Bypassing SACL Auditing on LSASS

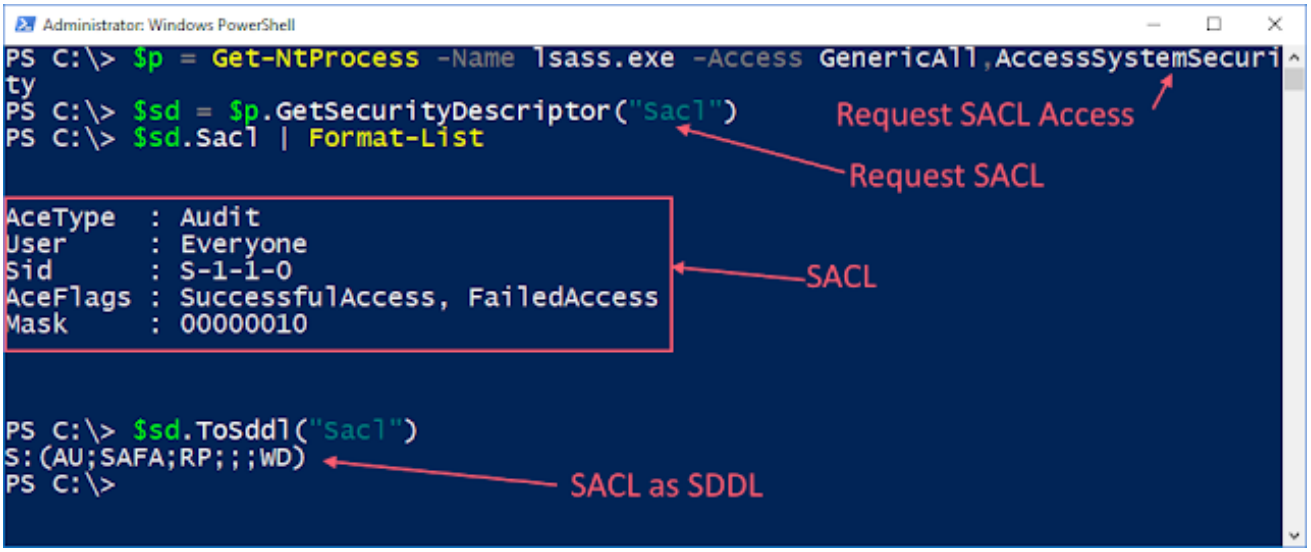
tiraniddo.dev/2017/10/bypassing-sacl-auditing-on-lsass.html

Windows NT has supported the ability to audit resource access from day one. Any audit event ends up in the Security event log. To enable auditing an administrator needs to configure which types of resource access they want to audit in the Local or Group security policy, including whether to audit success and failure. Each resource to audit then needs to have a System Access Control List (SACL) applied which determines what types of access will be audited. The ACL can also specify a principal which limits the audit to specific groups.

My interest was piqued in this subject when I saw a tweet pointing out a change in Windows 10 which introduced a SACL for the LSASS process. The tweet contains a screenshot from a page describing changes in Windows 10 RTM. The implication is this addition of a SACL was to detect the use of tools such as Mimikatz which need to open the LSASS process. But does it work for that specific goal?

Let's take apart this SACL for LSASS, what it means from an auditing perspective and then go into why this isn't a great mechanism to discover Mimikatz or similar programs trying to access the memory of LSASS.

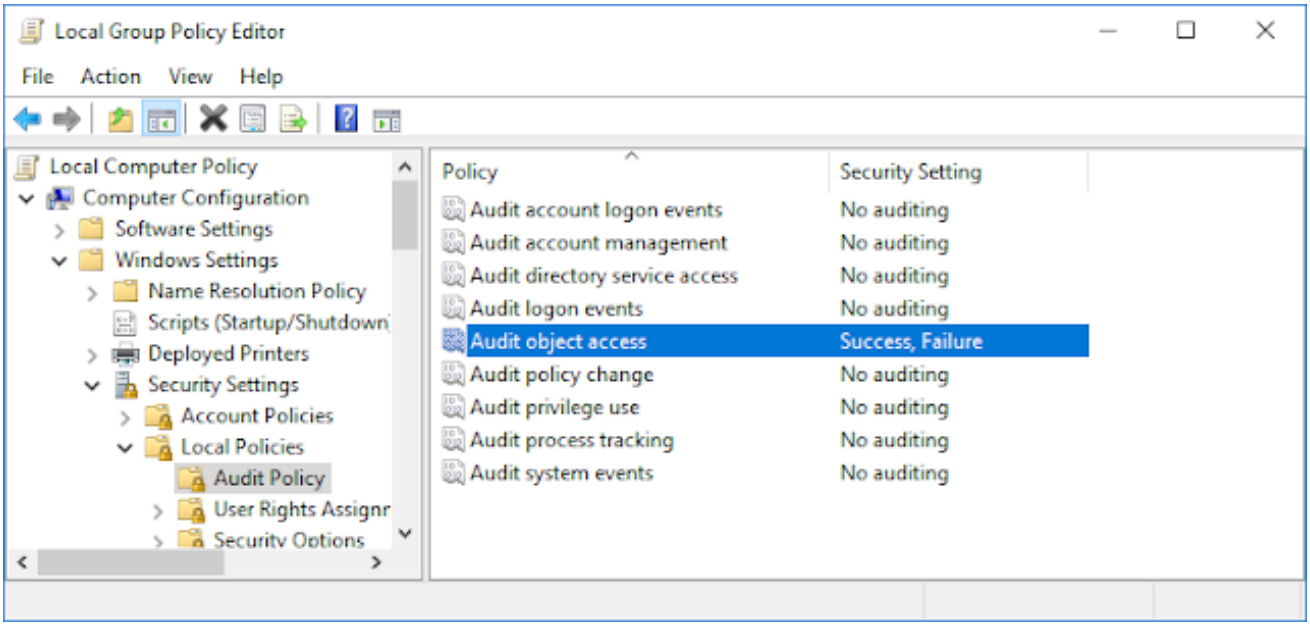
Let's start by setting up a test system so we can verify the SACL is present, then enable auditing to check that we get auditing events when opening LSASS. I updated one of my Windows 10 1703 VMs, then installed the NtObjectManager PowerShell module.



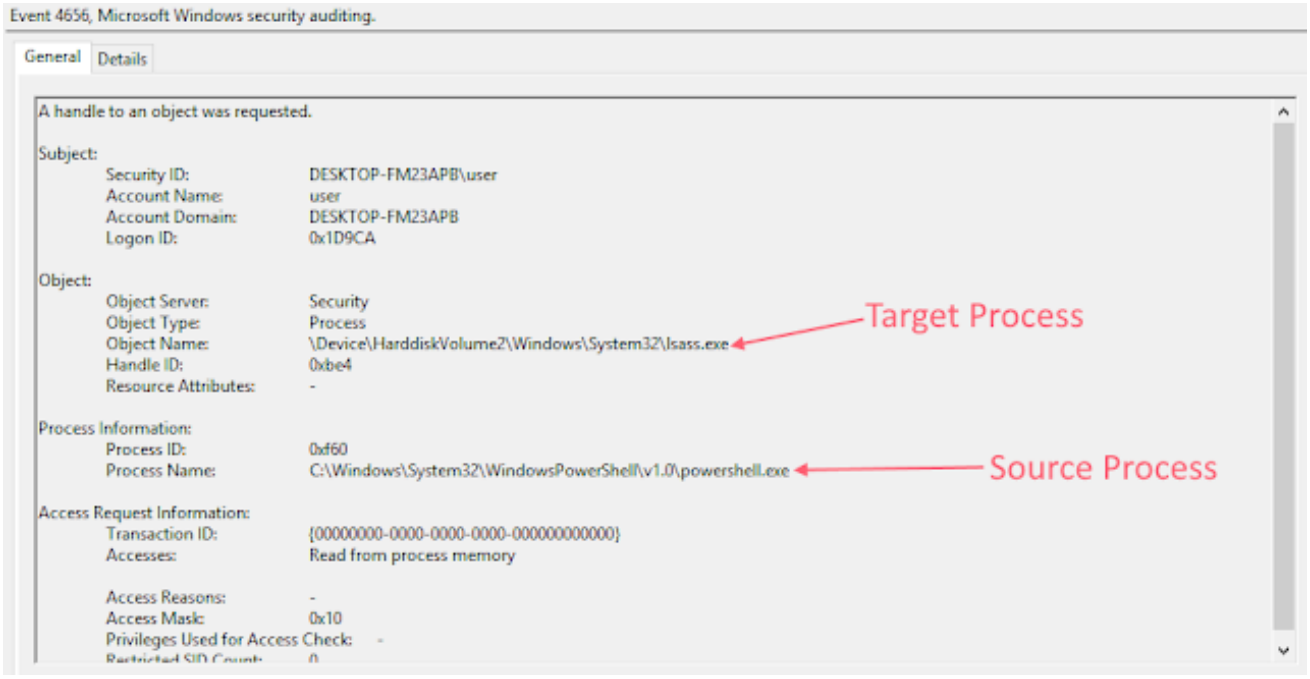
A few things to note here, you must request the ACCESS_SYSTEM_SECURITY access right when opening the process otherwise you can't access the SACL. You must also

explicitly request the SACL when access the process' security descriptor. We can see the SACL as an SDDL string, which matches with the SDDL string from the tweet/Microsoft web page. The SDDL representation isn't a great way of understanding a SACL ACE, so I also expand it out in the middle. The expanded form tells us the ACE is an Audit ACE as expected, that the principal user is the Everyone group, the audit is enabled for both success and failure events and that the mask is set to 0x10.

Okay, let's configure auditing for this event. I enabled Object Auditing in the system's local security policy (for example run gpedit.msc) as shown:



You don't need to reboot to change the auditing configuration, so just reopen the LSASS process as we did earlier in PowerShell, we should then see an audit event generated in the security event log as shown:



We can see that the event contains the target process (LSASS) and the source process (PowerShell) is logged. So how can we bypass this? Well let's look back at what the SACL ACE means. The process the kernel goes through to determine whether to generate an audit event based on a SACL isn't that much different from how the DACL is used in an access check. The kernel tries to find an ACE with a principal which is in the current token's groups and the mask represents one or more access rights which the opened handle has been granted. So looking back at the SACL ACE we can conclude that the audit event will be generated if the current token has the Everyone group and the handle has been granted access 0x10. What's 0x10 when applied to a process? We can find out using the Get-NtAccessMask cmdlet.

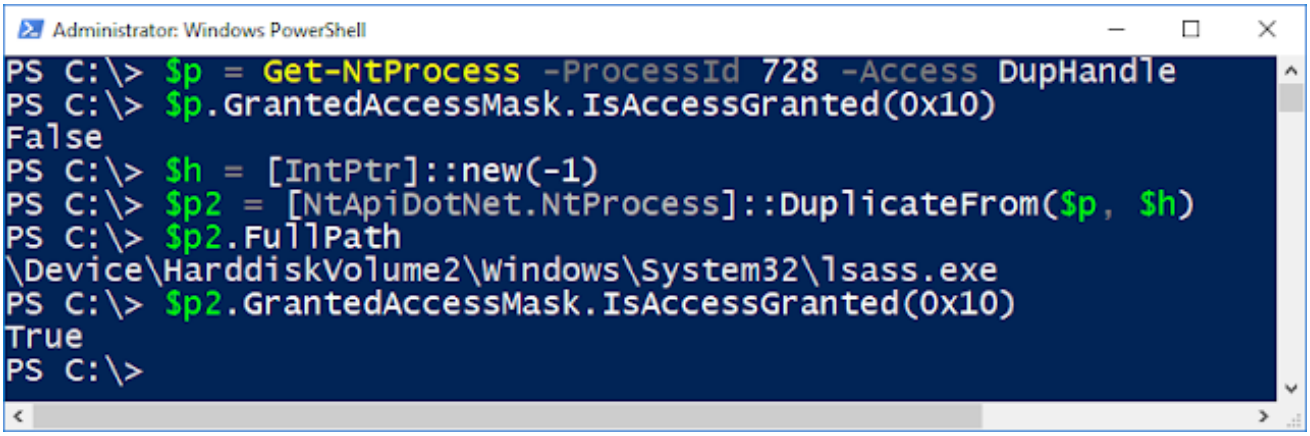
```
PS C:\> Get-NtAccessMask -AccessMask 0x10 -ToSpecificAccess Process
VmRead
```

This shows that the access represents PROCESS_VM_READ, which makes sense. If you're trying to block a process scraping the contents of LSASS the handle needs that access right to call ReadProcessMemory.

The first thought for bypassing this is can you remove the Everyone group from your token and then open the process, at which point the audit rule shouldn't match? Turns out not easily, for a start the only easy way of removing a group from a token is to convert it into a Deny Only group using CreateRestrictedToken. However, the kernel treats Deny Only groups as enabled for the purposes of auditing access checks. You can craft a new token without the group if you have SeCreateTokenPrivilege but it turns out that based on testing that the Everyone group is special and it doesn't matter what groups you have in your token it will still match for auditing.

So what about the access mask instead? If you don't request PROCESS_VM_READ then the audit event isn't triggered. Of course we actually want that access right to do the memory scraping, so how could we get around this? One way is you could open the process for ACCESS_SYSTEM_SECURITY then modify the SACL to remove the audit entry. Of course changing a SACL generates an audit event, though a different event ID to the object access so if you're not capturing those events you might miss it. But it turns out there's at least one easier way, abusing handle duplication.

As I explained in a P0 blog post the DuplicateHandle system call has an interesting behaviour when using the pseudo current process handle, which has the value -1. Specifically if you try and duplicate the pseudo handle from another process you get back a full access handle to the source process. Therefore, to bypass this we can open LSASS with PROCESS_DUP_HANDLE access, duplicate the pseudo handle and get PROCESS_VM_READ access handle. You might assume that this would still end up in the audit log but it won't. The handle duplication doesn't result in an access check so the auditing functions never run. Try it yourself to prove that it does indeed work.



Of course this is just the easy way of bypassing the auditing. You could easily inject arbitrary code and threads into the process and also not hit the audit entry. This makes the audit SACL pretty useless as malicious code can easily circumvent it. As ever, if you've got administrator level code running on your machine you're going to have a bad time.

So what's the takeaway from this? One thing is you probably shouldn't rely on the configured SACL to detect malicious code trying to exploit the memory in LSASS. The SACL is very weak, and it's trivial to circumvent. Using something like Sysmon should do a better job (though I've not personally tried it) or enabling Credential Guard should stop the malicious code opening LSASS in the first place.

UPDATE: I screwed up by description of Credential Guard. CG is using Virtual Secure Mode to isolate the passwords and hashes in LSASS from people scraping the information but it doesn't actually prevent you opening the LSASS process. You can also enable LSASS as a PPL which will block access but I wouldn't trust PPL security.