Preventing Score Forgery with Input Verification: Slick

Alec Thomson Alan Huang Chris Bille Casey McNamara

Abstract

This document assumes familiarity with the Flixel verifier documentation, and won't go into motivation and analysis already detailed there. Read that first.

Having made a score verifier for the Flixel game engine, we attempted to make one for the Java-based Slick game engine as well. The Slick verifier follows the same general strategy as the Flixel verifier: a client records its actions and submits them to the server along with its score, and the server reruns the game with the client's actions to verify the score's validity.

1 Design

Like the Flixel verification system, the Slick verification system is split into a client library for recording games and communicating with the server and a server-side program to verify the recordings. Both of these are sufficiently general to be used in various games.

1.1 Client Library

Unlike Flixel, Slick does not have a built-in replay recorder. Therefore we implemented a library that the game client must use to record every input event it handles. For developers adopting this system, this would involve some fairly small changes to game code – adding a line to log the input event at the start of each input handler method.

The recording library keeps track of the time when the game started and records the elapsed time since game start whenever it logs an input event. Time may be less reliable than frame number, but some games may make decisions about what an input does based on time rather than frame number, so it's unclear. Probably our system as it stands should not be used in games with precise time dependence, as discussed in the Limitations section of the Flixel documentation.

A logged input event looks like:

```
<Method>
<name>(name of handler method)</name>
<time>(timestamp of method call)</time>
<(parameter name)>parameter value</(parameter name>
(possibly more parameters)
</Method>
```

Given this log, we can replay the game calling the handler methods at the correct times to simulate the same input the user provided.

For our client library, we created a new class called "GameLogger" that performs recording and communication with the server. At the start of the game, the game calls "GameLogger.start()" which will start counting time. Whenever the game handles an input event, it calls "GameLogger.log(methodName, parameters...)" to add the event to the log. When

a score is obtained, the game calls "GameLogger.post(scoreValue)" which will then send a claimed score and a recorded input log to the score server.

We did not implement the Flixel verifier's notion of per-level scoring for the Slick verifier.

1.2 Verification Server

The Slick verifier can use the same server, "basic-score-server.py", as the Flixel verifier due to the server's modular design.

The Slick specific verifier works by running a jar file with a trusted version of the game code running within an instance of the "GameVerifier" class that parses the log and provides the recorded inputs at the appropriate times. Unlike with Flixel, no elaborate local servers are necessary; Java can write to standard out just fine. When the GameVerifier runs through all its inputs, it asks the game for a final score and either accepts or rejects the claimed score.

2 Evaluation

Size and time performance are similar to Flixel's. None of us owns a simple Slick game, so we did not attempt to integrate our system with an existing game, although we did write a trivial test game that uses it.

3 Limitations and Drawbacks

The Slick verifier, patterned after the Flixel verifier, has much the same limitations, as well as an additional implementation problem.

3.1 Getting Slick to work on a Server

We were unable to get the Slick verifier to run on our server. The client game can communicate with the server perfectly well and the server can pass logs to the verifier, which can parse them. But the Slick game, which uses Java's tie-ins to the OpenGL system, refuses to run on a server that doesn't have a display. There is likely some workaround for this, using software to simulate a display, that we could have found with more time and effort; but even if so, making game developers go through such a process to use our system would not be ideal.