Authentication & Security

Any code or data structure shown below is sample code, and sample data structures provided as best practice. As we start implementing this in our project, I expect the engineering team to update this page.

Authentication Flow

API Key Authentication

```
interface ApiKey {
  id: string;
  key: string; // Hashed
  roles: string[];
  permissions: Permission[];
  metadata: {
    createdAt: Date;
    lastUsed: Date;
    createdBy: string;
    environment: string;
  };
}

const authenticate = async (req: Request, res: Response, next: NextFunction) => {
  const apiKey = req.headers['x-api-key'];

  if (!apiKey) {
    throw new AuthError('API key required');
  }

  const keyDetails = await validateApiKey(apiKey);
  req.auth = {
    keyId: keyDetails.id,
    roles: keyDetails.roles,
    permissions: keyDetails.permissions
  };

  next();
};
```

Wallet Authentication

```
interface WalletAuth {
  address: string;
  nonce: string;
  signature: string;
  timestamp: number;
}

const verifyWalletSignature = async (req: Request, res: Response, next: NextFunction) => {
  const { address, nonce, signature, timestamp } = req.body;

  // Verify timestamp is recent
  if (Date.now() - timestamp > 5 * 60 * 1000) {
    throw new AuthError('Signature expired');
  }

  // Verify signature
  const message = `${nonce}:${timestamp}`;
  const recoveredAddress = ethers.utils.verifyMessage(message, signature);

  if (recoveredAddress.toLowerCase() !== address.toLowerCase()) {
    throw new AuthError('Invalid signature');
  }

  req.wallet = { address };
  next();
};
```

Authorization

Role-Based Access Control

```
interface Permission {
  resource: string;
  action: 'CREATE' | 'READ' | 'UPDATE' | 'DELETE';
  conditions?: Record<string, any>;
}

const ROLES = {
  ADMIN: {
    permissions: [
```

```
      { resource: '*', action: '*' }
    ]
  },
  VAULT_CREATOR: {
    permissions: [
      { resource: 'vault', action: 'CREATE' },
      { resource: 'vault', action: 'READ' }
    ]
  },
  VOTER: {
    permissions: [
      { resource: 'proposal', action: 'READ' },
      { resource: 'vote', action: 'CREATE' },
      {
        resource: 'vault',
        action: 'READ',
        conditions: { status: 'ACTIVE' }
      }
    ]
  }
};

const checkPermission = (required: Permission) => {
  return (req: Request, res: Response, next: NextFunction) => {
    const hasPermission = req.auth.permissions.some(p =>
      matchPermission(p, required)
    );

    if (!hasPermission) {
      throw new AuthError('Insufficient permissions');
    }

    next();
  };
};
```

Blockchain Security

Transaction Signing

```
interface SignedTransaction {
  hash: string;
  signature: string;
  signers: string[];
  threshold: number;
}

const validateTransaction = async (tx: SignedTransaction) => {
  // Verify sufficient signers
  if (tx.signers.length < tx.threshold) {
    throw new SecurityError('Insufficient signatures');
  }

  // Verify all signatures
  for (const signer of tx.signers) {
    const isValid = await verifySignature(tx.hash, signer);
    if (!isValid) {
      throw new SecurityError('Invalid signature');
    }
  }

  return true;
};
```

Smart Contract Interaction

```
const executeSecureContractCall = async (
  contract: ethers.Contract,
  method: string,
  params: any[],
  options: SecurityOptions
) => {
  // Validate parameters
  const validated = await validateContractParams(method, params);

  // Simulate transaction
  const simulation = await contract.callStatic[method](...validated);

  // Check security constraints
  if (!meetsSecurityConstraints(simulation, options)) {
```

```
    throw new SecurityError('Security constraints not met');
  }

  // Execute transaction
  return await contract[method](...validated);
};
```

Rate Limiting

Implementation

```
import rateLimit from 'express-rate-limit';
import RedisStore from 'rate-limit-redis';

const rateLimiter = rateLimit({
  store: new RedisStore({
    client: redisClient,
    prefix: 'rate-limit:'
  }),
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: (req) => {
    // Adjust limits based on authentication level
    if (req.auth?.roles.includes('ADMIN')) {
      return 1000;
    }
    return 100;
  },
  keyGenerator: (req) => {
    return `${req.auth?.keyId || req.ip}`;
  }
});
```

Security Headers

Configuration

```
import helmet from 'helmet';

app.use(helmet({
  contentSecurityPolicy: {
```

```
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'"],
      styleSrc: ["'self'"],
      imgSrc: ["'self'"],
      connectSrc: ["'self'", process.env.BLOCKCHAIN_RPC],
    }
  },
  referrerPolicy: { policy: 'same-origin' },
  hsts: {
    maxAge: 31536000,
    includeSubDomains: true,
    preload: true
  }
}));
```

Data Encryption

Sensitive Data Handling

```
interface EncryptedData {
  data: string;
  iv: string;
  tag: string;
}

const encryptSensitiveData = async (
  data: any,
  key: Buffer
): Promise<EncryptedData> => {
  const iv = crypto.randomBytes(12);
  const cipher = crypto.createCipheriv('aes-256-gcm', key, iv);

  const encrypted = Buffer.concat([
    cipher.update(JSON.stringify(data)),
    cipher.final()
  ]);

  return {
    data: encrypted.toString('base64'),
```

```
    iv: iv.toString('base64'),
    tag: cipher.getAuthTag().toString('base64')
  };
};
```

Security Monitoring

Audit Logging

```
interface AuditLog {
  action: string;
  actor: {
    id: string;
    type: 'API_KEY' | 'WALLET';
  };
  resource: {
    type: string;
    id: string;
  };
  metadata: Record<string, any>;
  timestamp: Date;
}

const auditLogger = {
  log: async (audit: AuditLog) => {
    await Promise.all([
      // Store in database
      db.collection('audit_logs').insertOne(audit),

      // Send to logging service
      logger.info('Security audit', { audit }),

      // Alert on suspicious activity
      detectSuspiciousActivity(audit)
    ]);
  }
};
```

Security Checklist

Authentication

- [ ] Implement API key rotation
- [ ] Set up wallet signature verification
- [ ] Configure token expiration
- [ ] Implement rate limiting

Authorization

- [ ] Define role permissions
- [ ] Implement access control
- [ ] Set up resource policies
- [ ] Configure audit logging

Blockchain

- [ ] Set up transaction validation
- [ ] Implement multi-sig requirements
- [ ] Configure contract interaction limits
- [ ] Set up transaction monitoring

Data Security

- [ ] Implement encryption at rest
- [ ] Set up secure key management
- [ ] Configure data backup
- [ ] Implement data sanitization

Infrastructure

- [ ] Configure security headers
- [ ] Set up DDOS protection

- [ ] Implement IP filtering
- [ ] Configure SSL/TLS

Monitoring

- [ ] Set up activity logging
- [ ] Configure alerts
- [ ] Implement anomaly detection
- [ ] Set up incident response

Revision #6
Created 25 November 2024 17:34:37 by Aric Fedida
Updated 9 December 2024 00:04:43 by Aric Fedida