SCORM Demos - Proof of Concept Documentation

Overview

This document presents a series of proof-of-concept demonstrations that showcase our technical approach and capabilities for implementing SCORM-compliant content delivery with cross-domain tracking. These demos are designed to validate our proposed architecture and demonstrate key technical competencies without revealing the complete proprietary implementation.

Purpose of These Demonstrations:
- Prove technical feasibility of the proposed solution
- Demonstrate understanding of SCORM standards and LMS integration
- Showcase cross-domain communication capabilities
- Validate security patterns for iframe-based content delivery
- Provide testable, working examples for client evaluation

Important Note: These are intentionally limited proof-of-concept demos. They demonstrate core technical capabilities while the full deliverable will include comprehensive SCORM tracking, advanced features, robust error handling, and production-ready code quality.

Demo 1: SCORM Light Wrapper

What It Demonstrates

Demo 1 showcases a minimal but functional SCORM 1.2 package that embeds external content via iframe and tracks completion across domain boundaries. This demonstrates the foundational architecture for the proposed solution.

Key Capabilities Proven:
- ✅ SCORM 1.2 compliance and LMS integration
- ✅ Cross-domain iframe communication using postMessage API
- ✅ Secure origin validation patterns
- ✅ External content hosted on separate domain (GitHub Pages)
- ✅ Completion tracking from external content to LMS
- ✅ SCORM "API Hunt" pattern implementation

Technical Architecture

The demo implements a two-layer architecture that separates SCORM communication from content delivery:

Layer 1: SCORM Package (Uploaded to LMS)
- `launch.html` - Entry point that embeds external content via iframe
- `scorm_api_proxy.js` - Handles SCORM API discovery and LMS communication
- `imsmanifest.xml` - SCORM 1.2 manifest defining package structure

Layer 2: External Content (Hosted on GitHub Pages)
- `index.html` - The actual learning content displayed to users
- `scorm_bridge.js` - Client-side bridge for postMessage communication
- `styles.css` - UI styling

Communication Flow:
1. LMS launches `launch.html` from SCORM package
2. `launch.html` loads external content in iframe from GitHub Pages
3. External content communicates with SCORM package via postMessage
4. SCORM package relays data to LMS via SCORM API
5. LMS records completion and tracking data

Security Implementation:
- Origin validation using allowlist pattern
- postMessage security best practices
- Cross-domain policy enforcement

How to Test It

Live Demo Access:
- **GitHub Repository:** https://github.com/ericbenong1/scorm-demos
- **External Content (GitHub Pages):** https://ericbenong1.github.io/scorm-demos/demo1_scorm_light/external_content/
- **SCORM Package:** Download `scorm_light_demo.zip` from repository

Testing Instructions:

1. **Option A: Test on ScormCloud (Recommended)**
   - Go to https://cloud.scorm.com/
   - Create free account or log in
   - Upload `demo1_scorm_light/scorm_light_demo.zip`
   - Launch the course
   - Click "Mark Complete" button in the embedded content
   - Verify completion status in ScormCloud dashboard

2. **Option B: Test on Your LMS**
   - Download the SCORM package from the repository
   - Upload to your SCORM 1.2 compatible LMS
   - Launch and test completion tracking

3. **Option C: View External Content Directly**
   - Visit the GitHub Pages URL to see the content interface
   - Note: SCORM functionality requires LMS environment

Expected Behavior:
- Content loads in iframe within SCORM wrapper
- "Mark Complete" button triggers completion signal
- LMS receives and records completion status
- Status updates visible in both content UI and LMS dashboard

What SCORM Elements It Tracks

This demo intentionally implements **limited SCORM tracking** to demonstrate core capability:

Currently Tracked:
- ✅ Lesson Status (incomplete → completed)
- ✅ Session Initialization
- ✅ Session Termination

NOT Tracked (Intentionally Limited):
- ❌ Score/grades
- ❌ Time tracking
- ❌ Suspend/resume data
- ❌ Interaction data
- ❌ Objectives
- ❌ Student progress details

Why This Proves Capability Without Revealing Full Solution

This proof-of-concept demonstrates the **hardest technical challenges**:
1. **SCORM API Integration** - Successfully implements API Hunt pattern and LMS communication
2. **Cross-Domain Architecture** - Proves ability to track content hosted on separate domain
3. **Security Patterns** - Shows understanding of origin validation and secure iframe communication
4. **Real-World Testing** - Successfully deployed and tested on actual SCORM LMS (ScormCloud)

What's Not Included (Reserved for Final Deliverable):
- Comprehensive SCORM data model implementation
- Advanced tracking (time, scores, interactions, objectives)
- Progress persistence and suspend/resume functionality
- Error recovery and offline capability
- Production-grade security hardening
- Performance optimization
- Comprehensive documentation and deployment guides
- Custom integration code for client's specific requirements

This approach proves we can deliver the solution while protecting the intellectual property of the complete implementation.

Relation to Client Requirements

This demo directly addresses the following client needs:

1. **SCORM Compliance** - Validates our ability to create LMS-compatible packages
2. **External Content Hosting** - Proves content can be hosted separately from LMS
3. **Cross-Domain Tracking** - Demonstrates capability to track learning across domains
4. **Production Viability** - Shows working implementation on real SCORM platform

The architecture demonstrated here scales to support the full requirements of the project, including comprehensive tracking, multiple content types, and advanced SCORM features.

Demo 2: postMessage Communication Tutorial

What It Demonstrates

Demo 2 is an interactive educational tutorial that demonstrates the underlying cross-domain communication mechanism used in SCORM Light implementation. This standalone demo isolates and explains the postMessage API patterns that enable secure iframe communication.

Key Capabilities Proven:
- ✅ Cross-domain postMessage communication (parent ↔ child)
- ✅ Origin validation and security patterns
- ✅ Acknowledgment/confirmation patterns
- ✅ Bidirectional communication (two-way messaging)
- ✅ beforeunload handling for critical communications
- ✅ Message structure and data patterns
- ✅ Real-time debugging and message logging

Educational Value:
- Interactive visualization of postMessage mechanics
- Live code examples with explanations
- Security best practices demonstrated in real-time
- Troubleshooting and debugging patterns

Technical Architecture

This demo implements a simplified parent-child communication model that mirrors the SCORM wrapper architecture without the complexity of SCORM API integration.

Component 1: Parent Window (`parent.html`)
- Acts as the "receiver" (similar to SCORM wrapper)
- Listens for messages from embedded iframe
- Validates message origins
- Sends acknowledgments back to child
- Includes configuration panel for origin validation settings
- Displays real-time message log with timestamps

Component 2: Child Window (`child.html` in iframe)
- Acts as the "sender" (similar to external content)
- Sends various message types (ping, greeting, completion, custom)
- Listens for acknowledgments from parent
- Tracks message statistics (sent, received, acknowledged)
- Demonstrates beforeunload handling

Message Structure:

```
{
  type: "ack" | "ping" | "greeting" | "completion" | "custom",
  originalType: "...",  // For acknowledgments
  success: true | false,
  timestamp: "ISO-8601 timestamp",
  message: "Optional custom message"
}
```

Security Features Demonstrated:
1. **Origin Validation**
- Configurable allowlist of trusted origins
- Wildcard option for demo purposes (with security warning)
- Demonstrates both secure and insecure configurations

1. **Acknowledgment Pattern**
   - "Fire and forget" vs. confirmed receipt
   - Prevents data loss from unreliable messages
   - Essential for SCORM completion tracking

2. **beforeunload Handling**
   - Shows limitations of sending data during page close
   - Explains why this is unreliable for critical operations
   - Best practices for session cleanup

Communication Flow:
1. User clicks action button in child iframe (e.g., "Send Ping")
2. Child sends postMessage to parent window with specific target origin
3. Parent receives message and validates origin
4. If valid, parent processes message and logs activity
5. Parent sends acknowledgment back to child
6. Child receives acknowledgment and updates UI
7. Both windows maintain synchronized message logs

How to Test It

Live Demo Access:
- **Direct URL:** https://ericbenong1.github.io/scorm-demos/demo2_postmessage/parent.html
- **GitHub Repository:** https://github.com/ericbenong1/scorm-demos

Interactive Testing Instructions:

1. **Test Basic Communication:**
   - Click "🔵 Send Ping" in the child window
   - Observe message appear in parent's "Received Messages" log
   - Note the acknowledgment received back in child window
   - Verify timestamp and message structure

2. **Test Different Message Types:**
   - Click "👋 Send Greeting" - sends friendly hello message
   - Click "✅ Send Completion" - simulates course completion signal
   - Type custom message and click "📤 Send Custom Message"
   - Observe how each message type is structured and handled

3. **Test Origin Validation (Security):**
   - Check the "Validate Origin" checkbox in parent window
   - Add allowed origin (e.g., `https://ericbenong1.github.io`)
   - Uncheck "Validate Origin" to see security difference warning
   - Note: Currently set to `*` for demo purposes

4. **Test Two-Way Communication:**
   - Scroll down to "Send to Child" section in parent window
   - Type a message in the input field

   - Click “Send Message →” button
   - Observe message received in child window

5. **Explore Educational Content:**
   - Scroll to “Understanding postMessage Security” section
   - Review code examples for origin validation
   - Study the acknowledgment pattern implementation
   - Read about beforeunload handling limitations

Expected Behavior:
- All buttons should successfully send messages
- Parent window should log all received messages with timestamps
- Child window should receive acknowledgments for each sent message
- Message counters should increment correctly
- Two-way communication should work in both directions

Debugging Features:
- Real-time message logs in both windows
- Timestamp tracking for message sequencing
- Message structure displayed in JSON format
- Success/failure indicators
- Origin information displayed for security awareness

Educational Value

This demo serves as both a proof-of-concept and a learning tool, providing:

1. Visual Understanding
- Side-by-side parent-child layout shows communication flow
- Color-coded message types for easy identification
- Real-time logging demonstrates asynchronous nature

2. Code Examples with Explanations
The demo page includes embedded code examples showing:
- ✅ **Good Practice:** Validating origins before processing
- ❌ **Bad Practice:** Accepting messages from any origin
- 🎯 **Target Origin:** Specifying who can receive messages
- 🔄 **Acknowledgment Pattern:** Confirming receipt

3. Security Awareness
- Explains why `event.origin` validation is critical
- Demonstrates the difference between secure and insecure configurations
- Shows real-world attack scenarios (XSS, clickjacking risks)

4. Practical Patterns
- How to structure message payloads
- When to use acknowledgments vs. fire-and-forget
- Handling unreliable communication scenarios
- Best practices for beforeunload events

What It Proves

This demonstration validates our deep understanding of the mechanisms that make SCORM Light implementation possible:

Technical Competency:
1. **postMessage Mastery** - Complete understanding of Web Messaging API
2. **Security Patterns** - Proper implementation of origin validation
3. **Asynchronous Communication** - Handling message timing and reliability
4. **Error Handling** - Graceful degradation and edge case handling
5. **User Experience** - Clear feedback and debugging capabilities

Foundation for SCORM Implementation:
- The acknowledgment pattern shown here is **essential** for reliable SCORM tracking
- Origin validation prevents malicious content from triggering false completions
- Two-way communication enables SCORM wrapper to send data to content (e.g., resume data)
- Message structure design carries over to SCORM data model implementation

Why This Matters for the Project:
- Proves we understand the **underlying technology**, not just copying SCORM examples
- Demonstrates ability to **debug and troubleshoot** cross-domain issues
- Shows commitment to **security best practices**
- Provides a **teaching tool** for client's team to understand the architecture

Relation to Client Requirements

This demo directly supports the following project needs:

1. **Cross-Domain Communication** - Demonstrates the core mechanism used in Demo 1
2. **Security Implementation** - Proves understanding of secure iframe patterns
3. **Reliability Patterns** - Shows acknowledgment mechanisms for critical tracking
4. **Debugging Capability** - Provides tools to diagnose communication issues
5. **Documentation & Training** - Can serve as reference for client's development team

Connection to Demo 1:
- Demo 1 uses the same postMessage patterns shown here
- The SCORM API proxy in Demo 1 implements origin validation like this demo
- Acknowledgment pattern ensures completion signals reach the LMS reliably
- This demo removes SCORM complexity to focus on the communication layer

Scalability:
- The patterns demonstrated here support not just completion tracking but:
- Progress updates (percent complete, pages viewed)
- Score reporting (quiz results, assessment data)
- Suspend data (bookmarking, state persistence)
- Time tracking (session duration, time spent)
- Custom interactions (any bidirectional data exchange)

Demo 3: Token Security Demo

What It Demonstrates

Demo 3 showcases **secure token-based launch architecture** for SCORM Light implementations. This demo simulates how an LMS would generate time-limited, signed tokens to prevent unauthorized access to course content while maintaining a seamless user experience.

Key Capabilities Proven:
- ✅ Token generation with user/course binding
- ✅ Time-limited token expiration (5 minutes demo, configurable)
- ✅ HMAC-like signature verification (simulated)
- ✅ Token validation before content access
- ✅ Secure rejection of invalid/expired tokens
- ✅ User-friendly error handling and messaging
- ✅ Professional UX with real-time token status

Security Concepts Demonstrated:
- **User Binding:** Tokens tied to specific user_id and course_id
- **Time Limitation:** Tokens expire to prevent replay attacks
- **Signature Verification:** Detects token tampering
- **Session Binding:** Tokens linked to session identifiers
- **Access Control:** Content inaccessible without valid token

Technical Architecture

This demo implements a three-page flow that simulates the complete token lifecycle from generation through validation to rejection:

Page 1: LMS Launch Simulator (`launcher.html`)
- Simulates the LMS "Launch" button action
- Accepts user_id and course_id inputs
- Generates secure token with payload and signature
- Displays token structure for educational purposes
- Redirects to content page with token parameter

Page 2: Protected Content (`content.html`)
- Validates token before displaying content
- Checks signature integrity
- Verifies token hasn't expired
- Displays user/course information from token
- Shows real-time countdown of token expiration
- Implements color-coded warnings (green → yellow → red)

Page 3: Access Denied (`rejected.html`)
- Displayed when token validation fails
- Shows specific error reason (expired, invalid, missing, tampered)
- Provides user-friendly explanations
- Offers troubleshooting guidance
- Includes technical details panel for debugging

Supporting Files:
- `token-utils.js` - Core token generation and validation library
- `styles.css` - Professional, accessible styling across all pages
- `README.md` - Comprehensive documentation with production guidelines

Token Structure:

```json
{
  "payload": {
    "user_id": "user_12345",
    "course_id": "course_scorm_light_101",
    "issued_at": 1738368000000,
    "expires_at": 1738368300000,
    "session_id": "sess_abc123xyz_1738368000000"
  },
  "signature": "a7f3c9e1b2d4f6a8c5e7d9f1b3a5c7e9..."
}
```

Validation Process:
1. **Extract Token:** Retrieve from URL parameter
2. **Decode:** Parse base64-encoded token
3. **Verify Signature:** Recalculate and compare signatures
4. **Check Expiration:** Validate current time < expires_at
5. **Validate Structure:** Ensure all required fields present
6. **Grant/Deny Access:** Display content or redirect to error

How to Test It

Local Testing (Recommended):

1. **Open the Demo:**
   - Navigate to `demo3_token_security/launcher.html`
   - Open in any modern web browser

2. **Test Valid Token Flow:**
   - Enter User ID (or use default: `user_12345`)
   - Enter Course ID (or use default: `course_scorm_light_101`)
   - Click "🚀 Launch Content" button
   - Observe token generation (2-second delay for visibility)
   - Automatically redirected to content page
   - View validated user information and token details
   - Watch real-time countdown of token expiration

3. **Test Token Expiration:**
   - **Option A:** Click "🧪 Test Expired Token" button on content page
   - **Option B:** Wait 5 minutes and see automatic expiration
   - Observe redirect to rejection page with expiration message

4. **Test Invalid Token:**
   - Access `content.html` with manually modified token in URL
   - Example: `content.html?token=invalid_token_xyz`
   - Observe "Invalid signature" error on rejection page

5. **Test Missing Token:**
   - Access `content.html` directly without token parameter
   - Observe "Token missing" error on rejection page

GitHub Pages Testing (After Deployment):
- Demo will be available at: `https://[username].github.io/scorm-demos/demo3_token_security/launcher.html`
- All functionality works client-side (no server required)

Expected Behaviors:

Test Scenario	Expected Result
Valid token within 5 minutes	Content displays with user info
Token expires naturally	Auto-redirect to rejection page
Click “Test Expired” button	Immediate redirect to rejection page
Manually modify token	“Invalid signature” error
Access content without token	“Token missing” error
Real-time countdown	Time changes color: green → yellow → red

⚠️ Important: JavaScript Simulation

This demo uses JavaScript for proof-of-concept only!

This is explicitly a **conceptual demonstration** to show understanding of token-based security for the proposal. The implementation simulates what would happen server-side in production.

Why JavaScript for the Demo:
- GitHub Pages doesn’t support server-side languages (PHP/Node.js/Python)
- Demonstrates the complete token lifecycle visually
- Shows the security concepts end-to-end
- Appropriate for proposal validation purposes
- Allows interactive testing without server infrastructure

Production Implementation Requirements:

In a **real SCORM Light implementation**, token operations MUST be server-side:

❌ Never Do This in Production:

```
// INSECURE - Secret key exposed to client
const SECRET_KEY = 'my_secret_key_12345';
const token = generateToken(userId, SECRET_KEY);
```

✅ Production Implementation (PHP Example):

```
// lms_launch.php (Server-Side Token Generation)
session_start();

// Verify user authentication
if (!isset($_SESSION['user_id'])) {
    die('Not authenticated');
}

$user_id = $_SESSION['user_id'];
$course_id = $_GET['course_id'];

// Check database for enrollment
$db = new Database();
if (!$db->isUserEnrolled($user_id, $course_id)) {
    die('User not enrolled in course');
}

// Generate secure token
$secret_key = getenv('TOKEN_SECRET_KEY'); // From environment
$issued_at = time();
$expires_at = $issued_at + (5 * 60); // 5 minutes

$payload = json_encode([
    'user_id' => $user_id,
    'course_id' => $course_id,
    'issued_at' => $issued_at,
    'expires_at' => $expires_at,
    'session_id' => session_id()
]);

// Sign with HMAC-SHA256
$signature = hash_hmac('sha256', $payload, $secret_key);
$token = base64_encode(json_encode([
    'payload' => json_decode($payload, true),
    'signature' => $signature
]));

// Store token hash in database
$token_hash = hash('sha256', $token);
$db->storeToken($token_hash, $user_id, $expires_at);

// Redirect to content
header("Location: content.php?token=" . urlencode($token));
exit;
```

```
// content.php (Server-Side Token Validation)
$token = $_GET['token'] ?? '';

// Validate token
$token_data = json_decode(base64_decode($token), true);
$payload = $token_data['payload'];
$signature = $token_data['signature'];

// Verify signature
$secret_key = getenv('TOKEN_SECRET_KEY');
$expected_sig = hash_hmac('sha256', json_encode($payload), $secret_key);

if (!hash_equals($expected_sig, $signature)) {
    header('Location: rejected.php?reason=invalid_signature');
    exit;
}

// Check expiration
if (time() > $payload['expires_at']) {
    header('Location: rejected.php?reason=expired');
    exit;
}

// Verify token in database
$db = new Database();
if (!$db->tokenExists(hash('sha256', $token))) {
    header('Location: rejected.php?reason=not_found');
    exit;
}

// Check session binding
session_start();
if (session_id() !== $payload['session_id']) {
    header('Location: rejected.php?reason=session_mismatch');
    exit;
}

// All checks passed - load content
include 'scorm_content.php';
```

What It Proves

This demonstration validates our understanding of critical security architecture for SCORM implementations:

Security Architecture Competency:
1. **Token-Based Authentication** - Prevents unauthorized direct access to content
2. **Time-Limited Sessions** - Protects against link sharing and replay attacks
3. **Cryptographic Signing** - Detects token tampering and forgery attempts
4. **Session Binding** - Links tokens to active user sessions
5. **Graceful Error Handling** - User-friendly messaging for security failures

Production Readiness Awareness:
- Clear documentation that JavaScript simulation is for demo only
- Comprehensive production implementation guidelines included
- Demonstrates understanding of client vs. server security boundaries
- Shows awareness of common security pitfalls and how to avoid them

User Experience Design:
- Clean, professional interface across all pages
- Real-time feedback on token status
- Color-coded warnings for approaching expiration
- Helpful error messages guide users to proper launch procedure
- Mobile-responsive design

Educational Value

This demo serves multiple purposes:

1. Security Awareness Training
- Shows WHY token security is necessary (prevents URL sharing)
- Demonstrates HOW tokens work (generation → validation → expiration)
- Explains WHEN tokens should be used (every content launch)

2. Technical Architecture Demonstration
- Illustrates complete token lifecycle
- Shows integration points with LMS
- Demonstrates error handling patterns

3. Implementation Guidance
- README.md includes production PHP examples
- Clear separation of demo vs. production requirements
- Best practices for token management

4. Client Confidence Building
- Proves deep understanding of security principles
- Shows commitment to production-quality security
- Demonstrates ability to communicate technical concepts clearly

Relation to Client Requirements

This demo addresses critical security concerns for the SCORM Light implementation:

1. Unauthorized Access Prevention
- Users cannot share direct links to bypass LMS authentication
- Content requires valid token from LMS launch
- Expired tokens automatically rejected

2. Session Security
- Tokens bound to user sessions prevent hijacking
- Time limits reduce window of vulnerability
- Signature verification prevents token forgery

3. Audit and Compliance
- Token validation creates security audit trail
- Failed access attempts can be logged
- User/course binding enables tracking

4. Production Scalability
- Token pattern scales to thousands of concurrent users
- Stateless validation (signature check) reduces server load
- Database token storage enables additional verification layers

Integration with Demo 1 (SCORM Light Wrapper):

In the complete implementation, token security integrates with the SCORM wrapper:

```
┌──────────────────┐
│  LMS (PHP)       │
│  1. Authenticate │
│  2. Generate     │
│     Token        │
└────────┬─────────┘
         │
         ↓ Redirect with token
┌──────────────────────┐
│  launch.html         │  ← SCORM Package
│  (with token param)  │
│  3. Validate token   │
│  4. If valid, load   │
│     external iframe  │
└────────┬─────────────┘
         │
         ↓ Load if token valid
┌──────────────────────┐
│  External Content    │
│  (GitHub Pages/CDN)  │
└──────────────────────┘
```

Security Flow:
1. User clicks “Launch” in LMS
2. LMS generates token (server-side PHP)
3. LMS redirects to SCORM package with token
4. SCORM package validates token (server-side PHP)
5. If valid, SCORM package loads external content in iframe
6. If invalid, redirects to rejection page
7. Token expires after set time (5-15 minutes)

Additional Security Layers (Production):
- Database token storage and verification
- Rate limiting on token generation
- IP address binding (optional)
- Device fingerprinting (optional)
- Logging and monitoring of failed attempts
- Automatic token revocation on logout

Testing Checklist

Use this checklist to verify all demo functionality:

- [ ] **Valid token flow works**
- [ ] Token generates successfully
- [ ] Content displays after validation
- [ ] User information shows correctly
- [ ] Token expiration countdown displays
- [ ] **Expiration handling works**
- [ ] “Test Expired Token” button redirects properly

- [ ] Natural expiration after 5 minutes works
- [ ] Expiration error message is clear
- [ ] **Invalid token handling works**
- [ ] Modified token shows “Invalid signature” error
- [ ] Corrupted token shows appropriate error
- [ ] Error page displays helpful information
- [ ] **Missing token handling works**
- [ ] Direct content access without token redirects
- [ ] Error message guides user to launcher
- [ ] **UI/UX quality**
- [ ] All pages styled consistently
- [ ] Mobile responsive design works
- [ ] Color coding (green/yellow/red) works
- [ ] Buttons and forms function properly
- [ ] **Educational content**
- [ ] Production notes clearly explain server-side requirement
- [ ] PHP examples are present and accurate
- [ ] Security warnings are prominent
- [ ] README.md is comprehensive

Conclusion

These proof-of-concept demonstrations validate our technical approach and capability to deliver the proposed SCORM solution. Demo 1 proves we can successfully implement the core architectural pattern - cross-domain content delivery with SCORM tracking - which forms the foundation for the complete solution.

We’re ready to proceed with full implementation based on your specific requirements and feedback on these demonstrations.

Questions or Testing Issues?
Please contact us if you encounter any issues testing the demos or have questions about the technical approach. We’re happy to provide additional clarification or demonstrations as needed.

Repository: https://github.com/ericbenong1/scorm-demos
Last Updated: January 31, 2026