# Generating Channel Ids in Virtual World Operating Systems (Extended Abstract)

Michael Roe

Microsoft Research

## Introduction

Two of the most popular software platforms for creating virtual worlds–Second Life and Metaplace–both use a message-passing architecture. *Objects* communicate by sending *messages* on *channels*. The channel's name or identifier acts as a capability: a program that knows the channel's identifier can send and receive from the channel.

## 1 Second Life

In Second Life, all users are placed in one very large virtual world where everyone has the ability to create new objects and attach programs to them. (In reality, Second Life's security model is more complex than this. For example, users who are below 18 years old are isolated from users who are 18+ by being put into an entirely separate virtual world. There are also discretionary access control on who may enter particular parts of the virtual space. But for the purpose of this explanation, we can ignore these details). This means that the main protection mechanism that stops an attacker sending control messages to your objects is that the channel ids are secret.

Metaplace, which was in open beta test from May until December 2009, has a rather different security model, which we describe later. The cryptographic protocol described in this paper was designed for platforms that are similar to Second Life, although they can also be applied to Metaplace.

The security of the system clearly relies on keeping channel ids secret. This usually works because the scripts run on a central server cluster owned by a single organization, not on the client, and the client never learns the channel ids. (In both Metaplace and Second Life, the client is assumed to be untrusted).

Channel ids are often embedded in the source code. This is not immediately a security problem, because the access permissions on scripts can be set so that a user of the script cannot read it.

However, there are two problems with embedding capabilities in scripts. Firstly, it makes the access permissions on scripts more security-critical than they would be otherwise. A compromise of the secrecy of a script (the attacker learns what algorithms it uses etc.) is perhaps not very serious. It is rather more

B. Christianson and J. Malcolm (Eds.): Security Protocols 2010, LNCS 7061, pp. 71–73, 2014.

serious if the attacker uses the capabilities embedded in the script to attack the authentication and integrity properties of the running system (e.g. by sending spoofed control messages to objects).

Secondly, it makes it difficult to organizations to run their own servers (or, more precisely, run other people's scripts on their own servers).

If servers are run by multiple organizations that do not trust each other, secret channel ids embedded in the source code are no longer a viable option. The operators of one server cluster can use their physical access to find out the channel ids used in their own cluster, and then use this knowledge to launch an attack on a rival organization's server. (This assumes that the same scripts are in use on both servers, and each organization has user-level access to the other's system).

## Proposed Solution

- Each software publisher has a private key, $K_A^{-1}$, and corresponding public key $K_A$.
- Scripts are signed: Program, Sign(program, $K_A^{-1}$)
- Clusters of servers each have a secret key, $K_{S_i}$.
- Within server cluster $S_i$, each program computes its channel ids as follows: channel=H($K_A$, name; $K_{S_i}$), where H is a keyed hash function and name is a name for the channel used within the program.
- Each server cluster checks the signature on a script before running it. This check need only be done once, the first time the script is uploaded to the server.

With this approach, we can use the underlying message-passing layer without modification, just changing how we generate the channel ids. It has the advantage that one server cluster (and the organization controlling it) cannot compute the channel ids that are used in a different server cluster. Authors of scripts retain the ability to write scripts that communicate with each other, by using the same $K_A$ to sign each program.

## 2 Metaplace

Metaplace, an alternative software platform for virtual worlds, was in open beta test from May until December in 2009. For Metaplace has an additional layer of protection. Each user gets their own miniature virtual world, that is isolated from the others (in particular, an object in one world cannot send a message to an object in another, at least not without using a different message-passing mechanism that is specifically for talking to untrusted things in other security contexts). Users can enter someone else's virtual world, but unless they have "superuser" privilege in that world, they cannot introduce new programs and attach them to objects.

Metaplace has an additional complication, that a single object in the visualization of the virtual world can have multiple programs (known as *scripts* in Metaplace) attached to it. Scripts attached to the same object can communicate via shared memory; locks are not needed to guard against concurrent access because the scheduler guarantees that at most one thread of control is active in the object at any one time; the object acts as a *monitor*. In Metaplace, the channel id is actually the name of the function that is called when a message is received on the channel). In Metaplace, read-only scripts are referred to as *closed source*. This should not be confused with the more common use of the term, which refers to a particular kind of software license.

**Acknowledgements.** I would like to thank George Danezis for discussions while I was writing this paper, and for presenting the paper at SPW XVIII.