SCORM Content Vulnerability Workarounds

04/02/2009

Jonathan Poltrack

Problem Solutions (PS) LLC

Introduction

This whitepaper describes possible workarounds for security concerns with the SCORM Run-Time API. Due to the nature of ECMAScript (JavaScript) and the current and past SCORM Run-Time API, it is not possible to totally secure access to client-side functions. A user can *cheat* the system by sending scores, progress status, and other writeable data model element values to the LMS. A final solution to this vulnerability will require an update to the IEEE standards used by SCORM, or the addition of a security layer to the API, thus solving this issue for all new content. However, a significant quantity of existing content would remain susceptible to *cheating* via rouge API calls. A way to secure legacy content even after a final solution is developed is still required. The requirements below were considered during the development of these workarounds. The SCORM content vulnerability workaround must:

- Ensure that any SCORM Run-Time Data Model values set and committed to the LMS are accurate per the content developer's original intent.
- Provide indicators about whether cheating or unauthorized API calls have occurred.
- Secure the vast majority of legacy content (both SCORM 2004 and SCORM 1.2).
- Not result in changes to the underlying standards or specifications.
- Be simple and lightweight from a content developer's perspective.

Workaround Descriptions

The workarounds described here do not prevent the possibility of unauthorized API access. However, they do ensure that appropriate values are stored in the vast majority of existing content. SCORM allows for individual SCORM data model elements to be set "n" times during a learner session. For example, a SCO could potentially set a score (`cmi.score.scaled`) to several different values as the learner progresses through the SCO. The only value that is used by the LMS is the final value set by the SCO before the SCO terminates (`Terminate()`). The final value is then used for sequencing evaluations and potentially learner records. If a cheat is used to set a score during a SCO's communication session, it has no effect unless it's the final value set immediately before the SCO terminates.

The "Cache it, Set It, Terminate It" Principle

Currently, content developers should consider caching (within the SCO) important data model values (ex: scores. completion and success statuses, etc) until the SCO is about to terminate. SCORM Run-Time Data model elements should be set at the last potential opportunity, immediately before calling `Terminate()`. This is more likely to create a situation where the intended values are set as the immediate last step before communication between the SCO and the LMS terminates. Any unauthorized API calls after `Terminate()` will be ignored and any associated data model calls will not affect data persisted by the LMS.

This principle does not work in all cases. For example, `cmi.progress_measure` might be set, potentially several times, as a learner progresses through a SCO. In this case, the best option may be to determine if any unexpected data model elements have been set.

Identifying Unauthorized API Access

Sometimes it is important to detect if an unauthorized API violation has occurred even if the correct values are maintained by an LMS. There are several methods of determining whether there was an attempt to cheat a SCO. The following list details some characteristics that can be used to determine unauthorized API use:

- Hashing expected values (see below)
- Comparing local cached values versus `GetValue()` calls to the LMS
- Getting data model values from the LMS that should not have been set by the SCO
- Identifying an unrealistic (shortened) amount of time spent experiencing a SCO

Hashing Expected Values

To determine if a value was intentionally set by the SCO, it may be useful to create hashed values for elements intentionally set by the SCO. The following example describes a possible flow through a SCO using hashing to identify unauthorized API use:

1. SCO is launched and initialized
2. User experiences content and takes an exam
3. SCO sets a `cmi.score.scaled` value via `SetValue()`
4. SCO uses the score value to create a value calculated by a hash function
5. User cheats the API to set a score of 100%
6. SCO begins to terminate
    - Before `Terminate()`, SCO gets the score value from the LMS, enters it into the hash function and compares the value versus the hashed value in step #4
    - If the hashed values do not correspond, then the value was changed by the user or some unauthorized process

After a situation as described above occurs, the SCO may reset any unexpected values and set another data model element to indicate that unauthorized API access has occurred. The use of this flag is proprietary and may not be used consistently across all LMSs.

Making it Easier

There are potential API wrapper modifications that could be used in conjunction with legacy content to assist in making a future solution backward compatible. It is possible to build this caching approach as well as verification via hashing into the API wrapper in a consistent manner to be used with many SCOs. For example, the following changes may assist in identifying unauthorized API access as well as using *the "Cache it, Set It, Terminate It" Principle*.

- `SetValue()` / `LMSSetValue()` – Extend to include local arrays used by the SCO to maintain cached values for data model elements set by the SCO, the values of the data model elements and/or hashed values for each `SetValue()` call
- `GetValue()` / `LMSGetValue()` – Extend to verify that values returned by the LMS result in the same hashed value if the element was set earlier in the same SCO
- `Terminate()` / `LMSFinish()` – Extend to set cached versions of data model elements used during the SCO. Extend to verify that values set previously in the SCO were maintained and not updated by an external process

Conclusion

The workarounds described in this document can be used to ensure that expected values are maintained by an LMS even if a learner or tool *cheats* the SCO with unauthorized API calls. In addition, a SCO can use the methods described here to determine if any data model elements were set via unauthorized API calls. This information can then be used to notify the system manager or content owner of the possibility of *cheating*. These workarounds are not intended as a final solution. An effort must be put forth to solve this issue with secure technologies in a future version of the SCORM. However, these approaches can help ensure that the persisted data model values associated with a learner attempt on a SCO are those that were intended by the content developer.