A user's SID can change, so make sure to check the SID history

devblogs.microsoft.com/oldnewthing/20141128-00

November 28, 2014

Raymond Chen

It doesn't happen often, but a user's SID can change. For example, when I started at Microsoft, my account was in the SYS-WIN4 domain, which is where all the people on the Windows 95 team were placed. At some point, that domain was retired, and my account moved to the REDMOND domain. We saw some time ago that the format of a user SID is

S-1-	version number (SID_REVISION)
-5-	SECURITY_NT_AUTHORITY
-21-	SECURITY_NT_NON_UNIQUE
-w-x-y-	the entity (machine or domain) that issued the SID
-z	the unique user ID for that entity

The issuing entity for a local account on a machine is the machine to which the account belongs. The issuing entity for a domain account is the domain.

If an account moves between domains, the issuing entity changes, which means that the old SID is not valid. A new SID must be issued.

Wait, does this mean that if my account moves between domains, then I lose access to all my old stuff? All my old stuff grants access to my old SID, not my new SID.

Fortunately, this doesn't happen, thanks to the *SID history*. When your account moves to the new domain, the new domain controller remembers all the previous SIDs you used to have. When you authenticate against the domain controller, it populates your token with your SID history. In my example, it means that my token not only says "This is user number 271828 on the REDMOND domain", it also says "This user used to be known as number 31415 on the SYS-WIN4 domain." That way, when the system sees an object whose ACL says, "Grant access to user 31415 on the SYS-WIN4 domain," then it should grant me access to that object.

The existence of SID history means that recognizing users when they return is more complicated than a simple `EqualSid`, because `EqualSid` will say that "No, S-1-5-21-REDMOND-271828 is not equal to S-1-5-21-SYS-WIN4-31415," even though both SIDs refer to the same person.

If you are going to remember a SID and then try to recognize a user when they return, you need to search the SID history for a match, in case the user changed domains between the two visits. The easiest way to do this is with the `AccessCheck` function. For example, suppose I visited your site while I belong to the SYS-WIN4 domain, and you remembered my SID. When I return, you create a security descriptor that grants access to the SID you remembered, and then you ask `AccessCheck`, "If I had an object that granted access only to this SID, would you let this guy access it?"

(So far, this is just recapping stuff I discussed a few months ago. Now comes the new stuff.)

There are a few ways of building up the security descriptor. In all the cases, we will create a security descriptor that grants the specified SID some arbitrary access, and then we will ask the operating system whether the current user has that access.

My arbitrary access shall be

```
#define FROB_ACCESS     1 // any single bit less than 65536
```

One way to build the security descriptor is to let SDDL do the heavy lifting: Generate the string `D:(A;;1;;;<SID>)` and then pass it to `StringSecurityDescriptorToSecurity-Descriptor`.

Another is to build it up with security descriptor functions. I defer to the sample code in MSDN for an illustration.

The hard-core way is just to build the security descriptor by hand. For a security descriptor this simple, the direct approach involves the least amount of code. Go figure.

The format of the security descriptor we want to build is

```
struct ACCESS_ALLOWED_ACE_MAX_SIZE
{
    ACCESS_ALLOWED_ACE Ace;
    BYTE SidExtra[SECURITY_MAX_SID_SIZE - sizeof(DWORD)];
};
```

The `ACCESS_ALLOWED_ACE_MAX_SIZE` structure represents the maximum possible size of an `ACCESS_ALLOWED_ACE`. The `ACCESS_ALLOWED_ACE` leaves a `DWORD` for the SID (`Sid-Start`), so we add additional bytes afterward to accommodate the largest valid SID. If you wanted to be more C++-like, you could make `ACCESS_ALLOWED_ACE_MAX_SIZE` derive from `ACCESS_ALLOWED_ACE`.

```
struct ALLOW_ONLY_ONE_SECURITY_DESCRIPTOR
{
    SECURITY_DESCRIPTOR_RELATIVE Header;
    ACL Acl;
    ACCESS_ALLOWED_ACE_MAX_SIZE Ace;
};
const ALLOW_ONLY_ONE_SECURITY_DESCRIPTOR c_sdTemplate = {
  // SECURITY_DESCRIPTOR_RELATIVE
  {
    SECURITY_DESCRIPTOR_REVISION,           // Revision
    0,                                      // Reserved
    SE_DACL_PRESENT | SE_SELF_RELATIVE,     // Control
    FIELD_OFFSET(ALLOW_ONLY_ONE_SECURITY_DESCRIPTOR, Ace.Ace.SidStart),
                                            // Offset to owner
    FIELD_OFFSET(ALLOW_ONLY_ONE_SECURITY_DESCRIPTOR, Ace.Ace.SidStart),
                                            // Offset to group
    0,                                      // No SACL
    FIELD_OFFSET(ALLOW_ONLY_ONE_SECURITY_DESCRIPTOR, Acl),
                                            // Offset to DACL
  },
  // ACL
  {
    ACL_REVISION,                           // Revision
    0,                                      // Reserved
    sizeof(ALLOW_ONLY_ONE_SECURITY_DESCRIPTOR) -
    FIELD_OFFSET(ALLOW_ONLY_ONE_SECURITY_DESCRIPTOR, Acl),
                                            // ACL size
    1,                                      // ACE count
    0,                                      // Reserved
  },
  // ACCESS_ALLOWED_ACE_MAX_SIZE
  {
    // ACCESS_ALLOWED_ACE
    {
      // ACE_HEADER
      {
        ACCESS_ALLOWED_ACE_TYPE,            // AceType
        0,                                  // flags
        sizeof(ACCESS_ALLOWED_ACE_MAX_SIZE),// ACE size
      },
      FROB_ACCESS,                          // Access mask
    },
  },
};
```

Our template security descriptor says that it is a self-relative security descriptor with an owner, group and DACL, but no SACL. The DACL consists of a single ACE. We set up everything in the ACE except for the SID. We point the owner and group to that same SID. Therefore, this security descriptor is all ready for action once you fill in the SID.

```
BOOL IsInSidHistory(HANDLE Token, PSID Sid)
{
  DWORD SidLength = GetLengthSid(Sid);
  if (SidLength > SECURITY_MAX_SID_SIZE) {
    // Invalid SID. That's not good.
    // Somebody is playing with corrupted data.
    // Stop before anything bad happens.
    RaiseFailFastException(nullptr, nullptr, 0);
  }
  ALLOW_ONLY_ONE_SECURITY_DESCRIPTOR Sd = c_sdTemplate;
  CopyMemory(&Sd.Ace.Ace.SidStart, Sid, SidLength);
```

As you can see, generating the security descriptor is a simple matter of copying our template and then replacing the SID. The next step is performing an access check of the token against that SID.

```
  const static GENERIC_MAPPING c_GenericMappingFrob = {
    FROB_ACCESS,
    FROB_ACCESS,
    FROB_ACCESS,
    FROB_ACCESS,
  };
  PRIVILEGE_SET PrivilegeSet;
  DWORD PrivilegeSetSize = sizeof(PrivilegeSet);
  DWORD GrantedAccess = 0;
  BOOL AccessStatus = 0;
  return AccessCheck(&Sd, Token, FROB_ACCESS,
    const_cast<PGENERIC_MAPPING>(&c_GenericMappingFrob),
    &PrivilegeSet, &PrivilegeSetSize,
    &GrantedAccess, &AccessStatus) &&
    AccessStatus;
}
```

So let's take this guy out for a spin. Since I don't know what is in your SID history, I'm going to pick something that should be in your token already (*Authenticated Users*) and something that shouldn't (*Local System*).

```
// Note: Error checking elided for expository purposes.
void CheckWellKnownSid(HANDLE Token, WELL_KNOWN_SID_TYPE type)
{
  BYTE rgbSid[SECURITY_MAX_SID_SIZE];
  DWORD cbSid = sizeof(rgbSid);
  CreateWellKnownSid(type, NULL, rgbSid, &cbSid);
  printf("Is %d in SID history? %d\n", type,
         IsInSidHistory(Token, rgbSid));
}
int __cdecl wmain(int argc, wchar_t **argv)
{
  HANDLE Token;
  // In real life you had better error-check these calls,
  // to avoid a security hole.
  ImpersonateSelf(SecurityImpersonation);
  OpenThreadToken(GetCurrentThread(), TOKEN_QUERY, TRUE, &Token);
  RevertToSelf();
  CheckWellKnownSid(Token, WinAuthenticatedUserSid);
  CheckWellKnownSid(Token, WinLocalSystemSid);
  CloseHandle(Token);
  return 0;
}
```

Related reading: Hey there token, long time no see! (Did you do something with your hair?)

Raymond Chen

Follow