Advanced Distributed Learning (ADL)

Cross-Domain Scripting Issue

Version: 1.0

October 13, 2003

Available at:
www.adlnet.org

This page intentionally left blank.

FOREWORD

This document defines and presents several solutions to the cross-domain scripting security issue with a Sharable Content Object (SCO)'s ECMAScript (i.e. JavaScript) access to a Sharable Content Object Reference Model (SCORM®) Application Program Interface (API) Instance. It is technical in nature and is meant for IT specialists and learning management system (LMS) vendors. The solutions herein were submitted by or derived from members of the ADL Community. See Appendix C for additional documentation on the Cross-Domain Scripting Issue.

During the development and publication of this document, the ADL Community submitted several additional cross-domain solutions. These solutions will be evaluated and tested for inclusion in a future version of this document.

This document may be modified or superceded as work proceeds. Comments, suggestions and alternative solutions for inclusion in a future version of this document are welcome.

This page intentionally left blank.

COPYRIGHT

DISTRIBUTION

Permission to distribute this document is granted under the following conditions:

1. The use of this document, its images and examples is for non-commercial, educational or informational purposes only.
2. The document, its images and examples are intact, complete and unmodified. The complete cover page, as well as the COPYRIGHT, DISTRIBUTION and REPRODUCTION sections are consequently included.

REPRODUCTION

Permission to reproduce this document completely or in part is granted under the following conditions:

1. The reproduction is for non-commercial, educational or informational purposes only.
2. Appropriate citation of the source document is used as follows:
    a. Source: Advanced Distributed Learning (ADL), Cross-Domain Scripting Issue Paper, October 2003. Available at http://www.adlnet.org.

For additional information or questions regarding copyright, distribution and reproduction, contact:

ADL Co-Laboratory
1901 North Beauregard Street
Alexandria, VA 22311
USA
(703) 575-2000

This page intentionally left blank.

Key ADL Technical Team Contributors to the ADL Cross-Domain Scripting Issue Paper:

William Capone
Clark Christensen
Philip Dodds
Jeff Falls
Dexter Fletcher
Matthew Handwork
Rob Harrity
Sue Herald
Alan Hoberney
Paul Jesukiewicz
Kirk Johnson
Mary Krauland
Jeff Krinock
Lori Morealli
Angelo Panar
Douglas Peterson
Jonathan Poltrack
Betsy Spigarelli
Schawn Thropp
Bryce Walat
Jerry West

Key ADL Community Contributors to ADL Cross-Domain Scripting Issue Paper:

Judy Brown
Ed Cohen
Jonathan Dean
Jeffrey Engelbrecht
Lenny Greenberg
Albert Ip
Boyd Nielsen
Michael Norberg
Claude Ostyn
Tyde Richards
Paul Roberts
Robby Robson
Eric Rosen
Steve Slosser
Roger St. Pierre
Brian Taliesin
John Toews
Jeff Webb

…and many others.

ADL would also like to thank the ADL Community for their contributions to the ADL Cross-Domain Scripting Issue Paper.

This page intentionally left blank.

TABLE OF CONTENTS

This page intentionally left blank.

1.0 INTRODUCTION

This concept paper is being provided to the ADL Community to clarify browser security issues pertaining to a Sharable Content Object (SCO)'s communication through a Sharable Content Object Reference Model (SCORM®) Application Program Interface (API) Instance. In the SCORM model, SCOs communicate with an LMS using ECMAScript via an LMS-provided API Instance. When a launched SCO and its LMS-provided API Instance are hosted on different domains, browser security restrictions prevent the ECMAScript API calls, thus prohibiting communication. Through collaboration with several members of the ADL Community and internal prototyping efforts, the ADL Technical Team has tested several different solutions to this problem and is providing them to the ADL Community at large. This paper details the Cross-Domain Issue and presents several known and tested solutions.

1.1 Description

In the SCORM Run-Time Environment (RTE), an LMS launches content objects by way of Hypertext Transfer Protocol (HTTP) in a Web browser. Typically, launched content resides in a frameset provided by the LMS or in a browser window opened by the LMS. A SCO that wishes to communicate with the LMS must find an LMS-provided API Instance. The SCORM requires that the API Instance reside in a predefined location in the browser's Document Object Model (DOM) hierarchy or window opener's hierarchy. Once the API Instance is located, the SCO is free to invoke calls on a set of predefined methods (the SCORM API) that provide execution management, state management and data transfer.

If the LMS-provided API Instance and the SCO originate from the same domains and have the same protocol, e.g. https, there are no issues. To clarify, a "hostname" is an ASCII string (e.g. "xyz.abc.com"), which consists of a local part ("xyz") and a domain name ("abc.com"). A group of computers whose hostnames share a common suffix have the same "domain name" and are not susceptible to the SCORM cross-domain scripting issue. For instance, if an LMS server and a content server are both located in the "abc.com" domain or are actually the same server as illustrated in Figure 1.1a, there are no security issues when the content attempts to use the API Instance. All API methods described in the SCORM are available without security concerns.

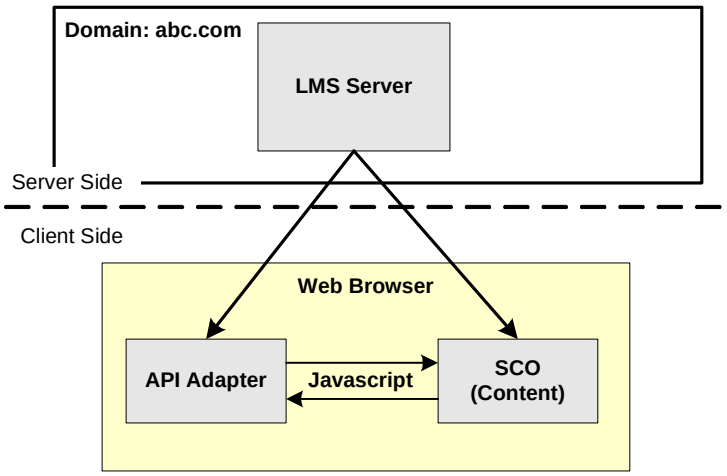

Figure 1.1a – LMS and content server on same domain

On the other hand, if the content server is located in a different domain than the LMS-provided API Instance, an ECMAScript security exception will result when methods are invoked on the API Instance. The ECMAScript security exception occurs for viable and documented security reasons. Historically, programmers with malicious intent could steal sensitive information from users' browsers with ECMAScript. This was possible due to security holes in the browser. Developers at the time recognized the vulnerability that cross-domain scripting could create if it was permitted. In response to this vulnerability, the "Same Origin Policy" was established. The policy defines an "origin" as the combination of a document's protocol and domain. Documents are restricted from accessing another document's DOM if the other document has a different "origin." It is important to note that this restriction is a deliberate feature, not a bug. Figure 1.1b illustrates a case where a security violation would occur when a SCO attempts to use the LMS-provided API Instance.

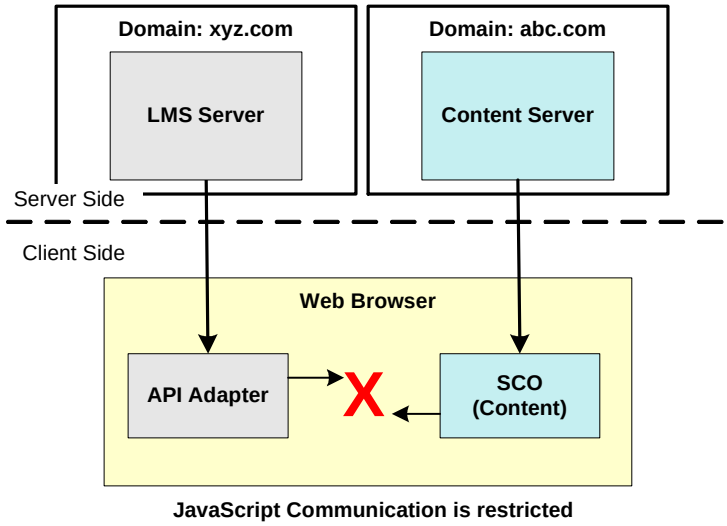

Figure 1.1b – LMS and Content server on different domains

The fact that cross-domain scripting causes a security violation may directly affect SCORM implementation. The remainder of this paper will discuss several cross-domain scripting scenarios and solutions. Each scenario can be solved in multiple ways, so there is no one-size-fits-all solution. SCORM implementers will need to evaluate their specific environment and content deployment needs to select or develop an appropriate solution.

2.0 SCORM API OVERVIEW

The SCORM Run-Time Environment Model requires that SCOs communicate with the LMS via ECMAScript through the SCORM API. This requirement, in certain content deployment scenarios, results in the cross-domain scripting issue. The following section provides an overview of the SCORM API for SCO to LMS communication. The information is taken from the SCORM Version 1.2, available at ADLNet.org. All information presented in this section references the SCORM Version 1.2; however, it also applies to the SCORM Version 1.3.

The use of a common API fulfills many of the SCORM's high-level requirements for interoperability and reuse. It provides a standardized way for SCOs to communicate with LMSs,

yet it shields the particular communication implementation from the content developer. In its simplest terms, an API is merely as set of predefined methods that the SCO can rely on being available. An API hides implementation details from SCOs and thus enables reuse and interoperability.

The SCORM Version 1.2 defines a SCO's responsibility in regards to the API as follows:

SCOs must be launched in a browser window that is a child window or a child frame of the LMS-provided window that contains the API Instance. It is the responsibility of the SCO to establish communication with the LMS by finding the API Instance and invoking the LMSInitialize("") API method. Once communication with the LMS is established, the SCO may invoke other API methods as desired until the SCO wishes to end communication with the LMS by invoking the LMSFinish("") API method.

The SCORM Version 1.2 details the following requirements for LMSs related to the API:

- An API Instance must be provided by the LMS.
- The only supported mechanism for API interaction from SCOs is through ECMAScript calls.
- The API Instance must be accessible via the DOM as an object named "API".

As described above, the LMS is responsible for providing an API Instance and a SCO is responsible for establishing communication with the LMS through that API Instance. When the API Instance and the SCO originate from the same domain, the communication is straightforward. However, in certain systems, the SCO may not originate from the same domain as the API Instance, thus the "Same Origin Policy" mentioned earlier comes into effect.

3.0 CROSS-DOMAIN SCRIPTING SCENARIOS

The following scenarios detail multiple configurations where the SCORM is in use today. Some of these scenarios are vulnerable to the cross-domain scripting issue. Recommended solutions for each scenario are listed in Section 4.0. This is not a comprehensive list, but is generalized to encompass the majority of situations in use today. Section 4.0 will detail the ADL prototyped solutions submitted by the ADL Community and provide insight on which solution to use based on the following generic scenarios.

3.1 One LMS/Content Server

A typical SCORM environment used today involves one server that executes the LMS software and houses the SCORM content. This environment is typically not susceptible to the cross-domain issue. Figure 3.1a depicts the One LMS/Content Server scenario. In this particular illustration, there is no cross-domain scripting issue.

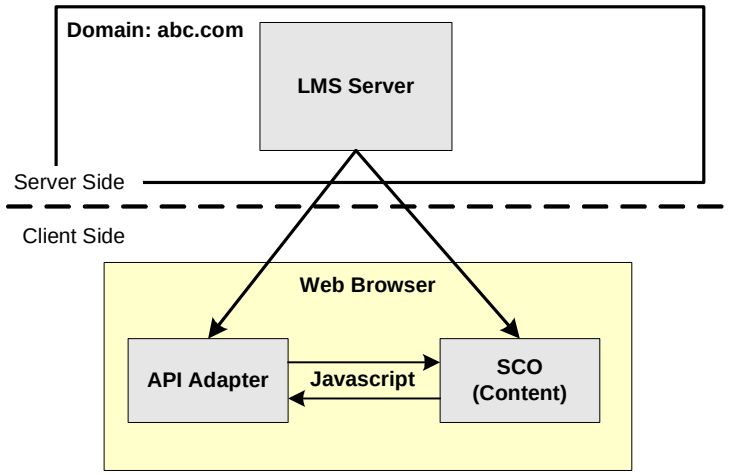

Figure 3.1a – One LMS/Content server with no virtual IP addresses

3.2 Separate LMS Server and Content Server

Another scenario characteristic of many SCORM environments in use today deals with separate LMS and content servers. Many times, separate LMS and content servers are used due to bandwidth concerns, digital rights management, SCO maintenance or security issues. For example, a content server may be configured to run a specific application server (e.g., a JavaServer Pages (JSP) Application Server). For this reason, a learning network may be set up so that the LMS server runs only the LMS software and is not bogged down with the burden of compiling JSPs to Servlets, then Servlets to byte code. Separating the content server and the LMS server, in this case, frees up the LMS server to handle many students while pushing the dynamic content delivery to a content server, thus freeing up bandwidth and speeding up end users' learning experience. This state of affairs can cause a cross-domain security restriction. Figure 3.2a illustrates the LMS Server and Content Server scenario.

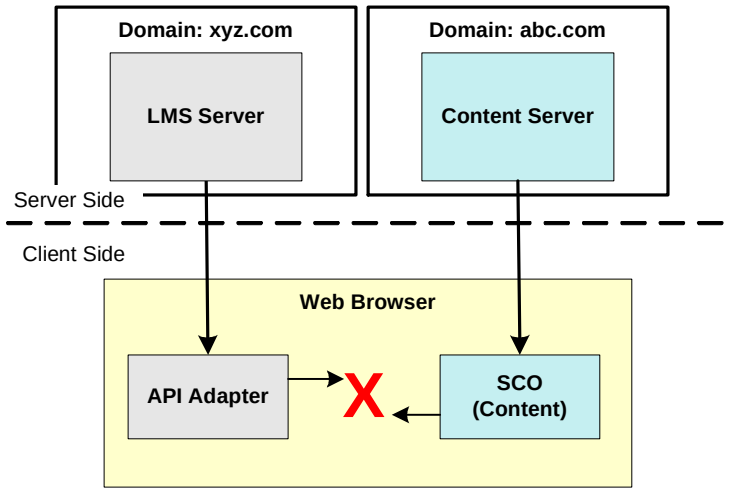

Figure 3.2a – Separate LMS server and Content Server

3.3 Central LMS Server and Distributed Content Servers

Yet another scenario characteristic of many SCORM environments employs several content servers managed by a centralized LMS. This scenario is similar to the environment discussed in Section 3.2. However, several other complications may arise due to the variety of network architectures used to create this environment. For example, the content servers may be located on secure networks behind firewalls. Also, the network of distributed content servers may not be "known" by the LMS prior to "course import." Both of these additional constraints will assist in determining the solution that best fits an individual situation. Figure 3.3a illustrates the generic scenario without regard to any additional constraints.

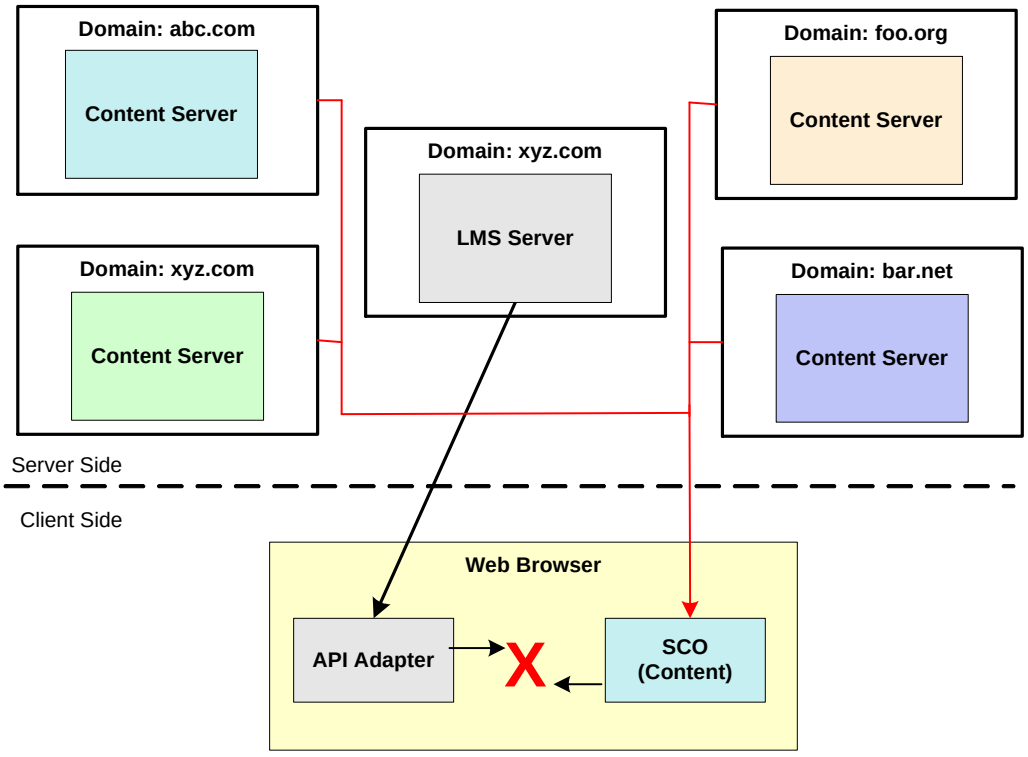

Figure 3.3a – Central LMS with distributed content servers

The scenario depicted above is common in learning networks where end users may be using the LMS and its courses while behind a firewall. This architecture is often put in place to meet certain restrictions defined in the Department of Defense's mobile code policy. This policy defines mobile code as "software obtained from remote systems outside the enclave boundary, transferred across a network, and then downloaded and executed on a local system without explicit installation or execution by a recipient." Many secure networks filter out mobile code before it can be executed in a user's browser. So, if content served from a remote content server located outside a network's firewall contains mobile code, this code is filtered out by the network security measures, which may render the content useless or incomplete.

Many times, content is developed using code that is restricted by the mobile code policy, but is known to be safe. To allow this mobile code to execute properly in a SCORM environment, several content servers are set up behind each firewall. These content servers are mirrors, each holding the same content as the others. When a user accesses the LMS and selects content for

delivery, the LMS server determines which network the user is accessing the LMS from and then delivers content from a content server behind the same firewall as the user. This allows mobile code embedded in content to be executed by the user instead of being filtered out by a firewall (e.g. Microsoft ActiveX Components).

3.4 Several LMS Servers and Distributed Content Servers

When one considers the concept of reusability, another variation of a scenario involving a network of distributed content servers surfaces. In this case, the same distributed content server network is presented. Furthermore, the additional constraints considered in Section 3.3 also apply to this scenario. The characteristic that makes this scenario unique is the consideration that multiple LMSs may utilize the same content servers. For example, some content depends on specific application servers to execute correctly. Several content developers host their content on in-house servers, configured specifically to execute the developers' content. Many of these courses may be general enough that they can be used in multiple arenas and consequently may be accessed by several LMSs. Figure 3.4a depicts two LMS servers that host content from a distributed network of content servers.

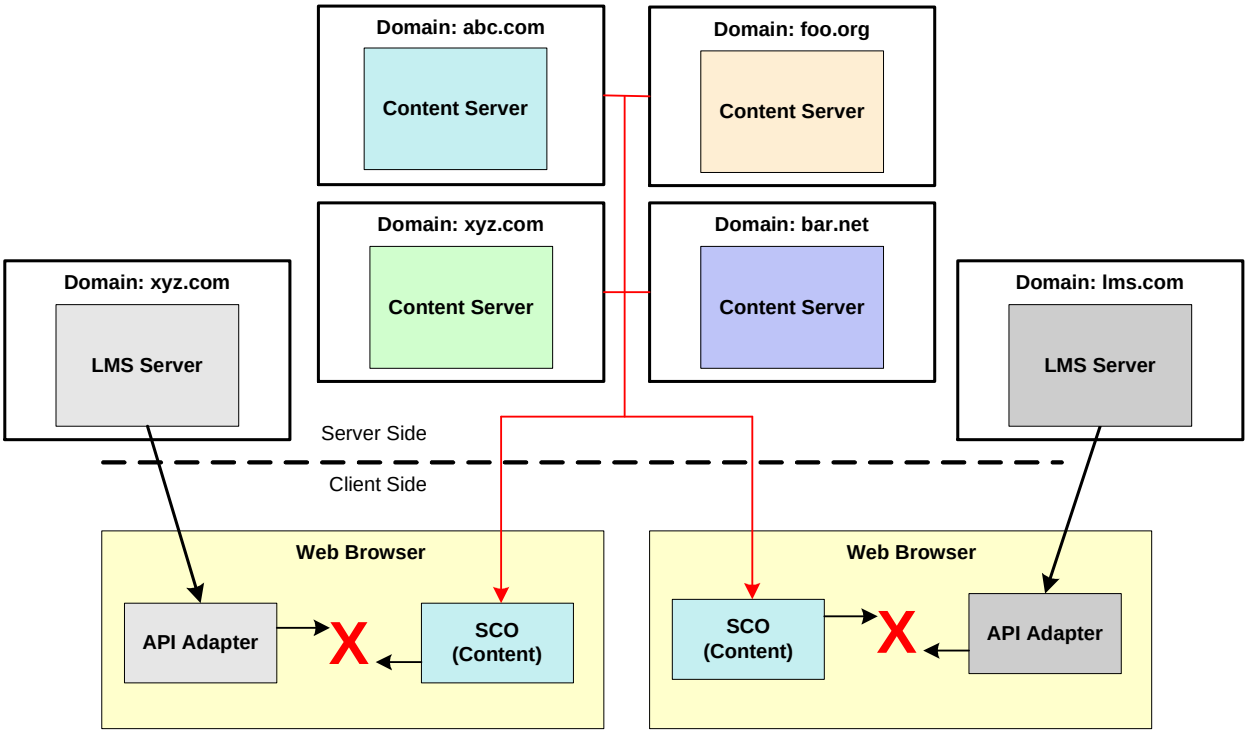

Figure 3.4a – Several LMS servers and Distributed Content servers

Again, this scenario does not account for aspects such as firewalls. If this architecture applies, several solutions can be selected, but any additional constraints should be considered before a final cross-domain scripting solution is selected.

4.0 CROSS-DOMAIN SCRIPTING SOLUTIONS

This section details a number of solutions to the cross-domain scripting issue. Each subsection includes a list of pros and cons to be considered before selecting that particular solution. Some

solutions may not adhere to a specific organization's IT policies, while others may require an undesired end user intervention to experience learning content. All of the solutions below should be evaluated and the "best fit" should be selected. The pros and cons are not intended to be a complete list, but an overview of some of the complications that may occur during deployment of a specific solution. To assist in the selection of a specific solution, the generic scenarios presented in Section 3.0 are considered below. The ADL Technical Team is also accepting comments and suggestions for possible inclusion in an update to this document. All members of the ADL Community are encouraged to submit solutions or additional issues through the Help & Info Center on ADLNet.org. The solutions listed below are not to be considered a comprehensive list, but rather a subset of possible solutions based on the practices that are most commonly used today. There is no significance of the order of the following solutions. They are presented in an arbitrary order.

4.1 Locate Content on LMS Server or in LMS's Domain

Locating the content (or at a minimum, the SCOs) on the LMS server or in the LMS server's domain will avoid the cross-domain scripting problem. If this type of configuration is acceptable and feasible for a specific situation, this solution is lightweight and fairly easy to implement.

Applicable Generic Scenario(s)

- Section 3.1 One LMS/Content Server

In the cases where the learning network is small enough to be scaled to one server, this is the ideal solution. By configuring the Web server to run from one domain, the issue can be solved. When content is "imported" into the LMS, it will communicate with the LMS without any security concerns.

Pros

- No end-user intervention is required.
- No special modifications to the content or the content package are required.
- No special modifications to the LMS are required.
- Lightweight and easy to implement.

Cons

- Applicable to learning networks that can house the LMS and all content from one domain.
- LMS and/or content delivery processing may slow due to all network traffic hitting one server.

4.2 SCO-Fetcher

The SCO-Fetcher cross-domain scripting solution was submitted by Albert Ip and Ric Canale in their paper, *Single Copy Re-use of Sharable Content Objects*[1]. They cover several scenarios in which SCOs are hosted by a mix of LMSs, single Content Management Systems (CMSs) and multiple distributed CMSs. There are several implementation specific ways to develop this cross-domain solution, but they can all be described in a general manner.

During content delivery, the LMS typically opens content in a child frame or newly opened window by changing the document.location.href of the window to the launch location of the learning resource. The SCO Fetcher solution modifies this approach by introducing a server-side LMS component (the SCO-Fetcher).

When a content server that is located in a different domain than the LMS hosts a SCO, a server-side module (the SCO-Fetcher) fetches the SCO from the content server and delivers the SCO to the Web browser. The location of the SCO-Fetcher URL is in the LMS's domain. The SCO's href value from the imsmanifest.xml file becomes a parameter in a query string appended to the URL of the SCO-Fetcher. Effectively, it is the same as appending a local path to the beginning of the href of the SCO (which is hosted by the LMS), albeit with a slight syntactic difference. The result of this approach is that for the browser, the SCO appears to come from the LMS's domain. There is no cross-domain scripting issue for the browser because the SCO and API Instance both appear to come from the same domain.

In general, this solution is deployed as follows:

1. The LMS identifies a URL associated with the SCO for delivery.
2. If the SCO is hosted locally by the LMS, the normal LMS operation is followed.
3. If the SCO is hosted by a content server, the SCO-Fetcher, which is hosted by the LMS, uses the remote SCO's location as a parameter to fetch the SCO from the content server.
4. Future versions of the SCO-Fetcher may intercept the HTTP Response Object and send back to the content server.

Figure 4.2a illustrates the basic architecture of this solution.

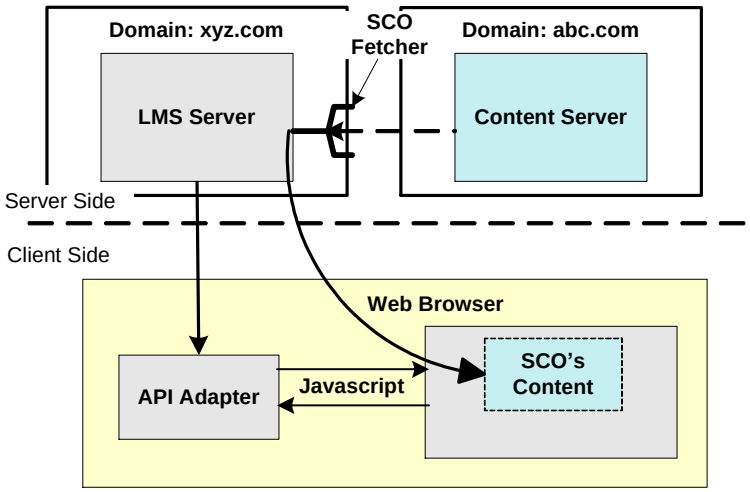

Figure 4.2a – SCO Fetcher Diagram

Applicable Generic Scenario(s)

- Section 3.2 Separate LMS and Content Server
- Section 3.3 Central LMS and Distributed Content Servers

- Section 3.4 Several LMS Servers and Distributed Content Servers

In the case where a learning network is too large to be hosted from a single domain, the SCO Fetcher solution can be effective. Even more importantly, some LMS systems are "unaware" of the location of the content that they deliver until run-time. If a specific SCORM environment contains "unknown" content servers, regardless of the generic scenario (Sections 3.2 through 3.4), this solution is effective.

Pros

- No end-user intervention is required.
- No special modifications to the content or the content package are required.
- Handles the cases where the content servers cannot be specially configured before they can be used by an LMS.
- Fairly straightforward and easy to implement and deploy.

Cons

- LMS modification is required to implement the SCO-Fetcher.
- Possible run-time performance issues may result in some implementations of the SCO-Fetcher.

For additional details on the applicability of this solution, see the *Single Copy Re-use of Sharable Content Objects*[1] paper.

4.3 Proxy Configuration (Virtual Server)

Another solution to the cross-domain scripting issue requires the use of a proxy or virtual server. This solution was submitted by Claude Ostyn in his white paper, *SCORM content delivery issues: Content from a host other than the LMS host*[2]. In this solution, a virtual server or proxy can be configured to redirect the HTML Response from the LMS and the content server. Each server (LMS and content) would physically reside in their respective domains, but would appear to a browser as being the same server. This solution may be set up as illustrated in Figure 4.3a.

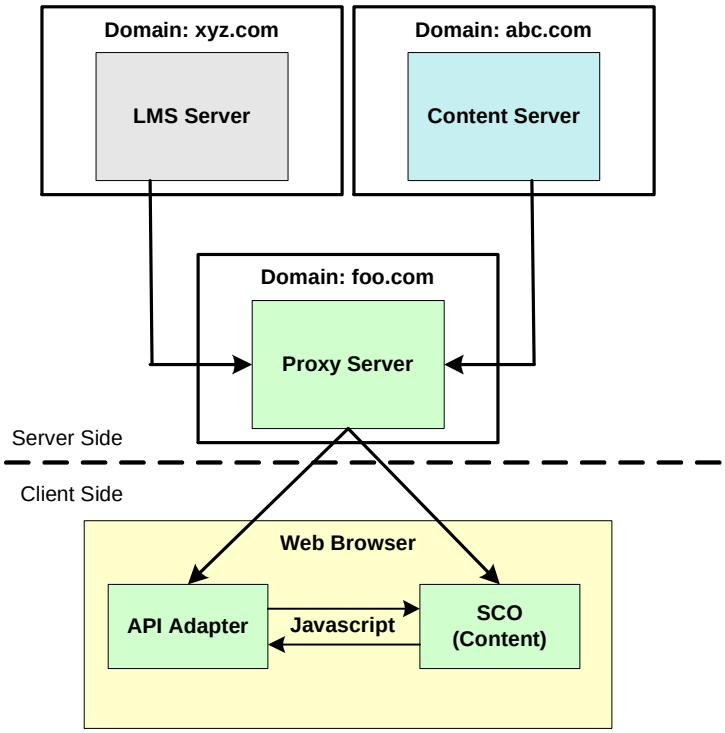

Figure 4.3a – Proxy server redirects LMS and content

It is important to note that the diagram above depicts only one way that this solution may be deployed. For instance, the LMS server itself could serve as the proxy server and the additional proxy server illustrated above may not be needed.

Applicable Generic Scenario(s)

- Section 3.2 Separate LMS and Content Server
- Section 3.3 Central LMS and Distributed Content Servers
- Section 3.4 Several LMS Servers and Distributed Content Servers

The proxy solution is effective in all of the generic scenarios involving more that one server. To configure the proxy solution for the expected results, the full network of servers in use must be known, unless a generic redirection formula is used. A rule must be defined in the proxy configuration for each server, or a generic rule may be used to automatically remap URLs that contain a specific "trigger" string. If each server is configured manually, adding or removing a content server to the configuration requires updating the proxy configuration file. If the remapping is formula-based, it is possible to use the proxy to redirect requests to unintended servers. In either case, the proxy logs will show every request and additional filtering can be added to prevent unintended use.

However, this solution may require additional steps for certain types of deployment. For example, in Section 3.3, a use case is described where content servers are mirrored behind multiple firewalls and are managed by a single LMS server outside these specific firewall zones. The content servers are distributed in order to avoid serving content from outside the firewall, which would filter out code in violation of the mobile code policy. Each of these content servers

has its own domain. In this case, a proxy redirector would be required inside each firewall zone, to redirect content requests to the content server inside the firewall. Otherwise, the user would get the content through a proxy that was outside the firewall, thus solving the cross-domain issue, but may disrupt the learning experience because the firewall may strip the content of needed mobile code.

Pros

- No end-user intervention is required.
- No special modifications to the content or the content package are required.
- The same proxy server can be used for multiple content servers.
- The solution is easy to configure and deploy.
- Proxy logs can be used to monitor and audit cross-domain requests.

Cons

- Depending on the approach used, may require upfront proxy configuration specific the LMS and all content servers that the LMS utilizes.
- May not work with a LMS implementation that does not lend itself to being hidden behind a proxy server, unless the LMS server and the proxy server are in effect the same.
- May experience performance issues due to content traffic through the proxy server.
- Firewalls and other security measures can complicate the solution.

For additional details on the applicability of this solution, see the *SCORM content delivery issues: Content from a host other than the LMS host*[2] white paper.

4.4 URL Rewrite

The URL Rewrite solution is the specific implementation of the proxy server model described above, applied to known Web servers such as Microsoft IIS or the Apache Web Server. It was submitted by Claude Ostyn in his white paper, *SCORM content delivery issues: Content from a host other than the LMS host*[2]. In this solution, custom or commercial utilities and/or features built into the Web servers can be used to rewrite URLs before they are presented in a Web browser. The prototype effort utilized two of these tools, ISAPI_Rewrite for Microsoft IIS and Apache mod_rewrite/mod_proxy for the Apache HTTP server. These tools required configuration rules that utilize regular expressions to remove a content object's domain and replace it with the LMS domain. See Appendix D for example configuration files for ISAPI Rewrite and mod_rewrite/mod_proxy. It is important to note that the tools listed above are not the only tools available on the market. They were selected arbitrarily and both resulted in a successful remedy to the cross-domain issue. It is likely that many other URL rewriting tools exist for the aforementioned Web servers and others not mentioned above.

Applicable Generic Scenario(s)

- Section 3.1 One LMS/Content Server
- Section 3.2 Separate LMS and Content Server
- Section 3.3 Central LMS and Distributed Content Servers
- Section 3.4 Several LMS Servers and Distributed Content Servers

Scenario variations

- Specific rules for specific content or content server addresses
- Generic rules triggered by a pattern in the URL.

The URL Rewrite solution applies to all of the generic scenarios in Section 3.0. It is effective in solving the cross-domain issue on simple and complicated networks where the content servers may not be known ahead of time. The main issue with this resolution is that the required configuration is not usually understood by application level developers, because it is at the infrastructure level. In addition, URL rewriting may not be acceptable to a specific IT department due to control and/or security issues that arise when put in place.

Pros

- No end-user intervention is required.
- No special modifications to the content or the content package are required.
- Can be "locked" down to work with specific content and/or specific servers; alternatively, can be configured to use generic rules that require no configuration updates when target content or target content servers change.
- Works on complicated and large learning networks where the full network may not be configured ahead of time.

Cons

- Some tools have an associated expensive purchasing and/or support cost.
- May be complicated and difficult to deploy as part of the LMS system; this is infrastructure, not application level configuration.
- If generic rules are used, potential for unauthorized relaying through the proxy server requires monitoring and/or additional security configuration.
- If generic rules are used, the LMS launch mechanism must massage the SCO URL into a form that will trigger the rule whenever a SCO that must be served buy a "foreign" server is launched; this requires the LMS and the proxy rules to follow the same conventions.

For additional details on the applicability of this solution, see the *SCORM content delivery issues: Content from a host other than the LMS host*[2] white paper.

4.5 Manipulation of document.domain

Another solution popular with the ADL Community today deals with setting the “domain” property of the HTML document hierarchy (document.domain). This solution requires that the end user view the content in Microsoft Internet Explorer (IE). As opposed to IE, Netscape and other popular Web browsers do not allow the domain property to be set. This is due to the W3C DOM Level 2 HTML specification[3], which states that document.domain is “read only.” For this resolution to work, the content’s domain must be set to the document.domain property of the LMS or vice versa.

Applicable Generic Scenario(s)

- Section 3.1 One LMS/Content Server
- Section 3.2 Separate LMS and Content Server
- Section 3.3 Central LMS and Distributed Content Servers
- Section 3.4 Several LMS Servers and Distributed Content Servers

This solution applies to all of the generic scenarios, but certain restrictions must be considered. First, the use of Microsoft IE must be mandated. Also, there are constraints in Internet Explorer that restrict this solution to a very specific use case. The document.domain property can only be set if the content and LMS only differ by a second level domain name. For example, Figure 4.5a illustrates a case where the content and LMS are both in the "xyz.com" domain. The content server contains a second level domain name "abc". In this case, the second level domain name associated with the content would cause a cross-domain issue. If IE was mandated for a learning network, and the network configuration was similar to the illustration above, this solution would be acceptable.

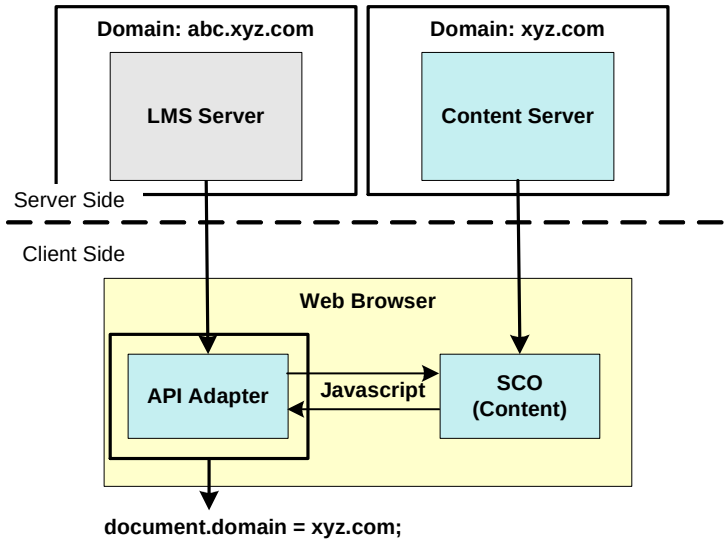

Figure 4.5a – document.domain solution

Pros

- No end-user intervention is required.
- No special configuration of the content or content server is required.

Cons

- This solution works only in older versions of Microsoft IE. It no longer works when the current security patches are applied to IE 6.0.
- LMS server must be modified to determine and set the domain of the content.
- This solution does not adhere to W3C's DOM Level 2 HTML standard.
- All content can only have a different second-level domain name. The primary domain must be identical.

This solution is not recommended because it depends on a brittle browser feature that is no longer supported by current versions. It will work in a specific case; however, it is advisable to use caution if deploying this solution. In addition to the "Cons" listed above, it is likely that new versions of IE will address the non-conformance of their implementation of document.domain to the W3C recommendation. This solution may be deprecated as the Internet Explorer browser mature.

For more information on the document.domain solution, see the *About Cross-Frame Scripting and Security*[4] Microsoft Developer Network (MSDN) article and/or the *Resolving Cross-Domain Security Issues in SCORM*[5] paper by Harvi Singh and Prashant Bhardwaj.

4.6 SCO URL Callback Technique

The SCO URL Callback Technique was submitted to ADL by Paul Roberts, *Questionmark*, August 27, 2003. This solution was originally applied to Perception, a Web server-based assessment system that can be used to conduct a test then return results to the LMS. The solution has been generalized in this paper to encompass any content provided by any type of content server residing in a different domain than the LMS.

In the SCO URL Callback Technique, communication between the LMS and the content server is via an HTML wrapper page that resides on the same server (and same domain) as the LMS. The LMS would normally open a new browser window when calling the wrapper to start the SCO, but it may also be contained in a child frame of the LMS.

Directly after being launched, this wrapper uses the JavaScript API to get any SCO information from the LMS (via the SCORM API) it needs to launch the SCO then creates an HTTP call to the content server with the required parameters in the query string. This SCO is then launched in a new window. The URL of the SCO contains a parameter that is the location of the wrapper used to launch the SCO. As the SCO executes, it may desire to set data. As data is set, parameters corresponding to these events are created and appended to the wrapper URL. When the SCO is ready to commit data to the LMS, an HTTP Request containing the wrapper's URL and newly added parameters is sent back to the original wrapper window. The wrapper uses the parameters sent via HTTP from the SCO to invoke the SCORM API methods needed to track the events that occurred during that SCO's session (LMSSetValue).

Applicable Generic Scenario(s)

- Section 3.1 One LMS/Content Server
- Section 3.2 Separate LMS and Content Server
- Section 3.3 Central LMS and Distributed Content Servers
- Section 3.4 Several LMS Servers and Distributed Content Servers

This solution is effective in all of the generic scenarios. It handles cases where the content servers are not known by the LMS ahead of time and works well where content may be launched by multiple LMS systems. This solution is similar to the solution presented in the next section

with the exception that the additional development complexity is required during content and content package implementation in contrast to the LMS as in the next section.

Pros

- No end-user intervention is required.
- No special modifications to the LMS are needed.
- Handles cases where the complete learning network is not known ahead of time.

Cons

- Requires content to be developed or modified to run with the LMS-served wrapper. Content must be configured to communicate via HTTP to the wrapper based on a parameter received during launch.
- Can be complicated and time consuming to implement or modify content.
- Only data values that are small enough to fit in a URL query string can be handled. There are the practical limitations on the length of a URL (which are poorly documented and vary by browser and server vendors). Also, the locator part of the URL may itself be quite long. For example, it does not seem wise to use this technique to transfer 4096 characters of suspend data.
- If content needs to get and set values many times during a learning session, additional complexity needs to be considered during implementation. The solution assumes that the SCO will need to get values, once, directly after launch and that it will set values, once, as it finishes its session.

4.7 Signed Java Applet Solution

The Signed Java Applet Solution was submitted by Claude Ostyn in his white paper, *SCORM content delivery issues: Content from a host other than the LMS host*[2]. This solution requires that the content server provide a signed applet that can then talk to the LMS even if the content is coming from another host. The sequence works like this:
The LMS determines it needs content. It launches a frameset that is specific to the LMS, but that is actually served by the content server (or appears to be served by the content server). This frameset contains the client side run-time component of the LMS, including a signed Applet that will be allowed by the browser to communicate back with the LMS across host boundaries.

Applicable Generic Scenario(s)

- Section 3.1 One LMS/Content Server
- Section 3.2 Separate LMS and Content Server
- Section 3.3 Central LMS and Distributed Content Servers
- Section 3.4 Several LMS Servers and Distributed Content Servers

The Signed Java Applet Solution applies to all of the generic scenarios. The following list details the pros and cons of this solution.

Pros

- No network configuration is required.

- No content or content package modifications required.

Cons

- Requires special relationship with content server that will serve the LMS specific part of the content, e.g. LMS vendor must be allowed to copy and update this content on the content vendor's host.
- Requires user intervention to accept the Applet as "trusted."
- Requires a compatible Java environment that trusts the certificate for the applet is available and enabled on the client.

4.8 Run-Time Service on Content Server

The Run-Time Service on Content Server solution was submitted by Jeff Engelbrecht in his article, *SCORM Deployment Issues in an Enterprise Distributed Learning Architecture* [6]. The solution is similar in nature to the Signed Java Applet Solution discussed in the last section.

In most cases, it is assumed that the LMS server hosts the API from its own domain. This, in many cases, is the root cause of the cross-domain scripting issue. In many scenarios, LMS vendors are aware that content launched by their systems resides on a definitive set of known content servers. If this is the case, an LMS provided Run-Time Service (RTS) containing the API Instance could be supplied to the content vendor for installation on their content servers. This creates an environment where the content and API Instance are located in the same domain, thus eliminating any cross-domain issues.

The second aspect of this solution requires additional LMS to content server and content server to LMS communication. An internal ADL prototype effort in conjunction with lessons learned from the ADL Community led to some general guidelines listed in this section. Note that these guidelines are not specified by the SCORM and are not intended for use as a SCORM Conformance measure in any way. The flow chart depicted in Figure 4.6a presents the entire process by which a piece of content is delivered, tracked and terminated.

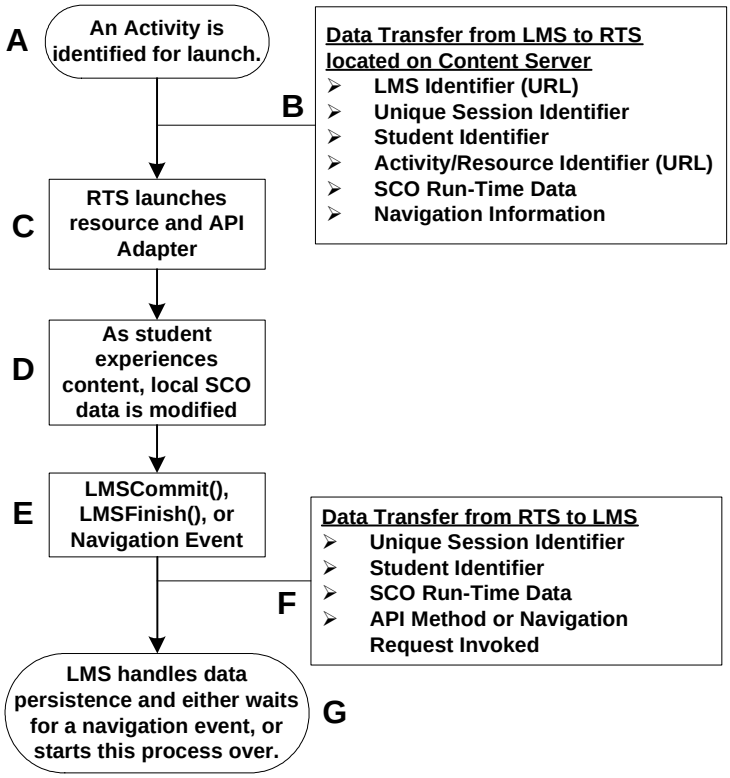

Figure 4.6a – Content Server RTS Flow Chart

The following list adds additional details needed to implement this solution. Several communication mechanisms discussed are outside the scope of the SCORM and can be selected based on the restrictions of an individual environment.

A. The LMS identifies an activity for launch. The way in which the activity is identified varies from SCORM 1.2 to SCORM 1.3. However, this is out of scope for the cross-domain issue. It is only important that an activity or, more importantly, its associated resource or launch location is identified.

B. The LMS must now notify the LMS service located on the content server that it should launch the learning resource. This communication mechanism is outside the scope of the SCORM. This cross-domain scripting research project uncovered multiple communication mechanisms used by the ADL Community today. See Appendix B for a brief description of each communication mechanism.

C. An RTS component provided by the LMS obtains both the API Instance and the content from the content server's domain, thus eliminating any cross-domain scripting issues.

D. The student experiences the content in the web-browser. Data may be set (LMSSetValue() or SetValue()) in a cached copy of the SCO Run-Time Data.

E. Some event occurs that requires communication with the LMS back-end. The set of events that require RTS to LMS communication include LMSCommit(“”), LMSFinish(“”) and any navigation event in the SCORM Version 1.2 and Commit(“”), Terminate(“”) and any navigation event in the SCORM Version 1.3.

F. To persist data, terminate a communication session, or trigger a navigation request, the RTS must communicate back to the LMS. As in step “B”, this communication mechanism is

outside the SCORM of the SCORM. See Appendix B for a brief description of each communication mechanism.

G. The LMS receives the data sent by the RTS and stores any data to persistent storage. In addition, the LMS may evaluate a navigation request and start this process again.

Figure 4.2b illustrates how this process may occur in an environment where an LMS is configured to communicate with a content server outside its own domain. The letter identifiers (A, B, C, etc) correspond directly to the last example.

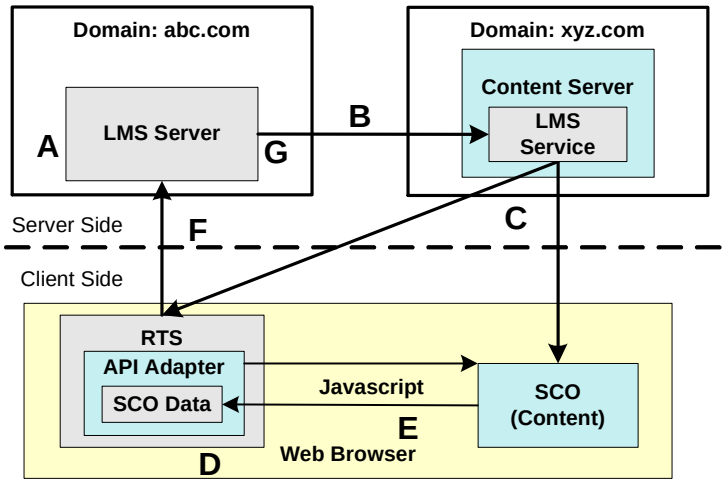

Figure 4.2b – RTS Solution General Architecture

Applicable Generic Scenario(s)

- Section 3.1 One LMS/Content Server
- Section 3.2 Separate LMS and Content Server
- Section 3.3 Central LMS and Distributed Content Servers
- Section 3.4 Several LMS Servers and Distributed Content Servers

In the case where the learning network is small enough to be scaled to one server, this is most likely an overcomplicated solution; however, it will work. In all other cases, this solution is extremely effective. It is also an excellent solution where security is a main concern. This is due to the communication mechanism between the LMS and RTS. An organization can select a communication mechanism that is acceptable to the organization's security policies.

Pros

- No end-user intervention is required.
- No special modifications to the content or the content package are required.

Cons

- Requires the content vendor and LMS vendor to establish a relationship to install and configure the LMS RTS on all content servers.
- Requires LMS modification so that the API is part of a RTS installed on the content servers.
- Can be complicated and time-consuming to implement and deploy.

For additional details on the applicability of this solution, see the *Problems of Implementing SCORM in an Enterprise Distributed Learning Architecture*[6] paper.

APPENDIX A: ACRONYM LIST

ADL	Advanced Distributed Learning
API	Application Program Interface
ASP	Active Server Pages
ASCII	American Standard Code for Information Interchange
CMS	Content Management System
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IE	Internet Explorer
IIS	Internet Information Services
IT	Information Technology
JSP	JavaServer Pages
LMS	Learning Management System
MSDN	Microsoft Developer Network
RTE	Run-Time Environment
RTS	Run-Time Service
SCO	Sharable Content Object
SCORM	Sharable Content Object Reference Model
URL	Universal Resource Locator
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

APPENDIX B: RTS TO LMS COMMUNICATION

The Solution described in Section 4.6, Run-Time Service (RTS) on the content server requires an alternative communication mechanism between the RTS located on the content server and the LMS. As stated in Section 4.6, this communication is outside the scope of the SCORM. However, some general guidance was presented including the minimum information that must be passed at different times during course execution. This solution was gathered from several in the ADL Community. Each vendor that provided this solution used a different mechanism for this communication. Each communication mechanism is described below. When implementing this cross-domain scripting resolution, a communication mechanism acceptable to the organization's IT department's policies can be selected from the list below or any other means acceptable for specific organization can be used.

Hypertext Transfer Protocol (HTTP)
Several organizations developed the proprietary communication described in Section 4.6 of this document with HTTP. HTTP is one of the protocols that powers the World Wide Web. Typically, when information needs to be transferred between the RTS and the LMS or vice versa, a package is created. This package contains text that corresponds to the information needed by the LMS or RTS. After the package is created, a "Request" is "Posted" to a URL corresponding to the LMS or RTS. A server-side component is used to read in the Request and parse out the relevant information. More information on HTTP can be found at http://www.w3.org/Protocols/.

Web Services
Web services provide a standard means of communication among different software applications involved in presenting dynamic context-driven information to the user. The communication mechanism described in section 4.6 can be implemented with a Web Service approach. The scenario requires a Web Service on the LMS and the RTS that both are listening for an XML message from the other. In order to promote interoperability and extensibility among these applications, as well as to allow them to be combined in order to perform more complex operations, a standard reference architecture is needed. The Web Services Architecture Working Group at W3C is tasked with producing this reference architecture. More information can be found at http://www.w3.org/2002/ws/.

Sockets
The SCORM Sample Run-Time Environment, available at http://www.adlnet.org is a free software product that implements the SCORM-imposed requirements of an LMS system. The Sample RTE implements the SCORM API as a Java Applet. The Applet is a client-side component that resides in the end-users browser. In order for it to communicate with the server-side LMS component, a communication mechanism needed to be established between the client and server sides. Sockets were selected for this communication mechanism. Although this is not the same problem as described in section 4.6, the general approach is still the same. Sockets can be opened and communicated through in many programming languages, in many different ways.

See the source code that ships with the Sample RTE for an example of how to use sockets for this communication. Even using sockets, communication from a client to a server in a domain that is not the same that served the client requires that the Java Applet be signed, and that the user accept the certificate before the communication can take place.

APPENDIX C: REFERENCES

1. Ip, A & Canale, R., *Single Copy Re-use of Sharable Content Objects*. 2003. Available at: http://koala.dls.au.com/scorm/

2. Ostyn, C., *SCORM content delivery issues: Content from a host other than the LMS host*. December 3, 2002.

3. W3C, *Document Object Model (DOM) Level 2 HTML Specification, Version 1.0*. Available at: http://www.w3.org/TR/DOM-Level-3-HTML/

4. MSDN, *About Cross-Frame Scripting and Security*. 2003. Available at: http://msdn.microsoft.com/library/default.asp?url=/workshop/author/om/xframe_scripting_security.asp)

5. Bhardwaj, P & Singh, H., *Resolving Cross-Domain Security Issues in SCORM*. Available at: http://www.centra.com/products/belearning/scormwhitepaper.pdf

6. Engelbrecht, J., *SCORM Deployment Issues in an Enterprise Distributed Learning Architecture*. *The eLearning Developers' Journal*, Feb 18, 2003. Available at: http://www.elearningguild.com/pdf/2/021803MGT-H.pdf

7. Kraan, W., *A feature or a bug; SCORM and cross domain scripting*. *The Centre for Educational Technology Interoperability Standards (CETIS)*. June 22, 2003 Available at: http://www.cetis.ac.uk/content/20030622203659/printArticle

8. Robson, R., *E-learning Meets Security Policies*. E-learning Magazine. February/March 2003

APPENDIX D: EXAMPLE CONFIGURATION FILES

The following code is an excerpt taken from the `httpd.ini` file that controls ISAPI Rewrite, a commercial utility for Microsoft IIS server, as described in Section 4.4 of this document. It was submitted for inclusion in this paper by Claude Ostyn. This is not an endorsement of ISAPI Rewrite, but is included as an example to guide your own research. You can find more information at www.isapirewrite.com.

```
# 3600 = 1 hour
CacheClockRate 3600

RepeatLimit 32

# Block external access to the httpd.ini and httpd.parse.errors files
RewriteRule /httpd(?:\.ini|\.parse\.errors).* [F,I,O]
# Block external access to the Helper ISAPI Extension
RewriteRule .*\.isrwhlp [F,I,O]

# Test proxy
RewriteCond URL (.*)
RewriteHeader Old-URL: ^$ $1

# The following assumes that the IIS server name is "lms"
# Redirect using generic ~p~ pattern
# For example http://lms/~p~/foobar.com/foo/bar/sco.htm will
# return content to the browser from
#   http://foobar.com/foo/bar/sco.htm
# as if it came from
#   http://lms/~p~/foobar.com/foo/bar/sco.htm
RewriteProxy (.*)~p~/(.*) http\://$2 [F,U]

# Redirect using a specific pattern tied to a specific server
# For example http://lms/saranjan/portugal.htm will
# return content to the browser from
#   http://saranjan.com/portugal.htm
# as if it came from
#   http://lms/saranjan/portugal.htm
RewriteProxy /saranjan(.*) http\://saranjan.com$1 [F,I,U]
```

The following code is an excerpt taken from the httpd.conf file that controls the Apache HTTP Server. Note that mod_rewrite and mod_proxy are both Apache features that are disabled by default. The Apache http.conf file needs to be configured to enable these features before the following code will work. This code was submitted for inclusion in this paper by Claude Ostyn. This is not an endorsement of Apache HTTP Server, but is included as an example to guide your own research.

```
# Security
# See Apache documentation for the various ways the server can be
# secured. This is generic web server administration though.
# The following should be customized by web server admin. based
# on local rules
# <Proxy *>
#     Order deny,allow
#    Deny from all
#    Allow from all
# </Proxy>

RewriteEngine on

#CRITICAL minimal security.
#Security to prevent free ride e.g. proxy:http://www.example.com
RewriteRule ^proxy:.*  -  [F]

# Redirect via proxy to a specific remote server through this server.
# through a sentinel directory name used to trigger the rule, as in
# http://<host>/snj/sitemap.htm
# The RedirectMatch is not required, but necessary if you want to
# handle the situation where
# IE won't automatically add a trailing slash to directory name,
# e.g., http://www/somestuff -- which leaves the proxied
# pages full of slashless (and therefore broken) URLs.
RedirectMatch ^/snj$       http://www.saranjan.com/

# ProxyPass by itself makes the browser aware of the redirect and does
# not work for SCORM
ProxyPass /snj/ http://www.saranjan.com/
# But adding ProxyPassReverse to ProxyPass works!
ProxyPassReverse /snj/ http://www.saranjan.com/

# But what if we want to use a generic proxy trigger in the URL, e.g.
# a URL in the form http://<host>/~p~/www.foo.com/something.htm?
# The directive below works!
# Note that the following will *not* if the URL ends up with a directory
# name that is not followed by a slash, in the hope of finding the
# default file. It will seem to work but the links will not be followed,
# e.g. graphics won't show, because this will be handled as a redirect
# rather than a proxy. To solve this, ensure that the URL
# includes either a correct file name, or ends with a slash, or
# add a rule to do this massaging as in the RedirectMatch example above.

RewriteRule (.*)~p~/(.*) http\://$2 [P,L]
##### End Claude's proxy rules
```