Programmierung in Windows Sicherheit mit JEDI Windows Security Code Library JWSCL

Christian Wimmer

28. Oktober 2007

Inhaltsverzeichnis

Einleitung

Während der Programmierung von Windows Anwendungen gab es immer wieder Fälle, in denen ich mich über bestimmte Windows Sicherheitsaspekte sorgen musste. Wollte ich den Computer herunterfahren, so musste zuerst ein bestimmtes Privileg aktiviert werden. Sollte der Zugriff auf eine Datei für andere Benutzer gestattet sein, so musste dies durch Programmtext erfolgen. Diese und andere Sicherheitsaspekte sind unter Win32 jedoch nur allein mit der Windows API zu bewerkstelligen, die in C implementiert ist und daher für Delphiprogrammierer etwas schwer zu verwenden ist. Anstatt sich mit dem eigentlichen Problem zu befassen musste ich mir Sorgen über Zeiger und Strukturen machen, die an bestimmte Funktionen übergeben werden mussten. Besonders dynamische Strukturen, die in ihrer Größe geändert werden können, jedoch in einem Stück im Speicher liegen sind weit entfernt davon „dynamisch“ zu sein. Änderungen, und wenn sie auch noch so klein sind, bedeuten eine Kopie des Speicherblocks mit der Änderung herzustellen und dann zu verwenden. Das war und ist alles andere als intuitiv und trivial. Der Beschluss die Win32 Sicherheits API oder zumindest ein Teil davon, als Klassensystem zu implementieren war daher der nächste Schritt. Im Nachhinein gesehen war der Aufwand jedoch alles andere als gering. Er war monumental! Ich fing im Januar 2007 and und noch immer sind nicht alle Implementationen abgeschlossen und zudem fehlen einige Sicherheitsaspekte der Win32 API gänzlich. Der Kern ist jedoch nahezu vollständig und 99% der Methoden, Klassen und mehr sind dokumentiert. Einige Beispielprogramme sind bereit, um die Bibliothek zu testen und ihre Mächtigkeit zu zeigen. Der erste gedachte Releasetermin war Anfang Oktober 2007, jedoch gab es keinerlei Information, wie man die Bibliothek verweden könnte. Es fehlte zudem eine Kommunikationsplattform. Ich entschloss mich daher noch etwas zu warten. Ein oder zwei Monate vor dem geplanten Release im Oktober wurde mir dann nahegelegt, die Bibliothek vielleicht mit Quelltext offenzulegen. Ein schwerer Schritt. Einerseits hätte ich ja die Bibliothek verkaufen können. Jedoch war andererseits der Aufwand und Support viel zu aufwendig. Die JEDI[1] kannte ich schon von früheren Begegnungen. Und nun nutzte ich die JEDI API Lib Header für die API Einbindung. Ich entschloss mich daher, der JEDI als eigenständiges Projekt beizutreten. Die JWSCL wird daher als Schwesterprojekt neben der JEDI API veröffentlicht. Dies wird auch geschehen in dem Glaube, dass noch andere Delphiprogrammierer ihren Beitrag leisten. Aber das wird die Zeit zeigen...

Euer Christian

[1] Joint Endeavour of Delphi Innovators

1 Präambeln

Bei der Entwicklung der JEDI Windows Security Code Library war immer der Gedanke der Einfachheit zu gegen. Einfachheit bedeutet jedoch nicht, dass ein Unwissender damit sofort umgehen kann. Wie alles auf der Welt muss ein Grundverständnis vorhanden sein. Einfachheit bedeutet hier vielmehr, dass die Benutzung ohne Gedanke an Untiefen und Klippen geschehen kann. Die objektorientierte Programmierung unterstüzt diesen Ansatz noch.
Leider kann die Einfachheit jedoch nicht immer aufrecht erhalten werden. Entweder etwas ist sehr einfach zu handhaben oder es ist sehr flexibel und mächtig. Beides auf einmal ist nicht möglich. In dieser Situation habe ich mich gegen die Einfachheit und für die Mächtigkeit entschieden. Dies geschah jedoch nicht ohne Dokumentation von Gefahren.

2 Grundlagen Sicherheits API

2.1 AccessCheck Teil 1

```
BOOL WINAPI AccessCheck(
  __in          PSECURITY_DESCRIPTOR pSecurityDescriptor,
  __in          HANDLE ClientToken,
  __in          DWORD DesiredAccess,
  __in          PGENERIC_MAPPING GenericMapping,
  __out_opt     PPRIVILEGE_SET PrivilegeSet,
  __in_out      LPDWORD PrivilegeSetLength,
  __out         LPDWORD GrantedAccess,
  __out         LPBOOL AccessStatus
);
```

AccessCheck[1]

Es ist recht ungewöhnlich am Anfang mit einer anscheindend willkürlichen Funktion zu beginnen. Die Definition der Funktion ist jedoch entscheidend. Denn sie enthält alles was Sicherheit under Windows ausmacht. *AccessCheck* ist **die** zentrale Funktion! Um AccessCheck verwenden zu können müssen daher einige Aspekte noch genauer erläutert werden. Dazu zählen

- Access mask
- Generic Mapping
- Security Descriptor
- Token
- Privilege

Wir werden uns daher zuerst mit diesen Dingen beschäftigen und dann zum Schluss nochmals AccessCheck zuwenden.

2.1.1 Access Mask

Die Zugriffsmaske bestimmt die Art und Weise, wie ein Zugriff auf ein Objekt (Datei, Registrierschlüssel) erfolgen soll. Die Maske bleibt solange bestehen, wie ein Zugriff

[1] http://msdn2.microsoft.com/en-us/library/aa374815.aspx

auf das Objekt erhalten bleibt. Sie entscheidet im weiteren Lebensverlauf des Objekts welche Art von Zugriff gestattet sein soll. Bei Dateien wäre das Beispielsweise Lesen und Schreiben, aber auch Löschen und Ausführen gehört dazu. In Windows gibt es für jedes Objekt eine Vielzahl von Zugriffsmöglichkeiten. Für die bereits angesprochene Dateizugriffe wären das die folgenden Zugriffsrechte[2]

- FILE_ADD_FILE
- FILE_ADD_SUBDIRECTORY
- FILE_ALL_ACCESS
- FILE_APPEND_DATA
- FILE_CREATE_PIPE_INSTANCE
- FILE_DELETE_CHILD
- FILE_EXECUTE
- FILE_LIST_DIRECTORY
- FILE_READ_ATTRIBUTES
- FILE_READ_DATA
- FILE_READ_EA
- FILE_TRAVERSE
- FILE_WRITE_ATTRIBUTES
- FILE_WRITE_DATA
- FILE_WRITE_EA
- STANDARD_RIGHTS_READ
- STANDARD_RIGHTS_WRITE
- SYNCHRONIZE

Für andere Objekttypen sind die Rechte ähnlich aufgebaut, sowie mehr oder weniger in der Anzahl.
Eine Zugriffsmaske wird immer bei der Anforderung eines Objektes angegeben. Man spricht hier jedoch von „DesiredAccess“, was zum Ausdruck bringt, dass diese Zugriffsmaske die späteren Operationen widerspiegelt.

[2]Die Liste gibt es unter http://msdn2.microsoft.com/en-us/library/aa364399.aspx

```
HANDLE WINAPI CreateFile(
  __in          LPCTSTR lpFileName,
  __in          DWORD dwDesiredAccess,
...);
```

Um eine Datei mit CreateFile[3] zu öffnen oder zu erstellen muss neben dem Dateiname auch der gewünschte Zugriff angegeben werden. Soll die Datei kurz darauf geschrieben werden so muss *FILE_WRITE_DATA* angegeben werden. Sollen Attribute gelesen werden, so muss *FILE_READ_ATTRIBUTES* angegeben werden.
Diese Angaben wird in CreateFile auf Korrektheit überprüft. Kann die Datei nicht beschrieben werden, weil der Zugriff verweigert wird, so schlägt der Aufruf von CreateFile fehl. Ist der Aufruf erfolgreich so wird im späteren Verlauf der Zugriff kein weiteres mal überprüft, auch wenn der Zugriff nachträglich verweigert wird. Dazu muss die Datei geschlossen und erneut mit CreateFile geöffnet werden. Das resultierende Dateihandle wird nur verwendet, um zu überprüfen, ob eine Operation gemäß der gewünschten Zugriffsmaske (DesiredAccess) möglich ist. Ist *FILE_READ_DATA* angegeben, so kann kein ReadFile ausgeführt werden. Und ist *FILE_WRITE_DATA* angegeben, so kann kein WriteFile ausgeführt werden.

Das folgende Programm zeigt, dass ReadFile mit Fehler 5 (Zugriff verweigert) scheitert, wenn in CreateFile *FILE_WRITE_DATA* für den Parameter *dwDesiredAccess* verwendet wird. Würde man im Beispiel *FILE_READ_DATA* statt *FILE_WRITE_DATA* verwenden und genau im dem Moment zwischen CreateFile und ReadFile über die Dateizugriffsrechte den Lesezugriff verweigern, so würde ReadFile trotzdem funktionieren.

```
program CreateFileTest;
uses
  SysUtils,
  types,
  jwaWindows;

var h : THandle;
    c : Cardinal;
    b : array[0..10] of char;
begin
  SetLastError(0);
```

[3] http://msdn2.microsoft.com/en-us/library/aa363858.aspx

```
  h := CreateFile(
   'CreateFileTest.dpr', //__in          LPCTSTR lpFileName,
   FILE_WRITE_DATA,//__in           DWORD dwDesiredAccess,
   0,//__in            DWORD dwShareMode,
   nil,//__in             LPSECURITY_ATTRIBUTES
       lpSecurityAttributes,
   OPEN_EXISTING,//__in          DWORD dwCreationDisposition,
   0,//__in            DWORD dwFlagsAndAttributes,
   0//__in           HANDLE hTemplateFile
  );
  if h = INVALID_HANDLE_VALUE then
    RaiseLastOSError;

  if not ReadFile(h,Pointer(@b),sizeof(b),@c,nil) then
   RaiseLastOSError;

  CloseHandle(h);
end.
```

Dateizugriffsrechte werden nur einmalig beim Öffnen des Handles überprüft. Dateizugriffsmasken werden bei jeder Operation überprüft.
Der Typ der Zugriffsmaske ist ein 32 Bit Typ. Die verschiedenen Bits haben die folgende Bedeutung:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GR[a]	GW[b]	GE[c]	GA[d]	Reserviert			AS[e]	Standard Zugriffsrechte								objektspezifische ZugriffsRechte															

[a] Generic Read
[b] Generic Wead
[c] Generic Execute
[d] Generic All
[e] Zugriff auf die System ACL

Die **objektspezifischen Rechte** sind, wie der Name schon sagt für jedes Objekt definiert. Die Bits 0 bis 15 sind zwar immer dieselben, jedoch definiert jedes Objekt nur ein Teil der kombinierten Bits als gültige Zugriffsrechte. Theoretisch können so bis zu $2^{16} - 1 = 65535$ verschiedene Zugriffsrechte definiert werden.

Die **Standardzugriffsrechte** sind für alle Objekte immer dieselben:

DELETE	Das Recht ein Objekt zu löschen
READ_CONTROL	Das Recht Sicherheitsinformationen des Objekts zu lesen. Das schließt nicht die SYSTEM ACL ein.
SYNCHRONIZE	Das Recht das Objekt für Synchronisation zu nutzen.
WRITE_DAC	Das Recht die Sicherheitsinformation des Objekts zu ändern.
WRITE_OWNER	Das Recht den Besitzer des Objekts zu ändern.

Es gibt Konstanten, die die obigen Standardrechte kombinieren.

STANDARD_RIGHTS_ALL	Kombiniert DELETE, READ_CONTROL, WRITE_DAC, WRITE_OWNER, und SYNCHRONIZE Zugriff.
STANDARD_RIGHTS_EXECUTE	Aktuell READ_CONTROL
STANDARD_RIGHTS_READ	Aktuell READ_CONTROL
STANDARD_RIGHTS_REQUIRED	Kombiniert DELETE, READ_CONTROL, WRITE_DAC, und WRITE_OWNER Zugriff.
STANDARD_RIGHTS_WRITE	Aktuell READ_CONTROL

Die System ACL[4] (kurz SACL) oder auch Überwachungsliste[5] definiert Einträge, die eine Überwachung des Objektes ermöglichen. Diese Liste kann nur mit dem Privilege ACCESS_SYSTEM_SECURITY[6] geändert werden. Das Bit 24 (SA) bestimmt daher nicht, ob der Zugriff auf die SACL gestattet ist. Es wird nur verwendet, um den Zugriff auf die SACL zu überwachen.

Die Bits 28 bis 31 sind für die generischen Zugriffsrechte bestimmt. Sie werden im folgenden Kapitel besprochen.

2.1.2 Generic Mapping

„Generic Mapping“ bezeichnet die übersetzung der generischen Zugriffsrechte auf die Standard- und Spezialrechten. Es gibt die folgenden Generalrechte:

- GENERIC_ALL
- GENERIC_READ
- GENERIC_WRITE
- GENERIC_EXECUTE

2.1.3 Security Descriptor

2.1.4 Token

2.1.5 Privilege

2.2 AccessCheck Teil 2

[4] Access Control List engl. Zugriffsliste
[5] engl. Audit ACL
[6] Was ein Privileg ist kann im Kapitel Privileg nachgelesen werden. Wir kommen darauf nochmals zurück.