SetEntriesInAcl

Problems dealing with discretionary access control lists (DACLs) that contain inheritable access control entries (ACEs)

Sean Barnum, Cigital, Inc. [vita[1]]

Copyright © 2007 Cigital, Inc.

2007-04-04

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 13054 bytes

Attack Category	• Identity Spoofing
Vulnerability Category	• Access Control
Software Context	• Authorization
Location	• aclapi.h
Description	The SetEntriesInAcl function creates a new access control list (ACL) by merging new access control or audit control information into an existing ACL structure. SetEntriesInAcl has some problems dealing with discretionary access control lists (DACLs) that contain inheritable access control entries (ACEs).
APIs	**Function Name** / **Comments** SetEntriesInAcl
Method of Attack	
Exception Criteria	None known.
Solutions	**Solution Applicability** / **Solution Description** / **Solution Efficacy** Always check the return code from the call to SetEntriedinACL. Address the issue of inherited ACEs in the in-param DACLs.
Signature Details	DWORD SetEntriesInAcl(ULONG cCountOfExplicitEntries, PEXPLICIT_ACCESS pListOfExplicitEntries, PACL OldAcl, PACL* NewAcl);

1. http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

Examples of Incorrect Code	

```
// This will succeed only if the
caller has WRITE_DAC access to
the object or is the owner of the
object

BOOL TakeOwnership(LPTSTR
lpszOwnFile)
{

BOOL bRetval = FALSE;

HANDLE hToken = NULL;
PSID pSIDAdmin = NULL;
PSID pSIDEveryone = NULL;
PACL pACL = NULL;
SID_IDENTIFIER_AUTHORITY
SIDAuthWorld =
SECURITY_WORLD_SID_AUTHORITY;
SID_IDENTIFIER_AUTHORITY
SIDAuthNT = SECURITY_NT_AUTHORITY;
const int NUM_ACES = 2;
EXPLICIT_ACCESS ea[NUM_ACES];
DWORD dwRes;

// Specify the DACL to use.
// Create a SID for the Everyone
group.
if (!
AllocateAndInitializeSid(&SIDAuthWorld,
1,
SECURITY_WORLD_RID,
0,
0, 0, 0, 0, 0, 0,
&pSIDEveryone))
{
printf("AllocateAndInitializeSid
(Everyone) error %u\n",
GetLastError());
goto Cleanup;
}

// Create a SID for the BUILTIN
\Administrators group.
if (!
AllocateAndInitializeSid(&SIDAuthNT,
2,
SECURITY_BUILTIN_DOMAIN_RID,
DOMAIN_ALIAS_RID_ADMINS,
0, 0, 0, 0, 0, 0,
&pSIDAdmin))
{
printf("AllocateAndInitializeSid
(Admin) error %u\n",
GetLastError());
goto Cleanup;
}
```

```
ZeroMemory(&ea, NUM_ACES *
sizeof(EXPLICIT_ACCESS));

// Set read access for Everyone.
ea[0].grfAccessPermissions =
GENERIC_READ;
ea[0].grfAccessMode = SET_ACCESS;
ea[0].grfInheritance =
NO_INHERITANCE;
ea[0].Trustee.TrusteeForm =
TRUSTEE_IS_SID;
ea[0].Trustee.TrusteeType =
TRUSTEE_IS_WELL_KNOWN_GROUP;
ea[0].Trustee.ptstrName =
(LPTSTR) pSIDEveryone;

// Set full control for
Administrators.
ea[1].grfAccessPermissions =
GENERIC_ALL;
ea[1].grfAccessMode = SET_ACCESS;
ea[1].grfInheritance =
NO_INHERITANCE;
ea[1].Trustee.TrusteeForm =
TRUSTEE_IS_SID;
ea[1].Trustee.TrusteeType =
TRUSTEE_IS_GROUP;
ea[1].Trustee.ptstrName =
(LPTSTR) pSIDAdmin;

if (ERROR_SUCCESS !=
SetEntriesInAcl(NUM_ACES,
ea,
NULL,
&pACL))
{
printf("Failed SetEntriesInAcl
\n");
goto Cleanup;
}

// Try to modify the object's
DACL.
dwRes = SetNamedSecurityInfo(
lpszOwnFile, // name of the
object
SE_FILE_OBJECT, // type of object
DACL_SECURITY_INFORMATION, //
change only the object's DACL
NULL, NULL, // do not change
owner or group
pACL, // DACL specified
NULL); // do not change SACL

if (ERROR_SUCCESS == dwRes)
{
```

```
printf("Successfully changed DACL
\n");
bRetval = TRUE;
// No more processing needed.
goto Cleanup;
}
if (dwRes != ERROR_ACCESS_DENIED)
{
printf("First
SetNamedSecurityInfo call failed:
%u\n", dwRes);
goto Cleanup;
}
```

```
// If the preceding call failed
because access was denied,
// enable the
SE_TAKE_OWNERSHIP_NAME privilege,
create a SID for
// the Administrators group, take
ownership of the object, and
// disable the privilege. Then
try again to set the object's
DACL.
```

```
// Open a handle to the access
token for the calling process.
if (!
OpenProcessToken(GetCurrentProcess(),
TOKEN_ADJUST_PRIVILEGES,
&hToken))
{
printf("OpenProcessToken failed:
%u\n", GetLastError());
goto Cleanup;
}
```

```
// Enable the
SE_TAKE_OWNERSHIP_NAME privilege.
if (!SetPrivilege(hToken,
SE_TAKE_OWNERSHIP_NAME, TRUE))
{
printf("You must be logged on as
Administrator.\n");
goto Cleanup;
}
```

```
// Set the owner in the object's
security descriptor.
dwRes = SetNamedSecurityInfo(
lpszOwnFile, // name of the
object
SE_FILE_OBJECT, // type of object
OWNER_SECURITY_INFORMATION, //
change only the object's owner
```

```
pSIDAdmin, // SID of
Administrator group
NULL,
NULL,
NULL);

if (dwRes != ERROR_SUCCESS)
{
printf("Could not set owner.
Error: %u\n", dwRes);
goto Cleanup;
}

// Disable the
SE_TAKE_OWNERSHIP_NAME privilege.
if (!SetPrivilege(hToken,
SE_TAKE_OWNERSHIP_NAME, FALSE))
{
printf("Failed SetPrivilege call
unexpectedly.\n");
goto Cleanup;
}

// Try again to modify the
object's DACL, now that we are the
owner.
dwRes = SetNamedSecurityInfo(
lpszOwnFile, // name of the
object
SE_FILE_OBJECT, // type of object
DACL_SECURITY_INFORMATION, //
change only the object's DACL
NULL, NULL, // do not change
owner or group
pACL, // DACL specified
NULL); // do not change SACL

if (dwRes == ERROR_SUCCESS)
{
printf("Successfully changed DACL
\n");
bRetval = TRUE;
}
else
{
printf("Second
SetNamedSecurityInfo call failed:
%u\n", dwRes);
}

Cleanup:

if (pSIDAdmin)
FreeSid(pSIDAdmin);

if (pSIDEveryone)
FreeSid(pSIDEveryone);
```

```
if (pACL)
LocalFree(pACL);

if (hToken)
CloseHandle(hToken);

return bRetval;

}
```

Examples of Corrected Code

```
// Return code is being checked.

int check_acl( const char * const
path )
{
DWORD dwrc, i;
int dirty = 0;
PSID psidOwner, psidGroup;
PACL paclDacl = NULL, paclSacl =
NULL, pNewDacl = NULL;
PSECURITY_DESCRIPTOR pSD;
PTRUSTEE pOwner = NULL, pGroup =
NULL;
ULONG cDaclLen, cSaclLen;
PEXPLICIT_ACCESS pDacl = NULL,
pSacl = NULL;
char lp[MAX_PATH];

strncpy( lp, path, MAX_PATH-1);
lp[MAX_PATH-1]='\0';

dwrc = GetNamedSecurityInfo( lp,
SE_FILE_OBJECT,
OWNER_SECURITY_INFORMATION |
GROUP_SECURITY_INFORMATION
| DACL_SECURITY_INFORMATION
| SACL_SECURITY_INFORMATION,
&psidOwner, &psidGroup, &paclDacl,
&paclSacl, &pSD );
if ( dwrc )
{
printf( "GetNamedSecurityInfo():
rc = %lu\n", dwrc );
return FALSE;
}

dwrc =
LookupSecurityDescriptorParts( &pOwner,
&pGroup, &cDaclLen, &pDacl,
&cSaclLen, &pSacl, pSD );
if ( dwrc )
{
printf( "LookupSecurityDescriptorParts():
rc = %lu\n", dwrc );
return FALSE;
}

printf( "%s:\n", lp );
```

```
if ( display_owner )
{
print_sid( 2, "Owner",
psidOwner );
if ( pOwner != NULL )
{
printf( " excess size(*pOwner) ==
%u bytes\n", LocalSize( pOwner ) -
sizeof *pOwner );
print_trustee( 2, "Owner",
*pOwner );
}
else
printf( " odd -- pOwner is NULL
\n" );
}

if ( display_group )
{
print_sid( 2, "Group",
psidGroup );
print_trustee( 2, "Group",
*pGroup );
}

if ( *( (int *) &display_dacl ) )
{
if ( paclDacl != NULL )
{
printf( " DACL: %d ACEs\n",
cDaclLen );
for ( i = 0; i < cDaclLen; i ++ )
print_ace( 0, i, paclDacl );
}
else
printf( " DACL: not present\n" );
}

if ( fix_inherit )
{
dirty = 0;
for ( i = 0; i < cDaclLen; i ++ )
{
// do we need to set the
inheritance?
if ( ( pDacl[i].grfInheritance &
SUB_CONTAINERS_AND_OBJECTS_INHERIT ) !
=
SUB_CONTAINERS_AND_OBJECTS_INHERIT )
{
pDacl[i].grfInheritance |=
SUB_CONTAINERS_AND_OBJECTS_INHERIT;
dirty = 1;
}
}
```

```
// write back the ACL, if dirty
bit set
dwrc = SetEntriesInAcl( cDaclLen,
pDacl, paclDacl, &pNewDacl );
if ( dwrc == 0 )
{
dwrc = SetNamedSecurityInfo( lp,
SE_FILE_OBJECT,
DACL_SECURITY_INFORMATION,
psidOwner, psidGroup, pNewDacl,
paclSacl );
if ( dwrc )
{
printf( "SetNamedSecurityInfo():
rc = %lu\n", dwrc );
}
}
else
{
printf( "SetEntriesInAcl(): rc =
%lu\n", dwrc );
}
}

if ( *( (int *) &display_sacl ) )
{
if ( paclSacl != NULL )
{
printf( " SACL: %d ACEs\n",
cSaclLen );
for ( i = 0; i < cSaclLen; i ++ )
print_ace( 1, i, paclSacl );
}
else
printf( " SACL: not present\n" );
}

if ( pOwner ) LocalFree( pOwner );
if ( pGroup )
LocalFree( pGroup );
if ( pDacl ) LocalFree( pDacl );
if ( pNewDacl )
LocalFree( pNewDacl );
if ( pSacl ) LocalFree( pSacl );

return 0;
}
```

```
ACL* pdacl = 0;
DWORD err = SetEntriesInAcl(1, &ea, 0, &pdacl);

ea.grfAccessPermissions = dwDesiredAccess;
ea.Trustee.ptstrName = (LPTSTR)(&world);

if (err == ERROR_SUCCESS)
{
```

	err = SetSecurityInfo(h, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION, 0, 0, pdacl, 0);
	LocalFree(pdacl);
	return (err == ERROR_SUCCESS);
	}
	else
	{
	TRACE("adjust_dacl");
	return(FALSE);
	}

Source References	• Howard, Michael & LeBlanc, David C. *Writing Secure Code*, 2nd ed. Redmond, WA: Microsoft Press, 2002, ISBN: 0735617228. • http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/setentriesinacl.asp[2] • Drzik, Dalibor. Find and kill a process which uses the specified loaded dynamic library (.dll)[3] (2005).
Recommended Resource	
Discriminant Set	**Operating System**: • Windows **Languages**: • C • C++

Cigital, Inc. Copyright

Copyright © Cigital, Inc. 2005-2007. Cigital retains copyrights to this material.

Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

For information regarding external or commercial use of copyrighted materials owned by Cigital, including information about “Fair Use,” contact Cigital at copyright@cigital.com[1].

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com