راهنـمای
SerialExpert
Version: 1.07

فهرست مطالب

۱. معرفی

SerialExpert یک فایل DLL استاندارد ویندوز است. به کمک توابع موجود در آن امکان بدست آوردن شماره سریال تمامی منابع ذخیره‌سازی بجز درایوهای نوری همچون هارد دیسک‌های داخلی، خارجی و فلش مموری‌ها فراهم شده است.

شماره سریال هارد دیسک منحصر به فرد بوده و معمولا روی برچسب هارد دیسک درج می‌شود. این شماره به صورت نرم‌افزاری قابل دستکاری و تغییر نیست. این سریال با پارتیشن‌بندی مجدد و یا فرمت درایوها تغییر نمی‌کند. بنابر این از این شماره می‌توان جهت تولید مشخصه سخت‌افزاری کامپیوتر استفاده کرد. و از اجرای نرم‌افزار کاربردی بر روی دیگر کامپیوترها جلوگیری کرد.

۲. مفهوم ایندکس

هر منبع ذخیره سازی با یک شماره که از صفر شروع می‌شود شناسایی می‌شود که به آن Index می‌گوییم. به اولین هارد دیسک داخلی روی کامپیوتر ایندکس صفر اختصاص می‌یابد. به هارد دیسک‌های داخلی بعدی ایندکس‌های یک به بعد اختصاص می‌یابد. بعد از آن منابع خارجی مانند هارد دیسک‌های خارجی و حافظه‌های فلش قرار خواهند داشت.

به هر حافظه USB که زودتر وصل سیستم شود، ایندکس پایین‌تر اختصاص داده می‌شود. فرض کنید دو حافظه فلش وصل سیستم شده‌اند و به ترتیب وصل شدن ایندکس‌های یک و دو به آنها اختصاص داده شده است. روی این کامپیوتر یک هارد دیسک اینترنال نیز وجود داشته که ایندکس صفر به آن اختصاص داده شده است. اگر اولین حافظه فلش که زودتر وصل سیستم شده است، خارج شود در ایندکس حافظه فلش دوم تغییری ایجاد نمی‌کند و همچنان ایندکس آن دو خواهد بود. ایندکس شماره یک در اینجا غیر معتبر خواهد بود. چنانچه هر حافظه دیگری وصل سیستم شود، ایندکس استفاده نشده‌ی یک به آن اختصاص داده خواهد شد.

۳. فراخوانی

به کمک API، LoadLibrary فایل se.dll فراخوانده می‌شود.

```
HMODULE hSE = LoadLibrary(TEXT("se.dll"));
```

چنانچه فایل se.dll در مسیر فایل اجرایی که آنرا فراخوانده است و یا در مسیر فولدر system32 از دایرکتوری ویندوز قرار داشته باشد، نیازی به وارد کردن مسیر فایل dll در API، LoadLibrary نیست.

۴. پیش الگوی توابع

از طریق se.dll هشت تابع در دسترس است. پیش الگوی این توابع به صورت زیر می‌باشد.

```
typedef VOID (__stdcall *FnSetLicense)(const char*);
typedef DWORD(__stdcall *FnGetDriveCount)();
typedef BOOL(__stdcall *FnGetDriveLettersA)(DWORD, char*);
typedef BOOL(__stdcall *FnGetDriveLettersW)(DWORD, wchar_t*);
typedef INT(__stdcall *FnGetDriveInfoA)(DWORD, char*, char*, char*);
typedef INT(__stdcall *FnGetDriveInfoW)(DWORD, wchar_t*, wchar_t*, wchar_t*);
typedef BOOL(__stdcall *FnIsVM)();
typedef VOID*(__stdcall *FnDecryptText)(VOID*, VOID*);
```

تعریف تابع SetLicense به صورت زیر خواهد بود.

```
FnSetLicense SetLicense = (FnSetLicense)GetProcAddress(hSE, "SetLicense");
SetLicense("your-license");
```

دیگر توابع نیز بصورت مشابه تعریف و فراخوانی می‌شوند.

۵. SetLicense

با استفاده از این تابع لایسنس ثبت می‌شود. تنها آرگومانی که این تابع دریافت می‌کند، رشته کاراکتری از نوع char می‌باشد و هیچ مقداری را برنمی‌گرداند.

```
SetLicense("your-license");
```

چنانچه License معتبری ثبت نشود، در هنگام دریافت اطلاعات دیسک، پیام 'نسخه ثبت نشده' نمایش داده خواهد شد.

۶. GetDriveCount

با استفاده از این تابع تعداد منابع ذخیره سازی بجز درایوهای نوری بدست می‌آید. این تابع هیچ آرگومانی دریافت نمی‌کند و تعداد منابع را بصورت DWORD برمی‌گرداند.

```
GetDriveCount();
```

مقداری که این تابع برمی‌گرداند آخرین ایندکس منابع نیست. مثلا اگر مقدار بازگشتی این تابع ۲ باشد. ممکن است ایندکس این دو منبع صفر و سه باشد. قبلا در قسمت مفهوم ایندکس علت متوالی نبودن ایندکس‌ها توضیح داده شده است.

۷. GetDriveLettersA

با استفاده از این تابع حروف مربوط به درایوها بدست خواهد آمد. مثلا اگر دیسک با ایندکس صفر، دو پارتیشن داشته باشد. آنگاه حروف مربوط به این پارتیشن‌ها احتمالا "C: D:" خواهد بود. این تابع دو آرگومان دریافت می‌کند. اولی ایندکس منبع از نوع DWORD و دومی اشاره‌گری به یک رشته کاراکتری از نوع char است که حاصل تابع در این متغییر قرار خواهد گرفت. اگر تابع با مشکلی مواجه شود که معمولا این مشکل بیشتر زمانی بوجود می‌آید که ایندکس وارد شده معتبر نباشد، مقدار بازگشتی تابع FALSE خواهد بود.

```
char szDL[80];
GetDriveLettersA(0, szDL);
```

این تابع بیشتر زمانی بکار می‌آید که کاربر اجازه انتخاب دیسک را داشته باشد. در اینصورت ساده‌ترین راهی که کاربر دیسک‌ها را از هم متمایز می‌کند، از روی حرف اختصاص داده شده به پارتیشن‌های دیسک خواهد بود.

۸. GetDriveLettersW

این تابع نسخه یونیکد GetDriveLettersA می‌باشد. و آرگومان دوم آن بجای اشاره‌گری از نوع char باید اشاره‌گری از نوع wchar_t باشد.

```
wchar t szDL[80];
GetDriveLettersW(0, szDL);
```

۹. GetDriveInfoA

این تابع مشخصات منبع ذخیره‌سازی را بدست می‌آورد. این تابع چهار آرگومان دریافت می‌کند. آرگومان اول از نوع DWORD بوده و مشخص کننده ایندکس منبع است و آرگومان‌های بعدی اشاره‌گری از نوع char هستند که به ترتیب شماره سریال، نوع منبع و مدل منبع را باز می‌گردانند.

```
char szSN[64], szType[64], szModel[64];
GetDriveInfoA(0, szSN, szType, szModel);
```

نوع منبع یکی از سه مورد "Fixed" برای هارد دیسک‌های داخلی، "External" برای هارد دیسک‌های خارجی و "Removable" برای فلش مموری‌ها می‌باشد.

این تابع در صورتی که نوع منبع هارد دیسک داخلی باشد عدد ۱، درصورتی که هارد دیسک خارجی باشد عدد ۲ و در صورتی که فلش مموری باشد عدد ۳ را باز می‌گرداند. اگر تابع با مشکلی مواجه شود که معمولا این مشکل بیشتر زمانی بوجود می‌آید که ایندکس وارد شده معتبر نباشد، مقدار بازگشتی تابع صفر خواهد بود.

۱۰. GetDriveInfoW

این تابع نسخه یونیکد GetDriveInfoA می‌باشد. و آرگومان دوم به بعد آن بجای اشاره‌گری از نوع char باید اشاره‌گری از نوع wchar_t باشد.

```
wchar_t szSN[64], szType[64], szModel[64];
GetDriveInfoW(0, szSN, szType, szModel);
```

۱۱. IsVM

این تابع هیچ آرگومانی دریافت نمی‌کند و در صورتی که مقدار TRUE را بازگرداند، به این معناست که برنامه شما بر روی یک ماشین مجازی مانند VMWare یا VirtualBox در حال اجراست. در این صورت تابع GetDriveInfo شماره سریال دیسک مجازی را بدست می‌آورد. این شماره سریال منحصر بفرد نبوده و به همین خاطر لازم است از اجرای برنامه خود بر روی ماشین مجازی جلوگیری کنید.

```
IsVM();
```

۱۲. DecryptText

این تابع رشته‌ی رمزگذاری شده را برمی‌گرداند. کافی است به ابتدای رشته ثابت مورد نظر SERIALEXPERTTXT_ را اضافه کنید. رمزگذاری پس از ساخت فایل اجرایی و توسط برنامه SEHideText انجام می‌شود. این تابع دو آرگومان که هردو اشاره‌گر هستند دریافت می‌کند و مقدار بازگشتی آن معادل آرگومان دوم می‌باشد. آرگومان دوم مقدار واقعی را نگه‌داری می‌کند. برای رمزگذاری رشته‌های یونیکد به ابتدای آن SETXTUC_ را اضافه کنید. رمزگذاری رشته‌های ثابتِ مهم باعث افزایش سردرگمی کرکرها می‌شود.

```
char szDecryptedA[12];
(char*)DecryptText("SERIALEXPERTTXT Sample Text", szDecryptesA);
wchar t szDecryptedW[12];
(wchar_t*)DecryptText(L"SETXTUC_Sample Text", szDecryptesW);
```

۱۳ . قفل نرم‌افزاری

با استفاده از شماره سریال هارد دیسک به عنوان مشخصه سخت‌افزاری می‌توان قفل نرم‌افزاری طراحی نمود. روش معمول برای این کار hash کردن مشخصه سخت‌افزاری با استفاده از یکی از الگوریتم‌های hash مانند SHA1 است. البته قبل از hash لازم است تعدادی کاراکتر ثابت که فقط برنامه‌نویس از آن مطلع است به ابتدا یا انتهای مشخصه نرم‌افزاری اضافه گردد. با انتخاب بخش‌هایی از نتیجه hash می‌توان Computer ID تولید کرد.

۱۴ . قفل سخت‌افزاری

از SerialExtert می‌توان جهت طراحی قفل سخت‌افزاری نیز استفاده کرد. در این حالت از فلش مموری بعنوان کلید استفاده می‌شود و شماره سریال فلش مموری بعنوان مشخصه منحصر بفرد عمل خواهد کرد.

اگر بخواهید Computer ID در زمان اجرای برنامه توسط کاربر تولید شود، لازم است به کاربر حق انتخاب یکی از فلش مموری‌های وصل شده به سیستم داده شود. با صفر تعریف کردن ایندکس و افزودن به آن می‌توانید منابع ذخیره‌سازی را پیمایش کنید. تعداد ایندکس‌های معتبر تشخیص داده شده نباید از مقداری که تابع GetDriveCount باز می‌گرداند بیشتر شود. چنانچه تابع GetDriveInfo مقدار ۳ را باز گرداند به این معناست که منبع فلش مموری است. مدل منبع و نیز حاصل تابع GetDriveLetters به کاربر در انتخاب فلش مموری مورد نظرش کمک می‌کند.

در هنگام بررسی صحت کد فعال سازی نیز لازم است تمامی منابع از ابتدا پیمایش شوند و نمی‌توان به ایندکسی که قبلا به فلش مموری مورد نظر اختصاص داده شده است اتکا کرد. این ایندکس با جدا شدن فلش مموری و وصل مجدد آن به سیستم ممکن است تغییر کند.

بخاطر داشته باشید فلش مموری برخلاف هارد دیسک‌های ثابت ممکن است در حین اجرای برنامه از سیستم قطع شوند، بنابر این لازم است علاوه بر ابتدای اجرای برنامه، حین اجرای برنامه نیز وجود آن بررسی شود.