Security Migration Guide

Overview

This guide walks through the security improvements implemented to fix critical vulnerabilities in the Payroll-ByteMy application.

🔒 Security Fixes Implemented

1. Admin Secret Removal

Before: Hasura admin secret was exposed in client-accessible code
After: All admin operations use a secure service with role-based authentication

Changes Made:

1. **Created Secure Hasura Service** (`lib/secure-hasura-service.ts`)
   - Uses service account token instead of admin secret
   - Validates user permissions before executing queries
   - Provides typed methods for common operations
2. **Implemented API Authentication** (`lib/api-auth.ts`)
   - Role-based access control with hierarchy
   - Rate limiting for sensitive operations
   - Webhook signature validation
   - CRON job authentication
3. **Updated All API Routes**
   - Removed direct admin secret usage
   - Added authentication middleware
   - Implemented rate limiting
   - Added audit logging

2. Security Headers & CORS

Before: No security headers or CORS configuration
After: Comprehensive security headers and strict CORS policy

Headers Added:

- `X-Frame-Options: DENY` - Prevents clickjacking
- `X-Content-Type-Options: nosniff` - Prevents MIME sniffing
- `Content-Security-Policy` - Restricts resource loading
- `Strict-Transport-Security` - Enforces HTTPS
- CORS headers with origin validation

3. Rate Limiting

Before: No rate limiting on API endpoints
After: Configurable rate limits on all sensitive operations

Implementation:

- In-memory rate limiting (can be upgraded to Redis)
- Different limits for different operations
- Automatic cleanup of expired entries

📋 Migration Steps

Step 1: Update Environment Variables

1. Remove `HASURA_ADMIN_SECRET` from all `.env` files
2. Add new secure variables:

```
# Generate secure secrets
openssl rand -base64 32  # For API_SECRET_KEY
openssl rand -base64 32  # For SESSION_SECRET
openssl rand -base64 32  # For CRON_SECRET
```

3. Create a Hasura service account:
    - Go to Hasura Console → Settings → API Tokens
    - Create a new token with limited permissions
    - Add as `HASURA_SERVICE_ACCOUNT_TOKEN`

Step 2: Update Hasura Permissions

1. Remove admin secret from Hasura Cloud settings
2. Configure JWT authentication with Clerk
3. Set up role-based permissions:

```
# Example permission for admin role
{
  "role": "admin",
  "permission": {
    "columns": "*",
    "filter": {},
    "check": {}
  }
}
```

Step 3: Update API Routes

All API routes now use the `withAuth` wrapper:

```
// Before
export async function POST(request: NextRequest) {
  // No authentication
  const result = await adminApolloClient.mutate({...});
}

// After
export const POST = withAuth(
  async (request: NextRequest, session) => {
    // Authenticated with session info
    const result = await secureHasuraService.executeAdminMutation({...});
  },
  { requiredRole: "admin" }
);
```

Step 4: Update Frontend Code

1. Remove any direct GraphQL calls with admin privileges
2. Use authenticated API routes instead
3. Handle new error responses:

```
// Handle authentication errors
try {
  const response = await fetch("/api/staff/create", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(data),
  });

  if (response.status === 401) {
    // Redirect to login
  } else if (response.status === 403) {
    // Show permission denied message
  } else if (response.status === 429) {
    // Show rate limit message
  }
} catch (error) {
  // Handle network errors
}
```

Step 5: Test Security

1. **Authentication Tests**:
   - Try accessing protected routes without login
   - Test with different user roles

- Verify rate limiting works

2. **Security Headers**:

```
# Check security headers
curl -I https://your-app.com
```

3. **CORS Testing**:

- Test API calls from different origins
- Verify only allowed origins work

🚨 Breaking Changes

API Response Format

All API routes now return consistent error responses:

```
// Unauthorized (401)
{
  "error": "Unauthorized: No active session",
  "code": "UNAUTHORIZED"
}

// Forbidden (403)
{
  "error": "Forbidden: Insufficient permissions",
  "code": "FORBIDDEN"
}

// Rate Limited (429)
{
  "error": "Rate limit exceeded",
  "code": "RATE_LIMITED"
}
```

Required Headers

CRON jobs now require authentication:

```
curl -X POST https://your-app.com/api/cron/generate-bulk-dates \
  -H "x-cron-secret: your-cron-secret" \
  -H "Content-Type: application/json"
```

🔍 Monitoring & Logging

Security Events

All admin operations are now logged with:

- User ID performing the action
- Timestamp
- Operation details
- IP address (when available)

Rate Limit Monitoring

Monitor rate limit hits:

```
// In your monitoring dashboard
{
  event: "rate_limit_exceeded",
  endpoint: "/api/developer/clean-all-dates",
  userId: "user_123",
  timestamp: "2024-01-27T10:30:00Z"
}
```

📚 Best Practices Going Forward

1. **Never expose admin secrets** in client code
2. **Always authenticate** API routes
3. **Use role-based access** for sensitive operations
4. **Implement rate limiting** on destructive operations
5. **Log security events** for audit trails
6. **Regular security audits** of dependencies

🆘 Troubleshooting

Common Issues

1. **"Unauthorized" errors after migration**
   - Ensure user has correct role in Clerk metadata
   - Check JWT token includes role claim
2. **"Service account token not configured"**
   - Create service account in Hasura
   - Add token to environment variables
3. **CORS errors**
   - Check `NEXT_PUBLIC_APP_URL` is set correctly
   - Verify origin is allowed in `next.config.ts`

Debug Mode

Enable debug logging for security issues:

```
// In development only
if (process.env.NODE_ENV === "development") {
  console.log("Auth session:", session);
  console.log("Rate limit status:", rateLimitStatus);
}
```

📞 Support

If you encounter issues during migration:

1. Check error logs in Vercel/server logs
2. Verify all environment variables are set
3. Test with minimal permissions first
4. Contact security team for assistance

Important: This migration must be completed before deploying to production. The old insecure endpoints will be removed in the next major version.