Written by : Barak Weichselbaum, all rights reserved 2000-2009

Site: http://www.komodia.com/
Email : barak@komodia.com

KOMODIA'S REDIRECTOR
Programmer's Manual

By Barak Weichselbaum

Introduction

This manual covers the programming aspects of Komodia's Redirector (http://www.komodia.com/index.php?page=redirector.html) when dealing directly with the Redirector's source code. Another option, for modifying the Redirector functionality without access to the source code, is writing a PCProxyDLL using a skeleton (http://www.komodia.com/PCProxyDLL.zip) we provide. Read more about this option in the manual on the DLL programming, at: http://www.komodia.com/KRDR_DLL_manual.pdf

Overview

The project .dsw file is called PCProxy, and there are two classes that the programmer needs to modify. You can view the actual internal implementation of the two classes in the product here:
http://www.komodia.com/LocalRelay.cpp
http://www.komodia.com/LocalClient.cpp

CLocalRelay

This class has only one instance, so all data added to the class will be shared. This class is called from multiple threads, so the programmer must maintain thread-safe practices when accessing its custom data.

The function to modify is:

```
virtual bool NewConnection(bool bFromEncryption,
                           unsigned short usListeningPort,
                           unsigned long& rIP,
                           unsigned short& rPort,
                           unsigned long ulPID,
                           bool& rEnableSSL,
                           bool& rSSLSwitch,
                           bool& bFreezeReceive,
                           DataVector& rData,
                           DataVector& rDataPeek,
                           DWORD& rWaitForMoreData,
                           CControlableClient* pClient)const;
```

The parameters are:

bFromEncryption – In case the data was originally encrypted with SSL but decrypted with the SSL decryptors, this flag will be true.

usListeningPort – The port of the socket that accepted the connection inside the Redirector (currently it will be 12344 or 12346).

rIP – The destination IP that this session is trying to connect to. The programmer can change this value and the session will connect to the new IP.

usListeningPort – The port that accepted the connection. As it is possible to have two ports listening for traffic, sometimes it is important to know which port accepted the connection.

rPort – The destination port that this session is trying to connect to. The programmer can change this value and the session will connect to the new port.

ulPID – The process ID of the process that the session came from.

rEnableSSL – Output this stream as SSL data (usually the port will be changed to 443.)

rSSLSwitch – Set to true to notify that we might change this connection to SSL at a later stage, but not now (useful for proxies.)

bFreezeReceive – Set to true to tell the Redirector not to send us any client data for processing until we are ready (useful when negotiating with 3rd party proxy.)

rData – If there's a need to send data after connection, an example usage would be in case we are connecting to a proxy, then the data is placed inside this vector and will be sent upon connection.

rDataPeek – This vector contains data "peeked" from the client, which means for example that in a web request this can contain the actual request before making any connection, this is useful when wanting to make connecting decisions based on the traffic type.

rWaitForMoreData – This flag sets the timeout in milliseconds to wait before invoking this function again, more information on this in a few paragraphs below.

pClient – The client class of the current session, used to extract more information.

Return value:
True – Allow this session
False – Reject this session

How to work with rDataPeek

When the NewConnection function is first called for a session the rDataPeek vector is empty, incase you want to peek to the data, you set the rWaitForMoreData to a specific timeout and return true (to exit the function), if there's data to give before that timeout occur this function will be called again with the data inside and the variable rWaitForMoreData set to zero, incase the timeout was met the function will called but with the variable rWaitForMoreData set to the timeout value (data that was already peeked will still be in the vector).

This process can occur many times for one session, for example in order to retrieve the web request of a long request (which can be more then one packet size of 1.5k) you can ask for more peeked data until you reach the final \r\n\r\n which indicates the web request is complete.

If you look at the file: LocalRelay.cpp then you will see how we used this feature to peek the request before choosing the correct proxy type.

Samples

Sample code:

```
//Allow only port 80
if (usPort!=80)
    return false;
else
    return true;
```

This code allows only port 80 sessions through. Keep in mind that it only blocks sessions that are intercepted by the LSP using the rule set you predefined in the VB console.

Sample code for redirecting every port 80 session to SSL:

```
//Is it HTTP port?
if (rPort==80)
{
      //Change port to HTTPS
      rPort=443;

      //Enable the SSL
      rEnableSSL=true;
}
```

CLocalClient

This class has an instance per session in the proxy, so for 1000 sessions we will have 1000 instances of this class, which means that custom data in this class will be visible only to the current session.

This class has two functions that you need to intercept. Each function has its dedicated thread, so you will have to maintain thread-safe practices if sharing data between the two functions.

A general outline of the data flow:

Application->LSP->Proxy->DataBeforeSend->Proxy->Final destination
Final destination->Proxy->DataBeforeReceive->Proxy->Application (LSP is not involved at this stage)

DataBeforeSend

This function is called before sending the data to the final destination, (HTTP requests are seen at this stage). The function looks like this:

```
virtual bool DataBeforeSend(const char* pData,
                            DWORD dwDataSize,
                            char** pNewBuffer,
                            DWORD& rNewDataSize,
                            bool& rTerminateSession,
                            bool& rNewBufferToReceive);
```

The parameters are:

pData – The data to process.

dwDataSize – The data size.

pNewBuffer – If the data is modified, this will hold the modified data. It must be allocated by the programmer, since the proxy will delete it.

rNewDataSize – If the data is modified, this will hold the modified data size.

rTerminateSession – Set to true to terminate the session when this function exits.

rNewBufferToReceive – Take the contents of the new data and send it to the application that initiated the session (used mostly for redirects.)

Return value:
True – Means data was modified.
False – Means data wasn't modified.

The flag rTerminateSession is not affected by the return value. You can terminate the session without changing the data.

Sample code:

```
        //Check that we are port 80
        //Get socket data
        DWORD dwTick;
        DWORD dwIP;
        unsigned short usPort;
        DWORD dwPID;
        GetConnectionInformation(dwTick,
                                 dwIP,
                                 usPort,
                                 dwPID);

        //Set terminate to false, just in case
        rTerminateSession=false;

        //Check we are port 80
        if (usPort==80 &&
            (!dwIP ||
             dwIP!=inet_addr("64.118.87.10")))
        {
              //Our quote
              std::string sQuote;
              sQuote='"';

              //Build a new string
              std::string sTmp;
              sTmp+="<html><head><META HTTP-
EQUIV="+sQuote+"Refresh"+sQuote+" CONTENT="+sQuote+"0;URL=";
              sTmp+="http://www.komodia.com";
              sTmp+=sQuote+"></head></html>";

              //Convert the size
              char aTmp[11];
              itoa(sTmp.size(),
                     aTmp,
                     10);

              //HTML header
              std::string sHeader;
              sHeader="HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n";
              sHeader+="Content-Encoding: text/html\r\n";
              sHeader+="Content-Length: ";
              sHeader+=aTmp;
              sHeader+="\r\n\r\n";
              sHeader+=sTmp;

              //Our size
              rNewDataSize=sHeader.size();

              //Allocate new buffer
              *pNewBuffer=new char[rNewDataSize];
              memcpy(*pNewBuffer,
                       sHeader.c_str(),
                       sHeader.size());

              //Send to client
              rNewBufferToReceive=true;

              //Terminate the session
              rTerminateSession=true;
```

```
            //Set to change
            return true;
      }
      else
            return false;
```

In this code, we get the session information, and then redirect every session destined to port 80 to Komodia's website using HTML redirection. The code compares the destination IP of Komodia's website to the destination IP of the session and if it matches this code will not perform a redirect; this is to avoid redirection loops.

DataBeforeReceive

This function is called before sending the data back to the application that initiated the session. An answer from a web server will be caught by this function. The function looks like this:

```
virtual bool DataBeforeReceive(const char* pData,
                               DWORD dwDataSize,
                               char** pNewBuffer
                               DWORD& rNewDataSize,
                               bool& rTerminateSession,
                               bool& rThawReceiveFreeze,
                               bool& rEnableSSL,
                               bool bServerClosed);
```

The parameters are:

pData – The data to process.

dwDataSize – The data to process size.

pNewBuffer – If the data is modified, this will hold the modified data. It must be allocated by the programmer, since the proxy will delete it.

rNewDataSize – If the data is modified, this will hold the modified data size.

rTerminateSession – Set to true to terminate the session when this function exits.

rThawReceiveFreeze – Set to true to allow the Redirector to process data from the client that was held thanks to the bFreezeReceive flag in the function NewConnection.

rEnableSSL – Set to true to switch to SSL session. This can be set only if the flag rSSLSwitch was set during the NewConnection method call.

bServerClosed – This variable will be set to true when the server side closed the connection, the data will have 0 length and it will allow you to do final data send to the client send before closing the connection to the client.

Return value:
True – Means data was modified.
False – Means data wasn't modified.

The flag rTerminateSession is not affected by the return value. You can terminate the session without changing the data.

SocketWasClosed

This function is called when the session has ended.

bError – Indicates whether the session ended because of an error (failed connection attempt), or because the remote peer ended the connection.

Function prototype:

```
virtual void SocketWasClosed(bool bError);
```

Threading model

Calls to NewConnection come from different threads.
Calls to DataBeforeReceive for a specific connection are always from the same thread.
Calls to DataBeforeSend for a specific connection are always from the same thread, but not the same thread as DataBeforeReceive.

C++ code to control proxy usage

The Redirector comes with a helper class that is used to control the Redirector service (PCProxy.EXE) and set the proxy usage.

SetProxyInformation

This function is used to set the proxy to use: (using a proxy is optional of course)

```
std::string SetProxyInformation(const std::string& rIP,
                                unsigned short usPort,
                                const std::string& rUsername,
                                const std::string& rPassword,
                                ProxyType aType);
```

The parameters are:

rIP – Proxy's IP address.

usPort – Proxy's port.

rUsername – Proxy's username - if proxy requires authentication.
rPassword – Proxy's password - if proxy requires authentication.
aType – Proxy type. Ccurrently only HTTP and HTTP Connect proxies are implemented.

Return value:
Empty string - All was OK
Error string – In case of a failure

GetProxyInformation

This function is used to get the proxy to use:

```
std::string GetProxyInformation(std::string& rIP,
                                unsigned short& rPort,
                                std::string& rUsername,
                                std::string& rPassword,
                                ProxyType& rType)const;
```

The parameters are:

rIP – Proxy's IP address.

usPort – Proxy's port.

rUsername – Proxy's username - if proxy requires authentication.

rPassword – Proxy's password - if proxy requires authentication.

aType – Proxy type.

Return value:
Empty string - All was OK.
Error string – In case of a failure

In case there is no proxy configured, rIP, rUsername, and rPassword will be empty strings and rPort will be zero.

ClearProxyServer

This function removes the proxy server, and the Redirector will not use the proxy server (which is the default mode):

```
std::string ClearProxyServer();
```

Return value:
Empty string - All was OK
Error string – In case of a failure