Windows Security Internals

A Deep Dive into Windows Authentication, Authorization, and Auditing

by James Forshaw

Errata updated to print 2

Page	Error	Correction	Print corrected
54	*Listing 2-34: Displaying the Authenticode signature for* ***a kernel driver***	*Listing 2-34: Displaying the Authenticode signature for* ***notepad.exe***	Print 2

109	`GRAPHITE\user S-1-4-21-2318445812-3516008893-216915059-1002 ❶`
	`...`
	`NT AUTHORITY\ANONYMOUS LOGON S-1-4-7 ❷`
	`...`
	`NT AUTHORITY\ANONYMOUS LOGON S-1-4-7 ❸`
	`...`
	`GRAPHITE\user S-1-4-21-2318445812-3516008893-216915059-1002 ❹`
		`GRAPHITE\user S-1-5-21-2318445812-3516008893-216915059-1002 ❶`
		`...`
		`NT AUTHORITY\ANONYMOUS LOGON S-1-5-7 ❷`
		`...`
		`NT AUTHORITY\ANONYMOUS LOGON S-1-5-7 ❸`
		`...`
		`GRAPHITE\user S-1-5-21-2318445812-3516008893-216915059-1002 ❹`
			Print 2

111	`GRAPHITE\user S-1-4-21-818064984-378290696-2985406761-1002`	`GRAPHITE\user S-1-5-21-818064984-378290696-2985406761-1002`	Print 2
124	For example, the SID of the *BUILTIN\Administrators* group is `S-1-4-32-544`.	For example, the SID of the *BUILTIN\Administrators* group is `S-1-5-32-544`.	Print 2
145	The final component of the security descriptor is the **security** access control list (SACL), which stores auditing rules.	The final component of the security descriptor is the **system** access control list (SACL), which stores auditing rules.	Print 2

240	`-KnownSid OwnerRights -First -AclType Dacl`
	`    if ($sids .Count -gt 0) {`
	`        return`
	`    }`
		`-KnownSid OwnerRights -First -AclType Dacl`
		`    if ($sids -ne $null -and $sids.Count -gt 0) {`
		`        return`
		`    }`
			Print 2

243	Notice that we're intentionally not checking `DenyCallback`. This is because the kernel does not support `DenyCallback` ACEs, although the user mode–only `AuthzAccessCheck` API does.	Notice that we're intentionally not checking `DeniedCallback`. This is because the kernel does not support `DeniedCallback` ACEs, although the user mode–only `AuthzAccessCheck` API does.	Print 2

Page	Error	Correction	Print corrected

258–259	❶ if (!$success) {
	  return Get-AccessResult STATUS_ACCESS_DENIED
	}
	
	❷ $capid = $SecurityDescriptor.ScopedPolicyId
	if ($null -eq $capid) {
	  return Get-AccessResult STATUS_SUCCESS $Context.Privileges $DesiredAccess
	}
	
	❸ $policy = Get-CentralAccessPolicy -CapId $capid.Sid
	if ($null -eq $policy){
	  return Get-AccessResult STATUS_SUCCESS $Context.Privileges $DesiredAccess
	}
	
	❹ $effective_access = $DesiredAccess
	foreach($rule in $policy.Rules) {
	  if ($rule.AppliesTo -ne "") {
	  $resource_attrs = $null
	  if ($sd.ResourceAttributes.Count -gt 0) {
	  $resource_attrs = $sd.ResourceAttributes.ResourceAttribute
	  }
	  ❺ if (!(Test-NtAceCondition -Token $Token -Condition $rule.AppliesTo -ResourceAttribute $resource_attrs)) {
	  continue
	  }
	  }
	  $new_sd = Copy-NtSecurityDescriptor $SecurityDescriptor
	  ❻ Set-NtSecurityDescriptorDacl $rule.Sd.Dacl
	
	  $Context.SecurityDescriptor = $new_sd
	  $Context.RemainingAccess = $DesiredAccess
	
	  ❼ Get-DiscretionaryAccess $Context
	  ❽ **$effective_access = $effective_access -band (-bnot $Context.RemainingAccess)**
	}
	
	❾ **if (Test-NtAccessMask $effective_access -Empty) {**
	  return Get-AccessResult STATUS_ACCESS_DENIED
	}
	❿ return Get-AccessResult STATUS_SUCCESS $Context.Privileges $effective_access
	
	Listing 7-31: The central access policy check
		❶ if (!$success) {
		  return Get-AccessResult STATUS_ACCESS_DENIED
		}
		
		❷ $capid = $SecurityDescriptor.ScopedPolicyId
		if ($null -eq $capid) {
		  return Get-AccessResult STATUS_SUCCESS $Context.Privileges $DesiredAccess
		}
		
		❸ $policy = Get-CentralAccessPolicy -CapId $capid.Sid
		if ($null -eq $policy){
		  return Get-AccessResult STATUS_SUCCESS $Context.Privileges $DesiredAccess
		}
		
		~~❹ $effective_access = $DesiredAccess~~
		foreach($rule in $policy.Rules) {
		  if ($rule.AppliesTo -ne "") {
		  $resource_attrs = $null
		  if ($sd.ResourceAttributes.Count -gt 0) {
		  $resource_attrs = $sd.ResourceAttributes.ResourceAttribute
		  }
		  if (!(Test-NtAceCondition -Token $Token -Condition $rule.AppliesTo
		❶-ResourceAttribute $resource_attrs)) {
		  continue
		  }
		  }
		  $new_sd = Copy-NtSecurityDescriptor $SecurityDescriptor
		  ❺ Set-NtSecurityDescriptorDacl **-SecurityDescriptor $new_sd -Ace $rule.SecurityDescriptor.Dacl**
		
		  $Context.SecurityDescriptor = $new_sd
		  $Context.RemainingAccess = $DesiredAccess
		
		  ❻ Get-DiscretionaryAccess $Context
		  ~~❽ $effective_access = $effective_access -band (-bnot $Context.RemainingAccess)~~
		
		  if (!(Test-NtAccessMask $Contex.RemainingAccess -Empty)) {
		  return Get-AccessResult STATUS_ACCESS_DENIED
		  }
		}
		❼ return Get-AccessResult STATUS_SUCCESS $Context.Privileges $effective_access
		
		Listing 7-31: The central access policy check
			Print 2

Page	Error	Correction	Print corrected
259	**Within the central access policy check, we first set the effective access to the original `DesiredAccess` ❶. We'll use the effective access to determine how much of the `DesiredAccess` we can grant after processing all the policy rules.** Next, we check the `AppliesTo` conditional expression for each rule. If there is no value, the rule applies to all resources and tokens. If there is a conditional expression, we must check it using `Test-NtAceCondition`, passing any resource attributes from the security descriptor ❺. If the test doesn't pass, the check should skip to the next rule. We build a new security descriptor using the owner, group, and SACL from the original security descriptor but the DACL from the rule's security descriptor ❻. If the rule applies, we do another discretionary access check for the `DesiredAccess` ❼. After **this check, we remove any bits that we weren't granted from the `effective_access` variable ❽.**	~~Within the central access policy check, we first set the effective access to the original `DesiredAccess` ❹. We'll use the effective access to determine how much of the `DesiredAccess` we can grant after processing all the policy rules.~~ Next, we check the `AppliesTo` conditional expression for each rule. If there is no value, the rule applies to all resources and tokens. If there is a conditional expression, we must check it using `Test-NtAceCondition`, passing any resource attributes from the security descriptor ❶. If the test doesn't pass, the check should skip to the next rule. We build a new security descriptor using the owner, group, and SACL from the original security descriptor but the DACL from the rule's security descriptor ❺. If the rule applies, we do another discretionary access check for the `DesiredAccess` ❻. After **the discretionary access check, we check that the rule granted all the desired access; if not we return access denied. After we've checked all the applicable rules, we can return success ❼ to grant access.**	Print 2