Configure Event Filters**
    - Filter by block_id (optional)
    - Set retry policy

Part 3: Build the Webhook Listener (8 min)

1. **Express.js Server Setup**

```
import express from 'express';
import crypto from 'crypto';

const app = express();
const PORT = process.env.PORT || 3000;

// IMPORTANT: Use raw body for HMAC verification
app.use(express.raw({ type: 'application/json' }));
```

2. **HMAC Signature Verification**

```
function verifyAutranexSignature(req, res, next) {
  const signature = req.headers['x-autranex-signature'];

  if (!signature) {
    return res.status(401).json({ error: 'Missing signature header' });
  }

  // Parse signature header
  const elements = signature.split(',');
  const timestamp = elements[0].split('=')[1];
  const receivedHash = elements[1].split('=')[1];

  // Verify timestamp (5 min tolerance)
  const currentTime = Math.floor(Date.now() / 1000);
  if (currentTime - parseInt(timestamp) > 300) {
    return res.status(401).json({ error: 'Request too old' });
  }

  // Calculate expected signature
  const payload = timestamp + '.' + req.body.toString();
  const expectedHash = crypto
    .createHmac('sha256', process.env.AUTRANEX_WEBHOOK_SECRET)
    .update(payload)
    .digest('hex');

  // Timing-safe comparison
  if (!crypto.timingSafeEqual(
    Buffer.from(receivedHash, 'hex'),
    Buffer.from(expectedHash, 'hex')
  )) {
    return res.status(401).json({ error: 'Invalid signature' });
  }

  req.body = JSON.parse(req.body.toString());
  next();
}
```

3. **Process the Order Hold Event**

```
// In-memory store for idempotency (use Redis in production)
const processedEvents = new Set();

app.post('/webhook/orders', verifyAutranexSignature, async (req, res)
=> {
  const event = req.body;

  // Idempotency check
  if (processedEvents.has(event.event_id)) {
    return res.status(200).json({ status: 'already_processed' });
```

```
  }

  // Process event
  if (event.event_type === 'order.held') {
    console.log(`Order ${event.data.order_id} held:
$${event.data.order_value}`);
    processedEvents.add(event.event_id);

    // Your business logic here
  }

  // MUST return 200 OK within 3 seconds
  res.status(200).json({ status: 'received' });
});

app.listen(PORT, () => {
  console.log(`Webhook listener on port ${PORT}`);
});
```

⚠️ **Important**: Your endpoint must return `200 OK` within 3 seconds or AUTRANEX will retry the webhook.

Part 4: Test the Complete Flow (2 min)

1. **Using the Test Helper**

```
curl -X POST https://staging.autranex.com/v2/test/webhook \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "webhook_url": "https://your-listener.com/webhook/orders",
    "event_type": "order.held",
    "order_value": 5500
  }'
```

2. **Verify the Results**
   - Check webhook logs for received event
   - Confirm order status via API:

```
curl -X GET https://staging.autranex.com/v2/orders/ord_abc123 \
  -H "Authorization: Bearer $ACCESS_TOKEN"
```

Advanced Topics

🔄 Webhook Retry Behavior & Idempotency

Retry Schedule

AUTRANEX retries failed webhooks (non-2xx responses) with exponential backoff:

Retry #	Delay	X-Retry-Count Header
1	1 min	1
2	2 min	2
3	4 min	3
4	8 min	4
5	16 min	5
6	32 min	6
7	64 min	7
8	128 min	8

Idempotency Implementation

```
// Use event_id as idempotency key
const processedEvents = new Set(); // Use Redis/DynamoDB in production

if (processedEvents.has(event.event_id)) {
  return res.status(200).json({ status: 'already_processed' });
}
```

🏢 Partner-Specific Integration Notes

Epicor Eclipse

Map AUTRANEX `HOLD` status to Eclipse `suspend`:

```
if (event.event_type === 'order.held') {
  await eclipseAPI.updateOrderStatus(event.data.order_id, 'suspend');
}
```

PartsTech

No special configuration needed - standard webhook flow works as-is.

Custom ERP

Generic status mapping available in ERP Integration Guide.

⚠️ Common Issues & Troubleshooting

Clock Skew

- AUTRANEX allows 5-minute timestamp tolerance

- Sync your server time with NTP to avoid signature failures

Payment Pre-Authorization

- Holding an order voids existing payment pre-auths
- New authorization happens on order release
- Alert finance teams to expect auth/void/reauth pattern

Debugging Signature Mismatches

1. Verify you're using the raw request body (not parsed JSON)
2. Check webhook secret matches dashboard value
3. Ensure no proxy is modifying the request body
4. Log both expected and received hashes (dev only!)

Sequence Diagram

```
graph TD
    A[Customer Order >$5000] --> B[AUTRANEX Rule Engine]
    B --> C{Matches ASB Rule?}
    C -->|Yes| D[Set Status: HOLD]
    D --> E[Trigger Webhook]
    E --> F[Your Listener]
    F --> G[Verify HMAC]
    G --> H[Process Event]
    H --> I[Return 200 OK]
    I --> J[Order Held Successfully]
    H --> K[Optional: PATCH /v2/orders/:id]
    K --> L[Update Order Metadata]
```

Related Resources

- Adaptive Sourcing Blocks API Reference
- Webhook Security Guide
- Risk Signals API (Coming Q3)
- Order Review & Release Policy (Internal)

Next Steps

- Implement additional risk scoring logic (see Risk Signals API when available)
- Set up monitoring dashboards for held order volume
- Configure alert thresholds for unusual hold patterns

Last verified against API version: 2.0 (January 2025)