Windows XP SP3 CSRSS Process Handling Local Privilege Escalation

by Matthew "j00ru" Jurczyk and Gynvael Coldwind
Hispasec

1. Basic Information

Name	Windows XP SP3 CSRSS Local Privilege Escalation
Class	Design Error
Impact	High
Credits	Matthew "j00ru" Jurczyk, Gynvael Coldwind
Discovered	2008-12-16
Published	2010-05-29

2. Abstract

Microsoft Windows XP is a commonly used desktop operating system, released with Service Pack 3 at the time of writing this paper.

One of the system's critical components is the *csrss.exe* executable file. When one of the users decides to log out from the local machine, all the processes running under his account are respectively terminated. Though, it is possible for a restricted user to perform specific actions, consequently leading to a so-called "keep alive" situation – after the current user logs out, some chosen processes are not being shutdown like the rest of them – their execution is kept until they are closed manually (using *taskmgr.exe* etc.)

Assuming that the attacker has direct access to a restricted user's account, and may launch a program able to survive the logout, a privilege escalation attack may take place, since the keep-alive process has access to hardware input and most of the system events, no matter whether/what user is currently logged on the local machine. This includes enumerating windows, manipulating keyboard and mouse (key-logging is possible as well as sending own input messages), making screen-shots and many, many others.

The analysis made so far indicates that Windows Vista is not prone to this vulnerability, though it is not yet confirmed.

3. Vulnerability details

The CSRSS (*csrss.exe*) Windows component, mentioned before, stands for ***C**lient/**S**erver **R**untime **S**ubsystem.* The entire subsystem is build with different modules, each responsible for handling specific type of process requests (*winsrv*, *basesrv*, *coniosrv*). It played an important role in some older system versions. Nowadays, it is mainly responsible for the messages regarding Win32 console windows and registering process-related events like *CreateProcess*, *TerminateProcess*, *CreateThread* etc. The last functionality is the one we are mostly interested in.

As *csrss.exe* is a system process, it always runs with the SYSTEM user privileges. Being designed to communicate with other processes, it must possess some kind of connection channel with every process running on the local machine. In this particular case, Microsoft decided to use the *port* mechanism. Every time a new process is being created, its parent sends a kind of "registration-packet", informing CSRSS about the ongoing event. What is more, before the new process code is executed (the execution reaches the program's EntryPoint), Windows loader calls the *NtSecureConnectPort* function, using a reserved "\Windows\ApiPort" name. After the connection with *csrss.exe* port is established, the process can send it's requests using another function from the *Csr** family – *CsrClientCallServer*. Enough theory, let' s have a look at the real vulnerability details.

The interesting part of the Windows XP internal *_ExitProcess* function listing is as follows:

```
.text:7C81CD69                call    esi ; NtTerminateProcess(x,x)
.text:7C81CD6B                mov     [ebp+var_E0], eax
.text:7C81CD71                call     LdrShutdownProcess@0
.text:7C81CD76                mov     [ebp+var_B4], edi
.text:7C81CD7C                push    4
.text:7C81CD7E                push    10003h
.text:7C81CD83                push    ebx
.text:7C81CD84                lea     eax, [ebp+var_DC]
.text:7C81CD8A                push    eax
.text:7C81CD8B                call    ds:__imp__CsrClientCallServer@16
.text:7C81CD91                push    edi
.text:7C81CD92                push    0FFFFFFFFh
.text:7C81CD94                call    esi ; NtTerminateProcess(x,x)
.text:7C81CD96                or      [ebp+ms_exc.disabled], 0FFFFFFFFh
.text:7C81CD9A                call    sub_7C841B32
.text:7C81CD9F
.text:7C81CD9F loc_7C81CD9F:                          ; CODE XREF: _ExitProcess(x)+1Fj
.text:7C81CD9F                mov     ecx, [ebp+var_1C]
.text:7C81CDA2                call    @  security_check_cookie@4
.text:7C81CDA7                call    __SEH_epilog
.text:7C81CDAC                retn    4
.text:7C81CDAC __ExitProcess@4 endp
```

As you can see, a message indexed as *10003h* is sent to the CSRSS port. This number turns out to point at the *_BaseSrvExitProcess* internal *basesrv.dll* function, responsible for the proper actions related to the termination of current process (removing from the process list etc). What it means is that every time any process is being closed, a specific request is sent to the subsystem, so as it can update its internal structures and stuff.

The vulnerability itself takes advantage of the fact that any process can send such a message during its execution, even if it is not yet going to be terminated. This is very likely to make CSRSS "think" that after it receives such a request, the sender application disappears and doesn't have to be cared about anymore. Therefore, if we run our program with the following line at the beginning of the code:

```
ULONG retValue=0;
CsrClientCallServer(&retValue,0,0x10003,4);
```

then the rest of the code would execute even if we decide to log out. This fact has very serious security consequences. Since such a process might stay unnoticed even after some other user (i.e. admin) logs on the machine, and it can control most of input messages from the keyboard, mouse, screen and what is more, enumerate existing windows as well as perform some operations on them (like setting the window's title), it can be easily used to escalate the privileges of the attacker's user.

As *csrss.exe* establishes a port connection with every process running on the system, the only thing that the attacker must be allowed to do is to launch his own executable file. In order to accomplish a successful privilege escalation attack, the attacker must then log out and wait until some higher-privileged user logs in. After that, the process should then immediately perform appropriate actions to elevate the privileges, before it could even be noticed by other computer users.

4. Impact

This vulnerability allows a local attacker to execute any code (using the controlled parts of system) in the context of another user, but only if that user logs into his account on the same computer.

The impact of a single attack, considering the above requirements, is considered as high. However, if malware would implement this exploit, the impact may change to very high.

5. Disclaimer