

# Taking advantage of the fact that the handle returned when you create a kernel synchronization object has full access regardless of the actual ACL

 [devblogs.microsoft.com/oldnewthing/20140828-00](http://devblogs.microsoft.com/oldnewthing/20140828-00)

August 28, 2014



Raymond Chen

A customer wanted some help deciding what security attributes to place on an event object intended to be used by multiple security contexts.

We have two processes, call them A and B, running in different security contexts. I have an event that process A creates and shares with process B. The only thing process A does with the event is signal it, and the only thing process B does with the event is wait on it. Our question is what ACLs you recommend for the event. For now, we're using `O:BAD:(A;;GR;;;WD)(A;;GA;;;LS)(A;;GA;;;BA)`. (In case it matters, process A is usually running as a service with Local System privileges, though for testing purposes it may be running as local administrator. Process B runs as a service with Local Service privileges.)

For those who don't speak SDDL, that weird line noise is shorthand for

- Owner: Builtin Administrators
- DACL:
  - Allow Generic Read to Everyone (aka World).
  - Allow Generic All to Local Service.
  - Allow Generic All to Builtin Administrators.

Given the requirements, there is no need to grant Everyone any access at all, so we can delete the `(A;;GR;;;WD)` ACE.

Since process B needs only to wait on the object, granting it Generic All access is far too broad. That would allow process B to signal the event or even change its ACL! To wait on an object, all you need is Synchronize, so the second ACE can be tightened to `(A;;0x00100000;;;LS)`. (There is no shorthand for Synchronize, so we use its hex value.)

The intention of the third ACE is to allow process A to signal the event, but for that all it needs is `EVENT_MODIFY_STATE`, not Generic All. But we can do better: We can delete the ACE entirely.

“But Mister Wizard, if you delete the third ACE, then process A won’t be able to signal the event!”

Ah yes it can, thanks to a special feature of the CreateEvent function:

| The handle returned by **CreateEvent** has the **EVENT\_ALL\_ACCESS** access right.

If you created the event, you get full access to the event regardless of what the ACLs on the event would normally say.

Therefore, the event can be ACL’d with simply O:BAD:(A;;0x00100000;;;LS). When process A creates the event, it needs to hold on tight to that event handle, since that is the process’s only way of setting the event! (If it loses the handle, it won’t be able to get it back because the attempt to reacquire the handle will be blocked by the ACL.)

Here’s a quick program that demonstrates the behavior.

```
#include <windows.h>
#include <sddl.h>
#include <tchar.h>
// This is a demonstration, so there is no error checking
// and we leak memory.
int __cdecl _tmain(int, TCHAR **)
{
    ULONG cb;
    SECURITY_ATTRIBUTES sa = { sizeof(sa), NULL, FALSE };
    // Create a security descriptor that grants access to no one.
    ConvertStringSecurityDescriptorToSecurityDescriptor(TEXT("D:"),
        SDDL_REVISION_1, &sa.lpSecurityDescriptor, &cb);
    // Create a handle with that security descriptor
    HANDLE h = CreateEvent(&sa, TRUE, TRUE,
        TEXT("NobodyCanAccessMeButMe"));
    // Even though nobody has access to the object, we can still
    // signal it using the handle returned by CreateEvent.
    SetEvent(h); // succeeds
    // But nobody else can obtain the handle via the object name.
    HANDLE h2 = OpenEvent(EVENT_MODIFY_STATE, FALSE,
        TEXT("NobodyCanAccessMeButMe")); // fails
    return 0;
}
```

The customer wrote back, “This worked perfectly. Thanks!”

For bonus points, you can be even more specific and grant Synchronize access only to process B’s service SID ( **NT SERVICE\***ServiceName* ) rather than to all local services.

Raymond Chen

**Follow**

