# 共享方案设计

## HOOK当前进程的Cookie变化，并向其他进程共享

## 启动时向其他进程请求Cookie

## 收到其他进程发来的Cookie时，需要设置到自身进程当中

## 为了保证进程通讯的正确性，需要在需要共享Cookie的进程主窗体上设置两个窗口标记

一个为“hook\_cookie”,说明该进程会分享自身的Cookie消息到其他进程

另一个为”use\_hook\_cookie”,说明该进程会响应其他进程共享Cookie的消息并设置到自身进程。

## 进程在请求共享时，向所有具有“hook\_cookie”属性的窗口发送WM\_CD\_GETCOOKIE CopyData消息

## 进程在共享Cookie时向所有具有“use\_hook\_cookie”属性的窗口发送WM\_CD\_SETCOOKIE CopyData消息

# 实现过程

## 定义消息ID

WM\_CD\_SETCOOKIE = (WM\_USER +1)

WM\_CD\_GETCOOKIE = (WM\_USER +2)

## 启动时获取其他进程的Cookie

在主窗口的OnCreate函数中增加以下代码：

SetProp(GetSafeHwnd(),"use\_hook\_cookie",HANDLE(1));

CWnd \* pDeskTopWnd = CWnd::GetDesktopWindow();

CWnd \*pFind = pDeskTopWnd->GetWindow(GW\_CHILD);

while(pFind) //向所有Cookie提供者请求Cookie

{

if (::GetProp(pFind->GetSafeHwnd(),"hook\_cookie"))

{

COPYDATASTRUCT stCopyData = {0};

stCopyData.cbData = 0;

stCopyData.lpData = NULL;

stCopyData.dwData = WM\_CD\_GETCOOKIE;

::SendMessage(pFind->GetSafeHwnd(), WM\_COPYDATA,

reinterpret\_cast<WPARAM>(GetSafeHwnd()),

reinterpret\_cast<LPARAM>(&stCopyData));

}

pFind = pFind->GetWindow(GW\_HWNDNEXT);

}

SetProp(GetSafeHwnd(),"hook\_cookie",HANDLE(1));

以上代码用于进程启动后从其他持有Cookie的进程获取Cookie

## 设置其他进程分享的Cookie到本进程

向主窗口添加COPYDATA处理消息，并在消息处理函数中添加以下处理代码：

if (pCopyDataStruct->dwData == WM\_CD\_SETCOOKIE)

{

CookieData \*pData = static\_cast<CookieData\*>(pCopyDataStruct->lpData);

if (pData->dwProcessId == GetCurrentProcessId())

return TRUE; // 发给自己的消息，不需要设置Cookie

wchar\_t \*pszUrl = NULL, \*pszCookieName = NULL, \*pszCookieData = NULL;

if (pData->dwSizeUrl > 0)

pszUrl = reinterpret\_cast<wchar\_t\*>(pData->pStrData);

if (pData->dwSizeCookieName > 0)

pszCookieName = reinterpret\_cast<wchar\_t\*>((pData->pStrData + pData->dwSizeUrl));

if (pData->dwSizeCookieData > 0)

pszCookieData = reinterpret\_cast<wchar\_t\*>((pData->pStrData + pData->dwSizeUrl + pData->dwSizeCookieName));

InternetSetCookieExW(pszUrl, pszCookieName, pszCookieData, pData->dwFlags, pData->dwReserved);

}

## 分享当前进程Cookie到其他进程

### 在Hook到本进程设置了新的Cookie时，将新的Cookie分享到其他进程，并将新的Cookie保存到历史记录里。

核心代码如下：

BOOL \_\_stdcall Hook\_HttpQueryInfoA(HINTERNET hRequest, DWORD dwInfoLevel,

LPVOID lpBuffer, LPDWORD lpdwBufferLength, LPDWORD lpdwIndex)

{

//获取HttpQueryInfoW的真实地址

static const Proc\_HttpQueryInfoA RealAddr\_HttpQueryInfoA =

reinterpret\_cast<Proc\_HttpQueryInfoA>(::GetProcAddress(GetModuleHandle("WININET.DLL"), "HttpQueryInfoA"));

//调用真实的HttpQueryInfoW

BOOL bRet = RealAddr\_HttpQueryInfoA(hRequest, dwInfoLevel, lpBuffer, lpdwBufferLength, lpdwIndex);

//检查响应头的内容

if (dwInfoLevel == HTTP\_QUERY\_RAW\_HEADERS\_CRLF) //IE总是通过此标志取得整个响应头，从而提取里Set-Cookie:的内容

{

CString strHeaders(static\_cast<char\*>(lpBuffer));

//如果有Set-Cookie:才处理，没有的话说明这个响应头不设置Cookie，直接忽略

if (strHeaders.Find("Set-Cookie:") != -1)

{

//现在响应头是有了，但是这个响应头里的Cookie是哪个网站的？

//所以我们要用hRequest这个句柄，再次调用HttpQueryInfoW查询这个响应头对应的请求头里面的HOST内容，才知道是哪个网站的

//知道了HOST，才能为B进程调用InternetSetCookieExW设置这个Cookie时提供第一个参数LPCWSTR lpszUrl

char \*pwszHost = new char[500];

DWORD dwBuffSize = sizeof(char) \* 500;

memset(pwszHost, 0 , dwBuffSize);

if (RealAddr\_HttpQueryInfoA(hRequest, HTTP\_QUERY\_FLAG\_REQUEST\_HEADERS | HTTP\_QUERY\_HOST, pwszHost, &dwBuffSize, NULL))

{

CStringW strHost = (wchar\_t\*)\_bstr\_t(pwszHost);

CStringW strHeadersW = (wchar\_t\*)\_bstr\_t(strHeaders);

SendCookieToShare(strHost, strHeadersW,true); //把这个头里的Cookie发给B进程

}

delete [] pwszHost;

pwszHost = NULL;

}

}

return bRet;

}

### 在收到Cookie共享请求时，将所有历史Cookie分享到其他进程

核心代码如下：

LRESULT CMainFrame::OnMsgSendCookie(WPARAM,LPARAM)

{

// 共享当前进程的Cookie到其他进程

EnterCriticalSection(&g\_CookieListCS);

for (vector<CookiePair>::const\_iterator iter = m\_HostCookieHistoryList.begin();iter!=m\_HostCookieHistoryList.end();iter++)

{

SendCookieToShare(iter->m\_strHost,iter->m\_strValue);

}

LeaveCriticalSection(&g\_CookieListCS);

return S\_OK;

}

### 分享Cookie相关代码

DWORD WINAPI ThreadFunSendCookie(LPVOID pData)

{

ThreadDataSendCookie \*pSendData = (ThreadDataSendCookie\*)(pData);

CStringW strHost = pSendData->strHost;

CStringW strHeaders = pSendData->strHeaders;

bool bSaveToList = pSendData->bSaveToList;

delete pSendData;

const CStringW strUrl = L"http://" + strHost + L"/";

int nPosL = 0, nPosR = 0;

CStringW strTempW; //保存每一条Cookie的内容

bool IsHttpOnly; //该条Cookie是否是HttpOnly的

//在一个循环中取出这个响应头中的所有Set-Cookie:中的Cookie内容

while (nPosL != -1)

{

nPosL = strHeaders.Find(L"Set-Cookie: ", nPosR);

if (nPosL == -1)

break;

nPosL += wcslen(L"Set-Cookie: ");

nPosR = strHeaders.Find(L"\r\n", nPosL);

if (nPosR == -1)

break;

strTempW = strHeaders.Mid(nPosL, nPosR - nPosL);

//检查是否是HttpOnly的Cookie，如果是的话在B进程中调用InternetSetCookieExW设置Cookie时

//第四个参数DWORD dwFlags应为INTERNET\_COOKIE\_HTTPONLY

if (strTempW.Find(L"HttpOnly") != -1)

IsHttpOnly = true;

else

IsHttpOnly = false;

//发送这条Cookie给进程B

PackCookieDataToSend(strUrl, strTempW, IsHttpOnly);

}

if(bSaveToList)

{

CookiePair cp;

cp.m\_strHost = strHost;

cp.m\_strValue = strHeaders;

EnterCriticalSection(&g\_CookieListCS);

if (m\_HostCookieHistoryList.size() > MAX\_COOKIE\_SETS\_COUNT)

m\_HostCookieHistoryList.erase(m\_HostCookieHistoryList.begin());

m\_HostCookieHistoryList.push\_back(cp);

LeaveCriticalSection(&g\_CookieListCS);

}

return 0;

}

bool SendCookieToShare(const CStringW &strHost, const CStringW &strHeaders,bool bSaveToList)

{

//这个Cookie对应的网站URL，用作B进程中InternetSetCookieExW的第一个参数LPCWSTR lpszUrl

ThreadDataSendCookie \* pThreadParam = new ThreadDataSendCookie;

pThreadParam->strHost = strHost;

pThreadParam->strHeaders = strHeaders;

pThreadParam->bSaveToList = bSaveToList;

CreateThread(NULL,0,ThreadFunSendCookie,pThreadParam,0,NULL);

return true;

}

void PackCookieDataToSend(LPCWSTR pwszUrl, LPCWSTR pwszCookieData, bool IsHttpOnly)

{

//获得pwszUrl和pwszCookieData两个字符串所占内存大小

DWORD dwSizeUrl = (wcslen(pwszUrl) + 1) \* 2;

DWORD dwSizeCookieData = (wcslen(pwszCookieData) + 1) \* 2;

DWORD dwTotalSize = sizeof(CookieData) + dwSizeUrl + dwSizeCookieData ; //要发送的数据总大小

void \*pBuffer = malloc(dwTotalSize);

memset(pBuffer, 0, dwTotalSize);

CookieData \*pData = static\_cast<CookieData\*>(pBuffer);

pData->dwSizeUrl = dwSizeUrl;

pData->dwSizeCookieName = 0; //CookieName为空，这个参数在InternetSetCookieExW中一般不用，直接由CookieData提供Cookie名

pData->dwSizeCookieData = dwSizeCookieData;

pData->dwProcessId = GetCurrentProcessId();

//如果是HttpOnly的Cookie，需要加上该标志

if (IsHttpOnly)

pData->dwFlags = INTERNET\_COOKIE\_HTTPONLY;

else

pData->dwFlags = NULL;

pData->dwReserved = NULL;

//拷贝两个字符串的数据到结构体尾部的柔性数组

char \*pTempChar = pData->pStrData;

memcpy(pTempChar, pwszUrl, dwSizeUrl);

pTempChar += dwSizeUrl;

memcpy(pTempChar, pwszCookieData,dwSizeCookieData);

pTempChar = NULL;

pData = NULL;

COPYDATASTRUCT stCopyData = {0};

stCopyData.cbData = dwTotalSize;

stCopyData.lpData = pBuffer;

stCopyData.dwData = WM\_CD\_SETCOOKIE;

CWnd \* pDeskTopWnd = CWnd::GetDesktopWindow();

CWnd \*pFind = pDeskTopWnd->GetWindow(GW\_CHILD);

while(pFind)

{

if (::GetProp(pFind->GetSafeHwnd(),"use\_hook\_cookie"))

{

//通过WM\_COPYDATA把数据发给另一个进程

::SendMessage(pFind->GetSafeHwnd(), WM\_COPYDATA,

reinterpret\_cast<WPARAM>(AfxGetMainWnd()->GetSafeHwnd()),

reinterpret\_cast<LPARAM>(&stCopyData));

}

pFind = pFind->GetWindow(GW\_HWNDNEXT);

}

free(pBuffer);

pBuffer = NULL;

}