Cross-Chain Bridge Testing Documentation

Overview

Auditzer provides comprehensive testing capabilities for cross-chain bridges, supporting multiple protocols and networks. This documentation covers setup, configuration, and testing procedures for various bridge implementations.

Supported Bridge Protocols

LayerZero

- **Networks**: Ethereum, Polygon, Arbitrum, Optimism, Avalanche, BSC
- **Features**: Omnichain applications, cross-chain messaging
- **Security Focus**: Message validation, endpoint security, relayer integrity

Stargate Finance

- **Networks**: Ethereum, Polygon, Arbitrum, Optimism, Avalanche, BSC, Fantom
- **Features**: Liquidity pools, instant finality, unified liquidity
- **Security Focus**: Pool security, slippage protection, MEV resistance

Radiant Capital

- **Networks**: Ethereum, Arbitrum, BSC
- **Features**: Cross-chain lending, unified liquidity
- **Security Focus**: Lending protocol security, cross-chain state consistency

Configuration

Environment Setup

```
# Install dependencies
npm install

# Configure environment variables
cp .env.example .env

# Edit configuration
LAYERZERO_ENDPOINT=https://api.layerzero.network
STARGATE_ENDPOINT=https://api.stargate.finance
RADIANT_ENDPOINT=https://api.radiant.capital

# Network RPC URLs
ETHEREUM_RPC=https://mainnet.infura.io/v3/your-key
POLYGON_RPC=https://polygon-mainnet.infura.io/v3/your-key
ARBITRUM_RPC=https://arb1.arbitrum.io/rpc
```

Bridge Configuration

```
// audityzer.config.js
module.exports = {
  bridges: {
    layerzero: {
      enabled: true,
      networks: ['ethereum', 'polygon', 'arbitrum'],
      endpoints: {
        ethereum: '0x66A71Dcef29A0fFBDBE3c6a460a3B5BC225Cd675',
        polygon: '0x3c2269811836af69497E5F486A85D7316753cf62',
        arbitrum: '0x3c2269811836af69497E5F486A85D7316753cf62'
      }
    },
    stargate: {
      enabled: true,
      networks: ['ethereum', 'polygon', 'arbitrum'],
      pools: {
        ethereum: {
          usdc: '0xdf0770dF86a8034b3EFEf0A1Bb3c889B8332FF56',
          usdt: '0x38EA452219524Bb87e18dE1C24D3bB59510BD783'
        }
      }
    },
    radiant: {
      enabled: true,
      networks: ['ethereum', 'arbitrum'],
      contracts: {
        ethereum: '0x...',
        arbitrum: '0x...'
      }
    }
  }
};
```

Testing Framework

LayerZero Testing

Basic Bridge Test

```
const { LayerZeroTester } = require('audityzer');

describe('LayerZero Bridge Tests', () => {
  let tester;

  beforeEach(async () => {
    tester = new LayerZeroTester({
      sourceChain: 'ethereum',
      targetChain: 'polygon',
      endpoint: process.env.LAYERZERO_ENDPOINT
    });
    await tester.initialize();
  });

  test('should validate cross-chain message delivery', async () => {
    const message = {
      payload: '0x1234567890abcdef',
      dstChainId: 109, // Polygon
      dstAddress: '0x...',
      gasLimit: 200000
    };

    const result = await tester.sendMessage(message);
    expect(result.success).toBe(true);
    expect(result.txHash).toBeDefined();

    // Verify message delivery
    const delivered = await tester.verifyDelivery(result.txHash);
    expect(delivered).toBe(true);
  });

  test('should detect message replay attacks', async () => {
    const message = {
      payload: '0x1234567890abcdef',
      dstChainId: 109,
      dstAddress: '0x...',
      nonce: 1
    };

    // Send original message
    await tester.sendMessage(message);

    // Attempt replay attack
    const replayResult = await tester.sendMessage(message);
    expect(replayResult.success).toBe(false);
    expect(replayResult.error).toContain('nonce already used');
  });
});
```

Advanced Security Tests

```
test('should validate endpoint security', async () => {
  const securityTest = await tester.runSecurityAudit({
    tests: [
      'endpoint_validation',
      'relayer_integrity',
      'message_tampering',
      'gas_griefing'
    ]
  });

  expect(securityTest.vulnerabilities).toHaveLength(0);
  expect(securityTest.score).toBeGreaterThan(90);
});

test('should test gas optimization', async () => {
  const gasTest = await tester.optimizeGas({
    message: '0x1234567890abcdef',
    dstChainId: 109
  });

  expect(gasTest.estimatedGas).toBeLessThan(300000);
  expect(gasTest.optimizations).toBeDefined();
});
```

Stargate Testing

Liquidity Pool Tests

```
const { StargateTester } = require('audityzer');

describe('Stargate Bridge Tests', () => {
  let tester;

  beforeEach(async () => {
    tester = new StargateTester({
      sourceChain: 'ethereum',
      targetChain: 'polygon'
    });
    await tester.initialize();
  });

  test('should validate cross-chain swap', async () => {
    const swap = {
      srcPoolId: 1, // USDC
      dstPoolId: 1, // USDC
      amount: ethers.utils.parseUnits('100', 6),
      minAmountOut: ethers.utils.parseUnits('99', 6),
      dstChainId: 109
    };

    const result = await tester.performSwap(swap);
    expect(result.success).toBe(true);
    expect(result.amountOut).toBeGreaterThan(swap.minAmountOut);
  });

  test('should detect slippage attacks', async () => {
    const slippageTest = await tester.testSlippageProtection({
      poolId: 1,
      amount: ethers.utils.parseUnits('1000000', 6), // Large amount
      maxSlippage: 0.01 // 1%
    });

    expect(slippageTest.protected).toBe(true);
    expect(slippageTest.actualSlippage).toBeLessThan(0.01);
  });
});
```

Radiant Capital Testing

Cross-Chain Lending Tests

```
const { RadiantTester } = require('audityzer');

describe('Radiant Capital Tests', () => {
  let tester;

  beforeEach(async () => {
    tester = new RadiantTester({
      networks: ['ethereum', 'arbitrum']
    });
    await tester.initialize();
  });

  test('should validate cross-chain lending', async () => {
    const lending = {
      asset: 'USDC',
      amount: ethers.utils.parseUnits('1000', 6),
      sourceChain: 'ethereum',
      targetChain: 'arbitrum'
    };

    const result = await tester.crossChainLend(lending);
    expect(result.success).toBe(true);
    expect(result.aTokens).toBeDefined();
  });

  test('should test liquidation mechanics', async () => {
    const liquidationTest = await tester.testLiquidation({
      user: '0x...',
      asset: 'USDC',
      amount: ethers.utils.parseUnits('500', 6)
    });

    expect(liquidationTest.canLiquidate).toBeDefined();
    expect(liquidationTest.healthFactor).toBeDefined();
  });
});
```

Security Testing

Vulnerability Detection

```
const { BridgeSecurityAuditor } = require('audityzer');

describe('Bridge Security Audit', () => {
  test('should detect common bridge vulnerabilities', async () => {
    const auditor = new BridgeSecurityAuditor();

    const audit = await auditor.auditBridge({
      protocol: 'layerzero',
      contracts: ['0x...', '0x...'],
      networks: ['ethereum', 'polygon']
    });

    // Check for specific vulnerabilities
    expect(audit.vulnerabilities).not.toContainEqual(
      expect.objectContaining({ type: 'reentrancy' })
    );
    expect(audit.vulnerabilities).not.toContainEqual(
      expect.objectContaining({ type: 'message_replay' })
    );
    expect(audit.vulnerabilities).not.toContainEqual(
      expect.objectContaining({ type: 'unauthorized_access' })
    );
  });
});
```

Fuzzing Tests

```
test('should perform fuzzing on bridge parameters', async () => {
  const fuzzer = new BridgeFuzzer({
    protocol: 'stargate',
    iterations: 1000
  });

  const results = await fuzzer.fuzzSwapParameters({
    poolIds: [1, 2, 3, 4],
    amounts: 'random',
    slippage: 'random',
    chains: ['ethereum', 'polygon', 'arbitrum']
  });

  expect(results.failures).toHaveLength(0);
  expect(results.coverage).toBeGreaterThan(0.95);
});
```

Performance Testing

Load Testing

```
test('should handle high transaction volume', async () => {
  const loadTester = new BridgeLoadTester({
    protocol: 'layerzero',
    concurrency: 100,
    duration: 60000 // 1 minute
  });

  const results = await loadTester.run({
    transactionType: 'message',
    rateLimit: 10 // TPS
  });

  expect(results.successRate).toBeGreaterThan(0.99);
  expect(results.averageLatency).toBeLessThan(5000); // 5 seconds
});
```

Gas Optimization

```
test('should optimize gas usage', async () => {
  const optimizer = new GasOptimizer({
    protocol: 'stargate'
  });

  const optimization = await optimizer.optimizeSwap({
    amount: ethers.utils.parseUnits('100', 6),
    srcChain: 'ethereum',
    dstChain: 'polygon'
  });

  expect(optimization.gasReduction).toBeGreaterThan(0.1); // 10% reduction
  expect(optimization.estimatedSavings).toBeDefined();
});
```

Monitoring and Alerting

Real-time Monitoring

```
const { BridgeMonitor } = require('auditzer');

const monitor = new BridgeMonitor({
  protocols: ['layerzero', 'stargate', 'radiant'],
  networks: ['ethereum', 'polygon', 'arbitrum'],
  alerting: {
    webhook: 'https://your-webhook-url.com',
    email: 'alerts@yourcompany.com'
  }
});

// Start monitoring
monitor.start();

// Custom alert rules
monitor.addAlert({
  name: 'High Slippage',
  condition: 'slippage > 0.05',
  severity: 'warning'
});

monitor.addAlert({
  name: 'Failed Transaction',
  condition: 'success_rate < 0.95',
  severity: 'critical'
});
```

Metrics Collection

```
// Prometheus metrics
const { BridgeMetrics } = require('auditzer');

const metrics = new BridgeMetrics({
  prometheus: {
    endpoint: 'http://localhost:9090',
    pushGateway: 'http://localhost:9091'
  }
});

// Custom metrics
metrics.registerGauge('bridge_transaction_count', 'Total bridge transactions');
metrics.registerHistogram('bridge_transaction_duration', 'Bridge transaction duration')
;
metrics.registerCounter('bridge_failures', 'Bridge transaction failures');
```

Best Practices

Security

1. **Always validate inputs** before sending cross-chain messages
2. **Use nonces** to prevent replay attacks

3. **Implement timeouts** for cross-chain operations
4. **Monitor gas prices** to prevent griefing attacks
5. **Validate message authenticity** on destination chain

Performance

1. **Batch transactions** when possible to reduce costs
2. **Optimize gas usage** with proper parameter tuning
3. **Use appropriate slippage** settings for swaps
4. **Monitor network congestion** and adjust accordingly
5. **Implement retry logic** for failed transactions

Testing

1. **Test on testnets** before mainnet deployment
2. **Use realistic test data** and scenarios
3. **Include edge cases** in test suites
4. **Perform load testing** under various conditions
5. **Validate cross-chain state** consistency

Troubleshooting

Common Issues

Message Not Delivered

```
// Check message status
const status = await tester.getMessageStatus(txHash);
if (status === 'pending') {
  // Wait for confirmation
  await tester.waitForDelivery(txHash, 300000); // 5 minutes
}
```

High Gas Costs

```
// Optimize gas parameters
const optimized = await tester.optimizeGas({
  message: payload,
  dstChainId: targetChain
});
```

Slippage Too High

```
// Adjust slippage tolerance
const swap = {
  ...originalSwap,
  minAmountOut: calculateMinAmount(amount, 0.02) // 2% slippage
};
```

API Reference

LayerZero Tester

```
class LayerZeroTester {
  constructor(config)
  async initialize()
  async sendMessage(message)
  async verifyDelivery(txHash)
  async runSecurityAudit(options)
  async optimizeGas(params)
}
```

Stargate Tester

```
class StargateTester {
  constructor(config)
  async initialize()
  async performSwap(swap)
  async testSlippageProtection(params)
  async getLiquidity(poolId)
}
```

Radiant Tester

```
class RadiantTester {
  constructor(config)
  async initialize()
  async crossChainLend(lending)
  async testLiquidation(params)
  async getHealthFactor(user)
}
```

Examples

See the examples directory (../tests/e2e/examples/) for complete working examples of bridge testing scenarios.

For more information, visit our documentation site (https://docs.audityzer.com) or join our Discord community (https://discord.gg/audityzer).