The project designed to copy the screen of a remote host . This part of project is a code of program in C++, which is embedded in the system process explore.exe on the remote host and makes a copy of screen . The pass of module on the remote computer can be realized , for example, by WMI , if an user has network administrator rights . The screen returns as a bmp file in the folder C:\TEMP. The program was tested on Windows XP and Windows 7.

Creating this application was preceded by some event. In Belarus the 8th of March is a holiday - international women's day. On this day men give women gifts and flowers. At work colleagues-women men congratulate the day before. A couple of days before the holiday, my boss asked me to make women surprise. In the morning, when women come to work and will turn on computers, instead of their usual wallpaper desktop screen should appear on the screen bouquet of flowers.

Break a ton of information on the Internet, I decided to use the knowledge that I used in my long career, namely C++. To me it also inspired the works of J. Richter . Also I was familiar not only theoretically, but used in practice, such a powerful tool Windows OS as WMI. So , armed with knowledge about kernel objects and Windows Script Host , I went to work. Of course you could use Sysinternals utilities Suite by Mark Russinovich , but I decided to create something of their own that can be used in further work. So, fix the problem. In the local network invisibly to the users I need to change the Wallpaper for their desktop. My computer I decided to use as a source of information for all other workstations. For this I wrote the Windows Host script , which is installed the agent program on all the hosts and started it . The program agent consists of two parts , in fact the agent and the block of code that were injected in the system process explorer.exe.This clever scheme was necessary due to the fact that if we tried to change the screen by the agent, then nothing would have happened in this case because our agent will act on behalf of the remote user. The agent should be injected its code in one of the processes that are running on behalf of this user to become a user of the computer. All was going well. I wrote the script, created the agent. And suddenly I was faced with a problem. On workstations with Windows 7 my agent is not running. Tomorrow need to do to women surprise , but not everything works. So, ahead of a sleepless night. Again tons of information on the Internet. And , Oh gods, salvation is found. On the website http://securityxploded.com/ I found an article about the vulnerabilities Windows OS, where it was also presents information about undocumented functions. These functions mentioned by Jeffrey Richter in his writings too and he warned that these functions must be used with great caution. But I was lucky , undocumented function worked and the next day all women colleagues were happy .

And now a little about the code provided here. This is only the agent , which is written in C++.Also changed the code block which is embedded in the system process explore.exe. The code that is inserted into explorer.exe does not change the desktop screen. He makes a copy of the screen and gives it to the administrator. I am not an expert Access Control Model MS Windows , but some aspects of this model ,for example, Access tokens had to use to solve my problem."An access token is an object that describes the security context of a process or thread.". It is the definition from

MSDN.

Take a look at our main function of the agent.

```
int _tmain(int argc, _TCHAR* argv[])
 {
BOOL ifSuccess = ObtainSeDebugPrivilege(GetCurrentProcess());
 if (ifSuccess) LaunchAppIntoDifferentSession();
 return 0;
 }
```

As you can see, it consists of two function calls. The first call is very important. If it is successful, we will have the technical implementation of our plan. Although here, too, is not without pitfalls. It was mentioned in the first part of our story. We will return to this later.

So, consider the call

```
BOOL ifSuccess = ObtainSeDebugPrivilege(GetCurrentProcess());
```

As argument of the function is GetCurrentProcess(). From the first part of the story we know that on each remote computer using the script starts the agent. So, the function GetCurrentProcess() returns a handle to agent process. Using ObtainSeDebugPrivilege(GetCurrentProcess()) our agent on the remote computer receives extensive rights that allow him to inject code into any system process. Implementation details this function, we will not give here, so this is a topic for a separate discussion. Now we will just say that it's pretty covered in detail in MSDN.

After successfully overcome the first barrier, the agent proceeds to the next task.

```
 if (ifSuccess) LaunchAppIntoDifferentSession();
```

First, the agent finds the process explorer.exe and obtains a handle to this process.Then it initializes the data structure INJDATA needed to run the embedded process. The next step is the allocation of memory within the process eexplorer.exe for injected data and software module. Here it is necessary to mention function static void __stdcall t() { }

It defined the end of code injected in explorer.exe. This is used when determining the amount of the allocated memory.

```
pProcRemote = (PDWORD)VirtualAllocEx(hProcess, 0, ((SIZE_T)((unsigned int)t -
(unsigned int)SaveBitmapFunc)), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

```
bResult = WriteProcessMemory(hProcess, pProcRemote, SaveBitmapFunc, ((SIZE_T)
((unsigned int)t - (unsigned int)SaveBitmapFunc)),
```

And finally, the agent creates and launches a remote injected process. If it doesn't , starts the above undocumented function NtCreateThreadEx.

```
 HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0,
(LPTHREAD_START_ROUTINE)pProcRemote, pDataRemote, 0/
*CREATE_SUSPENDED*/, &dwThreadID);
```

```
if (!hThread) { // Probably it is Windows 7. Attempt to create a thread for Windows 7
LPFUN_NtCreateThreadEx funNtCreateThreadEx = (LPFUN_NtCreateThreadEx)
GetProcAddress(modNtDll, "NtCreateThreadEx");
```

After the successful launch of embedded remote process the code programmed in the SaveBitmapFunc function   executes.

```
static DWORD WINAPI SaveBitmapFunc(INJDATA* pArguments)
```

The argument passed into the function is a pointer INJDATA. As mentioned earlier , this structure is initialized necessary data before using it. Basically it addresses system functions , which is called inside the function body. In our case, this function works with the screen and the file system. After completing its mission, the remote embedded process is removed.