

CYBERNET

OpenSkies

Networking Engine

Security Guide



www.openskies.net

MMPG
MASSIVE
MULTIPLAYER
ONLINE GAMING

Openskies Security Guide

This document is the programmer guide for writing authentication modules that reside on the *FedHost* and *LobbyManager* servers. The server-side Linux/Netmax distribution also includes the source, headers, and makefile for a sample authentication module.

In the Openskies IET Guide we discussed the optional *pSecurityData* argument for several of member functions of the class *CNetInterface*. These parameters, used for user authentication and are described in more detail in that document. This document describes how that *pSecurityData* buffer is used in the servers (*FedHost* and *LobbyManager*).

The public servers accept plug-in modules that will receive the data you send via *pSecurityData* and *dwSecurityDataSize*. You must provide exactly two plug-in modules so that the public server can process your authentication data. One of these modules is loaded by the *LobbyManager*, and one is loaded for each *FedHost*. These modules are “.so” files for LINUX, i.e., shared object (similar to Windows DLL) files. Each module must export two functions:

```
extern "C" CSecurityInfo *CopySecurityInfoClass(const
    CSecurityInfo *pInfo);

extern "C" CSecurityInfo *CreateSecurityInfoClass(DWORD
    dwSecurityDataSize, const void *pvSecurityData);
```

where *CSecurityInfo* is defined in *SecurityInfo.h*. These functions should authenticate the incoming class, and they should return NULL if authentication fails. Besides these two functions, each plug-in must also define a class derived from *CSecurityInfo*. In fact, *CSecurityInfo* itself is a purely virtual class. In order to define the two functions prototyped above, one would have no alternative but to derive a class from *CSecurityInfo*.

To understand *CSecurityInfo*, one must understand how it is used. When a network client (federate) wants to start a new federation, it first makes a connection to a *LobbyManager*. Before anything else can be transmitted via this connection, the client must transmit authentication data to *LobbyManager*. *LobbyManager* then calls *CreateSecurityInfoClass* in its plug-in to create a *CSecurityInfo* class. If the client qualifies as a “root federate”, i.e., if “*IsRootFederate*” returns true, the client is then allowed to start a new federation. If the client actually does start a new federation, *LobbyManager* forwards the authentication data block to *FedHosts* of the new federation. The *FedHosts* then construct *CSecurityInfo* from the same authentication data, and use it to authenticate all incoming connections by calling “*CSecurityInfo::DoLoginCheck*” and “*CSecurityInfo::DoRootFederateCheck*”.

After a “root federate” connects securely to the *LobbyManger*, and before a new federation is created, *LobbyManager* calls “*CopySecurityInfoClass*” to make a copy of the *CSecurityInfo* class. *LobbyManager* later uses this copy to forward the authentication data block to *FedHosts* of the new federation.

The following is a detailed explanation of the *CSecurityInfo* class.

virtual DWORD CSecurityInfo::GetDataSize(void) const

This is a purely virtual function that you must override. It returns the size of an authentication data block according to your data format.

virtual bool CSecurityInfo::CopyData(void *pvBuffer, DWORD dwBufferSize) const

This is a purely virtual function that you must override. When the public server wants to copy authentication data from this class into a memory buffer for network transmission, this function is called. The return value is true if the copy action is successful, and it is false if otherwise.

virtual bool CSecurityInfo::DoLoginCheck(DWORD dwSecurityDataSize, const void *pvSecurityData)

This is a purely virtual function that you must override. This class is constructed by the FedHost with data coming from the LobbyManager. When FedHost receives an authentication data block from a client, FedHost calls this function. You should return true if the data is verified, and you should return false if otherwise.

virtual bool CSecurityInfo::DoRootFederateCheck(DWORD dwSecurityDataSize, const void *pvSecurityData).

This is a purely virtual function that you must override. This class is constructed by the FedHost with data coming from the LobbyManager. When FedHost receives an authentication data block from a client, FedHost calls this function to determine if this client is the one that started the federation. You should return true if the data is verified, and you should return false if otherwise.

virtual bool CSecurityInfo::IsLoginRequired(void) const

This is a purely virtual function that you must override. Return false only if the network is open to anyone.

virtual bool CSecurityInfo::IsLoggedIn(void) const

This is a purely virtual function that you must override. Since this class is created with data received over the network, this function can be called to verify whether a user's login status is timed out or not, etc.

virtual bool CSecurityInfo::IsRootFederate(void) const

This is a purely virtual function that you must override. Since this class is created with data received over the network, this function can be called to verify if the user who sent the data has permission for starting a new federation.

virtual bool CSecurityInfo::IsSysAdministrator(void) const

This is a virtual function that you may override as an option. This gives the user additional privileges such as shutting down the entire system.

Besides these member functions, you need to add constructors and member variables in your plug-in's to store the data received over the network. You may also add code in these plug-in's to communicate with database servers, etc.