一、简介

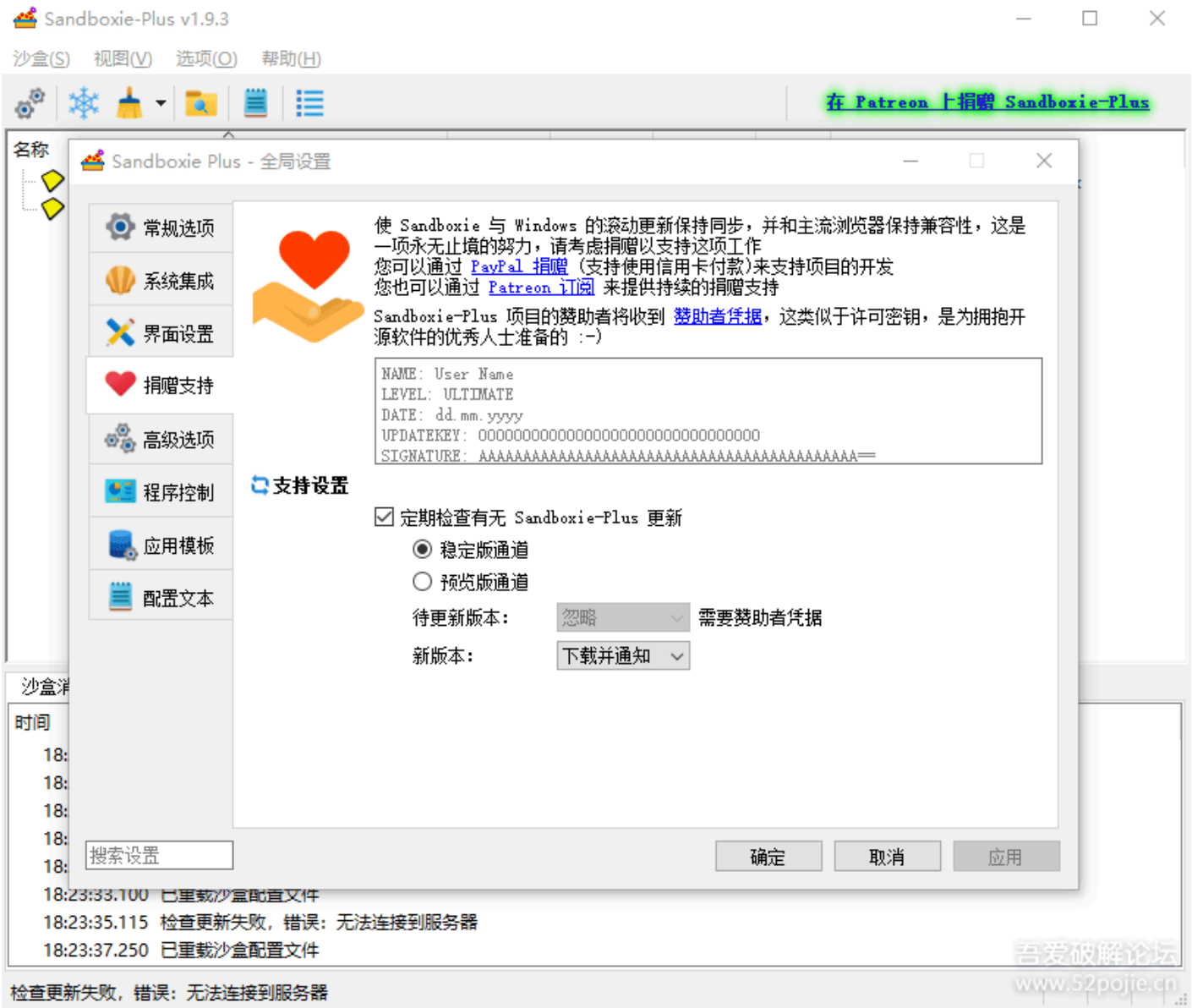
Sandboxie是Windows上的虚拟环境，可以用来测试不受信任的软件，类似轻量级的虚拟机，用于隔离安装流氓软件也十分方便

Sandboxie最初是收费的商业软件，后来停止开发并开源，现在由github的开发者David Xanatos维护，添加了许多新功能，称为Sandboxie Plus，但这些新功能需要向作者赞助获取激活码才能使用

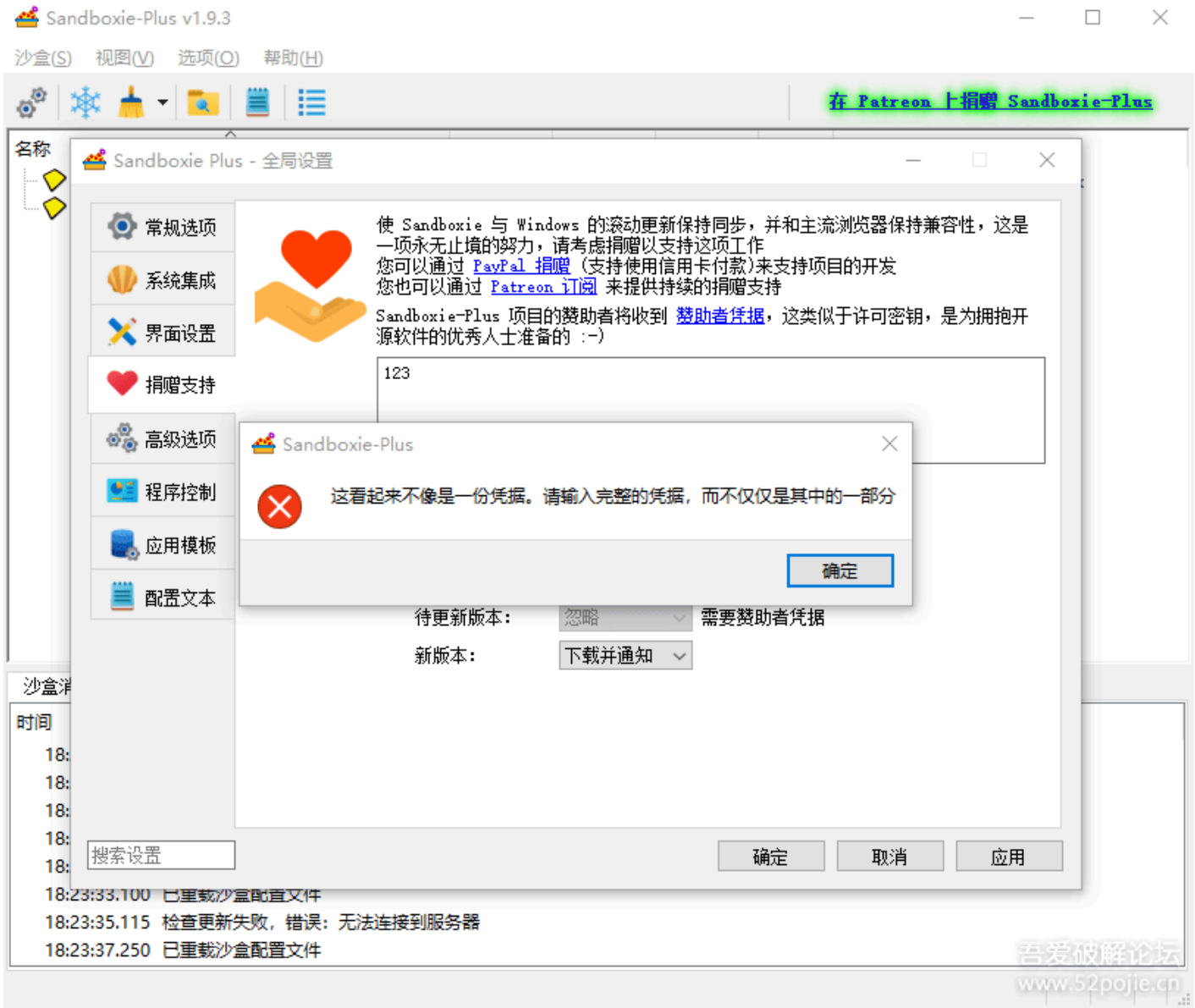
既然有源代码，不妨以此为例分析软件的注册机制，以及逆向思路

二、分析

首先下载Sandboxie Plus的安装包，安装后打开设置，看到以下捐赠界面：



先随便输入，弹出错误提示：



意思是格式不对，那么接下来就去源码中找到对应的部分。可以git clone到本地然后用Visual Studio打开，也可以用github自带的搜索，结果如图：

Sandboxie / SandboxiePlus / SandMan / **sandman_zh_CN.ts**

Code | Blame | 7416 lines (7393 loc) · 342 KB

```
4311            <source>Supporter certificate required</source>
4312            <oldsource>Supproter certificate required</oldsource>
4313            <translation>需要赞助者凭据</translation>
4314        </message>
4315        <message>
4316            <location filename="Windows/SettingsWindow.cpp" line="1112"/>
4317            <source>Run &amp;Un-Sandboxed</source>
4318            <translation>在沙盒外运行(&amp;U)</translation>
4319        </message>
4320        <message>
4321            <location filename="Windows/SettingsWindow.cpp" line="1371"/>
4322            <source>This does not look like a certificate. Please enter the entire certificate, not just a portion of it.</source>
4323            <translation>这看起来不像是一份凭据。请输入完整的凭据，而不仅仅是其中的一部分</translation>
4324        </message>
4325        <message>
4326            <location filename="Windows/SettingsWindow.cpp" line="1390"/>
4327            <source>This certificate is unfortunately expired.</source>
4328            <translation>很不幸此凭据已过期</translation>
4329        </message>
4330        <message>
4331            <location filename="Windows/SettingsWindow.cpp" line="1392"/>
4332            <source>This certificate is unfortunately outdated.</source>
4333            <translation>很不幸此凭据已过时</translation>
4334        </message>
4335        <message>
4336            <location filename="Windows/SettingsWindow.cpp" line="1395"/>
4337            <source>Thank you for supporting the development of Sandboxie-Plus.</source>
4338            <translation>感谢您对 Sandboxie-Plus 开发工作的支持</translation>
4339        </message>
4340        <message>
4341            <location filename="Windows/SettingsWindow.cpp" line="1402"/>
4342            <source>This support certificate is not valid.</source>
4343            <translation>此赞助者凭据无效</translation>
4344        </message>
```


注意到 `location` 标签包含了调用处的文件和行号，于是找到对应位置：

Sandboxie / SandboxiePlus / SandMan / Windows / SettingsWindow.cpp

Code | Blame | 1912 lines (1580 loc) · 66 KB

```
1344
1345    bool CSettingsWindow::ApplyCertificate(const QByteArray &Certificate, QWidget* widget)
1346    {
1347            QString CertPath = theAPI->GetSbiePath() + "\\Certificate.dat";
1348            if (!Certificate.isEmpty()) {
1349
1350                    auto Args = GetArguments(Certificate, L'\n', L':');
1351
1352                    bool bLooksOk = true;
1353                    if (Args.value("NAME").isEmpty()) // mandatory
1354                            bLooksOk = false;
1355                    //if (Args.value("UPDATEKEY").isEmpty())
1356                    //      bLooksOk = false;
1357                    if (Args.value("SIGNATURE").isEmpty()) // absolutely mandatory
1358                            bLooksOk = false;
1359
1360                    if (bLooksOk) {
1361                            QString TempPath = QDir::tempPath() + "/Sbie+Certificate.dat";
1362                            QFile CertFile(TempPath);
1363                            if (CertFile.open(QFile::WriteOnly)) {
1364                                    CertFile.write(Certificate);
1365                                    CertFile.close();
1366                            }
1367
1368                            WindowsMoveFile(TempPath.replace("/", "\\"), CertPath.replace("/", "\\"));
1369                    }
1370                    else {
1371                            QMessageBox::critical(widget, "Sandboxie-Plus", tr("This does not look like a certificate. Please enter the entire certificate, not just a portion of it."));
1372                            return false;
1373                    }
1374            }
1375            else if(!g_Certificate.isEmpty()){
1376                    WindowsMoveFile(CertPath.replace("/", "\\"), "");
1377            }
1378
1379            if (Certificate.isEmpty())
1380                    return false;
1381
1382            if (!theAPI->ReloadCert().IsError())
1383            {
1384                    g_FeatureFlags = theAPI->GetFeatureFlags();
1385                    g_Certificate = Certificate;
1386                    theGUI->UpdateCertState();
```


分析 ApplyCertificate 函数的逻辑，参数 Certificate 是一个字节数组，首先判断许可证是否为空，然后通过 GetArguments 函数解析为指定格式。下面是两个判断， NAME 和 SIGNATURE 不能为空，否则就弹出刚才的错误提示。如果格式正确，就把许可证写入文件。

继续看，发现下面调用了 theAPI->ReloadCert() 然后进行条件判断，意识到 ApplyCertificate 函数的作用只是对许可证的初步处理，验证逻辑并不在这里。

```
1381
1382            if (!theAPI->ReloadCert().IsError())
1383            {
1384                    g_FeatureFlags = theAPI->GetFeatureFlags();
1385                    g_Certificate = Certificate;
1386                    theGUI->UpdateCertState();
1387
1388                    if (g_CertInfo.expired || g_CertInfo.outdated) {
1389                            if(g_CertInfo.expired)
1390                                    QMessageBox::information(widget, "Sandboxie-Plus", tr("This certificate is unfortunately expired."));
1391                            else
1392                                    QMessageBox::information(widget, "Sandboxie-Plus", tr("This certificate is unfortunately outdated."));
1393                    }
1394                    else {
1395                            QMessageBox::information(widget, "Sandboxie-Plus", tr("Thank you for supporting the development of Sandboxie-Plus."));
1396                    }
1397
1398                    return true;
1399            }
1400            else
1401            {
1402                    QMessageBox::critical(widget, "Sandboxie-Plus", tr("This support certificate is not valid."));
1403
1404                    g_CertInfo.State = 0;
1405                    g_Certificate.clear();
1406                    return false;
1407            }
1408    }
1409
```

那么继续看 theAPI 是什么，然而文件中找不到定义，头文件也没有，最后发现在 stdafx.h 中（这一步用VS就方便的多）

Sandboxie / SandboxiePlus / SandMan / **stdafx.h**

Code | Blame | 141 lines (124 loc) · 2.75 KB

```
101  #include <QProgressBar>
102  #include <QInputDialog>
103  #include <QToolTip>
104  #include <QColorDialog>
105  #include <QToolButton>
106  #include <QScreen>
107  #include <QRadioButton>
108  #include <QGridLayout>
109  #include <QActionGroup>
110  #include <QStandardPaths>
111  #if QT_VERSION < QT_VERSION_CHECK(6, 0, 0)
112  #include <QDesktopWidget>
113  #endif
114
115  // other includes
116
117  //#define _T(x)      L ## x
118
119  #define STR2(X) #X
120  #define STR(X) STR2(X)
121
122  #define ARRSIZE(x)      (sizeof(x)/sizeof(x[0]))
123
124  #ifndef Max
125  #define Max(a,b)            (((a) > (b)) ? (a) : (b))
126  #endif
127
128  #ifndef Min
129  #define Min(a,b)            (((a) < (b)) ? (a) : (b))
130  #endif
131
132  #ifdef _DEBUG
133  #define SAFE_MODE
134  #endif
135
136  #include "../MiscHelpers/Common/DebugHelpers.h"
137
138  #define USE_QEXTWIDGETS
139
140  extern class CSettings*         theConf;
141  extern class CSbiePlusAPI*      theAPI;
```


原来它是 CSbiePlusAPI* 类型，但在这个类中没有 ReloadCert() 函数，于是到父类 CSbieAPI 中查找。在 SbieAPI.cpp 找到定义，只有一行，调用了 ReloadConf(SBIE_CONF_FLAG_RELOAD_CERT)，紧接着就是它的定义：

Sandboxie / SandboxiePlus / QSbieAPI / SbieAPI.cpp

Code Blame 2768 lines (2246 loc) · 75.2 KB

```
2075  SB_STATUS CSbieAPI::ReloadCert()
2076  {
2077          return ReloadConf(SBIE_CONF_FLAG_RELOAD_CERT);
2078  }
2079
2080  SB_STATUS CSbieAPI::ReloadConf(quint32 flags, quint32 SessionId)
2081  {
2082          __declspec(align(8)) ULONG64 parms[API_NUM_ARGS];
2083
2084          memset(parms, 0, sizeof(parms));
2085          parms[0] = API_RELOAD_CONF;
2086          parms[1] = SessionId;
2087          parms[2] = flags;
2088
2089          NTSTATUS status = m->IoControl(parms);
2090          if (!NT_SUCCESS(status))
2091                  return SB_ERR(status);
2092
2093          emit ConfigReloaded();
2094
2095          m_bBoxesDirty = true;
2096
2097          return SB_OK;
2098  }
```


大概是对参数进行处理，然后调用 `IoControl` ，继续跟进：

```
NTSTATUS IoControl(ULONG64 *parms)
{
        IO_STATUS_BLOCK IoStatusBlock;
        return NtDeviceIoControlFile(SbieApiHandle, NULL, NULL, NULL, &IoStatusBlock, API_SBIEDRV_CTLCODE, parms, sizeof(ULONG64) * API_NUM_ARGS, NULL, 0);
}
```


发现处理后的参数最终传入Windows的系统函数 `NtDeviceIoControlFile` ，这个函数有些陌生，上网查找它的用法，其中 `API_SBIEDRV_CTLCODE` 是关键，这是设备IO控制代码，用于指定要执行的具体操作。于是全局搜索这个名称，最后在驱动中找到了处理位置：

Sandboxie / Sandboxie / core / drv / api.c

Code Blame 1500 lines (1088 loc) · 39.7 KB

```
481     if (IoControlCode == API_SBIEDRV_CTLCODE) {
482
483         buf_len = InputBufferLength;
484         if (buf_len >= sizeof(ULONG64)
485                             && buf_len <= sizeof(ULONG64) * API_NUM_ARGS)
486             buf = InputBuffer;
487     }
488
489     if (! buf) {
490
491         IoStatus->Status = STATUS_INVALID_DEVICE_REQUEST;
492         return TRUE;
493     }
494
495     //
496     // find calling process
497     //
498
499     ApcsDisabled = KeAreApcsDisabled();
500
501     if (PsGetCurrentProcessId() == Api_ServiceProcessId)
502         proc = NULL;
503     else {
504
505         proc = Process_Find(NULL, NULL);
506         if (proc == PROCESS_TERMINATED) {
507
508             IoStatus->Status = STATUS_PROCESS_IS_TERMINATING;
509             return TRUE;
510         }
511     }
512
513     //
514     // capture parameter and call function
515     //
516
517     func_code = 0;
518     func_ptr = NULL;
519
520     __try {
521
522         ProbeForRead(
```


继续往下看，这里首先把参数 buf 赋值到 user_args ，第一个元素就是函数的编号， Api_Functions 是存放函数指针的数组，然后计算出下标，取出对应的函数指针。

```
520         __try {
521
522             ProbeForRead(
523                 buf, sizeof(ULONG64) * API_NUM_ARGS, sizeof(ULONG64));
524
525             memzero(user_args, sizeof(ULONG64) * API_NUM_ARGS);
526             memcpy(user_args, buf, buf_len);
527
528             func_code = (ULONG)user_args[0];
529
530             if (func_code > API_FIRST && func_code < API_LAST)
531                 func_ptr = Api_Functions[func_code - API_FIRST - 1];
532
533             if (func_ptr) {
534
535                 status = func_ptr(proc, user_args);
536
537             } else
538                 status = STATUS_INVALID_DEVICE_REQUEST;
539
540         } __except (EXCEPTION_EXECUTE_HANDLER) {
541             status = GetExceptionCode();
542         }
```


继续搜索，发现 Api_Functions 由 Api_SetFunction 函数初始化：

```
267     //---------------------------------------------------------------------------
268     // Api_SetFunction
269     //---------------------------------------------------------------------------
270
271
272     _FX void Api_SetFunction(ULONG func_code, P_Api_Function func_ptr)
273     {
274         if (! Api_Functions) {
275
276             ULONG len = (API_LAST - API_FIRST - 1) * sizeof(P_Api_Function);
277             Api_Functions = Mem_AllocEx(Driver_Pool, len, TRUE);
278
279             if (Api_Functions)
280                 memzero(Api_Functions, len);
281             else
282                 Api_Functions = (void *)-1;
283         }
284
285         if ((Api_Functions != (void *)-1) &&
286                 (func_code > API_FIRST) && (func_code < API_LAST)) {
287
288             Api_Functions[func_code - API_FIRST - 1] = func_ptr;
289         }
290     }
```


寻找调用处，原来在另一个文件中：

Sandboxie / Sandboxie / core / drv / **conf.c**

Code | Blame | 1734 lines (1313 loc) · 45.2 KB

```
1706        Conf_Read(-1);
1707
1708        //
1709        // set API functions
1710        //
1711
1712        Api_SetFunction(API_RELOAD_CONF,        Conf_Api_Reload);
1713        Api_SetFunction(API_QUERY_CONF,         Conf_Api_Query);
1714
1715        return TRUE;
1716    }
1717
```


这个 API_RELOAD_CONF 就是传给驱动的参数，它对应的函数指针是 Conf_Api_Reload ，发现它很长，但关键的地方在前几行：

```
1415    _FX NTSTATUS Conf_Api_Reload(PROCESS *proc, ULONG64 *parms)
1416    {
1417        NTSTATUS status;
1418        ULONG flags;
1419
1420        if (proc)
1421            return STATUS_NOT_IMPLEMENTED;
1422
1423        flags = (ULONG)parms[2];
1424
1425        if (flags & SBIE_CONF_FLAG_RELOAD_CERT) {
1426            status = MyValidateCertificate();
1427            goto finish;
1428        }
1429
1430        status = Conf_Read((ULONG)parms[1]);
1431
```


看到调用了 MyValidateCertificate() ，最后返回 status ，这很可能就是验证函数。紧接着搜索它的定义，发现它又调用了 KphValidateCertificate() ：

Sandboxie / Sandboxie / core / drv / **util.c**

Code | Blame | 472 lines (349 loc) · 13.8 KB

```
376 //---------------------------------------------------------------------------
377 // MyValidateCertificate
378 //---------------------------------------------------------------------------
379
380 BOOLEAN Driver_Certified = FALSE;
381
382 NTSTATUS KphValidateCertificate();
383
384 _FX NTSTATUS MyValidateCertificate(void)
385 {
386     NTSTATUS status = KphValidateCertificate();
387
388     Driver_Certified = NT_SUCCESS(status);
389
390     if (status == STATUS_ACCOUNT_EXPIRED)
391         status = STATUS_SUCCESS;
392
393     return status;
394 }
395
```


继续跟进，终于来到了真正的验证函数：

Sandboxie / Sandboxie / core / drv / verify.c

Code Blame 829 lines (677 loc) · 25.7 KB

```
503
504 _FX NTSTATUS KphValidateCertificate(void)
505 {
506     BOOLEAN CertDbg = FALSE;
507
508     static const WCHAR *path_cert = L"%s\\Certificate.dat";
509     NTSTATUS status;
510     ULONG path_len = 0;
511     WCHAR *path = NULL;
512     STREAM *stream = NULL;
513
514     MY_HASH_OBJ hashObj;
515     ULONG hashSize;
516     PUCHAR hash = NULL;
517     ULONG signatureSize = 0;
518     PUCHAR signature = NULL;
519
520     const int line_size = 1024 * sizeof(WCHAR);
521     WCHAR *line = NULL; //512 wchars
522     char *temp = NULL; //1024 chars, utf8 encoded
523     int line_num = 0;
524
525     WCHAR* type = NULL;
526     WCHAR* level = NULL;
527     //WCHAR* key = NULL;
528     LARGE_INTEGER cert_date = { 0 };
529
530     Verify_CertInfo.State = 0; // clear
531
532     if(!NT_SUCCESS(status = MyInitHash(&hashObj)))
533         goto CleanupExit;
534
535     path_len = wcslen(Driver_HomePathDos) * sizeof(WCHAR);
536     path_len += 64;     // room for \Certificate.ini
537     path = Mem_Alloc(Driver_Pool, path_len);
538     line = Mem_Alloc(Driver_Pool, line_size);
539     temp = Mem_Alloc(Driver_Pool, line_size);
540     if (!path || !line || !temp) {
541         status = STATUS_INSUFFICIENT_RESOURCES;
542         goto CleanupExit;
543     }
```


这个函数特别长，但是还得仔细看，因为它修改了一些全局变量，也就是有副作用，如果直接写 `return 0` 可能导致意外状况。看到它首先调用了 `MyInitHash` ，转到它的定义，分析一下，它的作用是初始化 `pHashObj` 的成员，其实都是指向算法函数的指针：

```
NTSTATUS MyInitHash(MY_HASH_OBJ* pHashObj)
{
    NTSTATUS status;
    ULONG hashObjectSize;
    ULONG querySize;
    memset(pHashObj, 0, sizeof(MY_HASH_OBJ));

    if (!NT_SUCCESS(status = BCryptOpenAlgorithmProvider(&pHashObj->algHandle, KPH_HASH_ALGORITHM, NULL, 0)))
        goto CleanupExit;

    if (!NT_SUCCESS(status = BCryptGetProperty(pHashObj->algHandle, BCRYPT_OBJECT_LENGTH, (PUCHAR)&hashObjectSize, sizeof(ULONG), &querySize, 0)))
        goto CleanupExit;

    pHashObj->object = ExAllocatePoolWithTag(PagedPool, hashObjectSize, 'vhpK');
    if (!pHashObj->object) {
        status = STATUS_INSUFFICIENT_RESOURCES;
        goto CleanupExit;
    }

    if (!NT_SUCCESS(status = BCryptCreateHash(pHashObj->algHandle, &pHashObj->handle, (PUCHAR)pHashObj->object, hashObjectSize, NULL, 0, 0)))
        goto CleanupExit;

CleanupExit:
    // on failure the caller must call MyFreeHash

    return status;
}
```


接着往下看，下一处验证在这里，功能是验证许可证有效期和类型，格式错误则验证失败：

```
            if (_wcsicmp(L"DATE", name) == 0 && cert_date.QuadPart == 0) {
                // DD.MM.YYYY
                KphParseDate(value, &cert_date);
            }
            else if (_wcsicmp(L"TYPE", name) == 0 && type == NULL) {
                // TYPE-LEVEL
                WCHAR* ptr = wcschr(value, L'-');
                if (ptr != NULL) {
                    *ptr++ = L'\0';
                    if(level == NULL) level = Mem_AllocString(Driver_Pool, ptr);
                }
                type = Mem_AllocString(Driver_Pool, value);
            }
            else if (_wcsicmp(L"LEVEL", name) == 0 && level == NULL) {
                level = Mem_AllocString(Driver_Pool, value);
            }
            //else if (_wcsicmp(L"UPDATEKEY", name) == 0 && key == NULL) {
            //    key = Mem_AllocString(Driver_Pool, value);
            //}
            else if (_wcsicmp(L"SOFTWARE", name) == 0) { // if software is specified it must be the right one
                if (_wcsicmp(value, SOFTWARE_NAME) != 0) {
                    status = STATUS_OBJECT_TYPE_MISMATCH;
                    goto CleanupExit;
                }
            }
```


至于许可证类型到底有哪些，别急，先往下分析。

```
680         if(!NT_SUCCESS(status = MyFinishHash(&hashObj, &hash, &hashSize)))
681             goto CleanupExit;
682 
683         if (!signature) {
684             status = STATUS_INVALID_SECURITY_DESCR;
685             goto CleanupExit;
686         }
687 
688         status = KphVerifySignature(hash, hashSize, signature, signatureSize);
689 
690         if (NT_SUCCESS(status)) {
691 
692             Verify_CertInfo.valid = 1;
693 
```


这里又出现了两个函数 MyFinishHash 和 KphVerifySignature ，分别查看定义：

```
95  NTSTATUS MyFinishHash(MY_HASH_OBJ* pHashObj, PVOID* Hash, PULONG HashSize)
96  {
97      NTSTATUS status;
98      ULONG querySize;
99 
100     if (!NT_SUCCESS(status = BCryptGetProperty(pHashObj->algHandle, BCRYPT_HASH_LENGTH, (PUCHAR)HashSize, sizeof(ULONG), &querySize, 0)))
101         goto CleanupExit;
102 
103     *Hash = ExAllocatePoolWithTag(PagedPool, *HashSize, 'vhpK');
104     if (!*Hash) {
105         status = STATUS_INSUFFICIENT_RESOURCES;
106         goto CleanupExit;
107     }
108 
109     if (!NT_SUCCESS(status = BCryptFinishHash(pHashObj->handle, (PUCHAR)*Hash, *HashSize, 0)))
110         goto CleanupExit;
111 
112     return STATUS_SUCCESS;
113 
114 CleanupExit:
115     if (*Hash) {
116         ExFreePoolWithTag(*Hash, 'vhpK');
117         *Hash = NULL;
118     }
119 
120     return status;
121 }
122 
```


经过分析，它的作用是计算许可证的hash，如果成功则返回 STATUS_SUCCESS 。但它只是完成了计算，校验不在这里，不需要修改。

而 KphVerifySignature 就是真正的验证函数了，分析这段代码，发现它是个纯算法函数，仅仅判断hash是否有效，而不创建或者修改外部资源，因此可以放心修改，方法是在第一行直接返回 STATUS_SUCCESS ：

```
217  NTSTATUS KphVerifySignature(
218      _In_ PVOID Hash,
219      _In_ ULONG HashSize,
220      _In_ PUCHAR Signature,
221      _In_ ULONG SignatureSize
222      )
223  {
224      return STATUS_SUCCESS;
225
226      NTSTATUS status;
227      BCRYPT_ALG_HANDLE signAlgHandle = NULL;
228      BCRYPT_KEY_HANDLE keyHandle = NULL;
229      PVOID hash = NULL;
230      ULONG hashSize;
231
232      // Import the trusted public key.
233
234      if (!NT_SUCCESS(status = BCryptOpenAlgorithmProvider(&signAlgHandle, KPH_SIGN_ALGORITHM, NULL, 0)))
235          goto CleanupExit;
236      if (!NT_SUCCESS(status = BCryptImportKeyPair(signAlgHandle, NULL, KPH_BLOB_PUBLIC, &keyHandle,
237          KphpTrustedPublicKey, sizeof(KphpTrustedPublicKey), 0)))
238      {
239          goto CleanupExit;
240      }
241
242      // Verify the hash.
243
244      if (!NT_SUCCESS(status = BCryptVerifySignature(keyHandle, NULL, Hash, HashSize, Signature,
245          SignatureSize, 0)))
246      {
247          goto CleanupExit;
248      }
249
250  CleanupExit:
251      if (keyHandle)
252          BCryptDestroyKey(keyHandle);
253      if (signAlgHandle)
254          BCryptCloseAlgorithmProvider(signAlgHandle, 0);
255
256      return status;
257  }
```


经过这么多判断之后，终于来到了 `Verify_CertInfo.valid = 1;` 这一行。经过以上分析，发现它就是除了返回值以外验证许可证的条件之一。如果只关注返回值，却没发现函数对外部变量的修改，验证依然会失败。在商业软件中，会用到更复杂的多个变量验证的方法，就是所谓的“暗桩”。

```
763             if (type && _wcsicmp(type, L"CONTRIBUTOR") == 0) {
764                 // forever - nothing to check here
765             }
766             else if (type && _wcsicmp(type, L"BUSINESS") == 0) {
767                 Verify_CertInfo.business = 1;
768                 if (level) { // in months
769                     TEST_EXPIRATION(0, (CSHORT)_wtoi(level), 0);
770                 }
771                 else { // 1 year default
772                     TEST_EXPIRATION(0, 0, 1);
773                 }
774             }
775             else if (type && _wcsicmp(type, L"EVALUATION") == 0) {
776                 Verify_CertInfo.evaluation = 1;
777                 // evaluation
778                 if (level) { // in days
779                     TEST_EXPIRATION((CSHORT)_wtoi(level), 0, 0);
780                 }
781                 else { // 5 days default
782                     TEST_EXPIRATION(5, 0, 0);
783                 }
784             }
785             else /*if (!type || _wcsicmp(type, L"PERSONAL") == 0 || _wcsicmp(type, L"PATREON") == 0 || _wcsicmp(type, L"SUPPORTER") == 0) */ {
786                 // persistent
787                 if (level && _wcsicmp(level, L"HUGE") == 0) {
788                     //
789                 }
790                 else if (level && _wcsicmp(level, L"LARGE") == 0 && cert_date.QuadPart < KphGetDate(1,04,2022)) { // valid for all builds released with 2 years
791                     TEST_CERT_DATE(0, 0, 2); // no real expiration just ui reminder - old certs
792                 }
793                 else if (level && _wcsicmp(level, L"LARGE") == 0) { // valid for all builds released with 2 years
794                     TEST_VALIDITY(0, 0, 2);
795                 }
796                 else if (level && _wcsicmp(level, L"MEDIUM") == 0) { // valid for all builds released with 1 year
797                     TEST_VALIDITY(0, 0, 1);
798                 }
799                 // subscriptions
800                 else if (level && _wcsicmp(level, L"TEST") == 0) { // test certificate 5 days only
801                     TEST_EXPIRATION(5, 0, 0);
802                 }
803                 else if (level && _wcsicmp(level, L"ENTRY") == 0) { // patreon entry level, first 3 months, later longer
804                     TEST_EXPIRATION(0, 3, 0);
805                 }
```


接下来是确定许可证的类型和有效期，代码逻辑很清晰，这时发现一个隐藏属性 CONTRIBUTOR ，能直接跳过有效期判断。去官网看了一下价格，相当于赞助1000欧获得的Huge Supporter Certificate，果然内鬼比土豪更可怕……

所以许可证的 `TYPE` 可以写 `CONTRIBUTOR` 或者 `PERSONAL-HUGE` ， `DATE` 都可以省略了。

回顾一下，实际上修改的只有一个函数。因为重点是分析和理解验证的流程，如果要实现任意输入都能通过验证，需要在多处进行修改，不利于初学者理解。

接下来就要验证是否破解成功，这里有个简便的方法，因为代码库在github上，而且配置了CI/CD，因此只要fork原仓库，提交自己的修改，等待自动构建完成，下载生成的Artifacts就可以了。这样就无需在本机花费大量时间和空间安装Qt，VS和WDK。如果想在本机编译，装好环境后用VS打开工程编译即可，可以参考readme，这里不详细讲述了。

三、验证

前面说过，为了让修改尽可能简洁，并不是随便输入都能通过验证。根据前面的分析，构造出如下许可证：

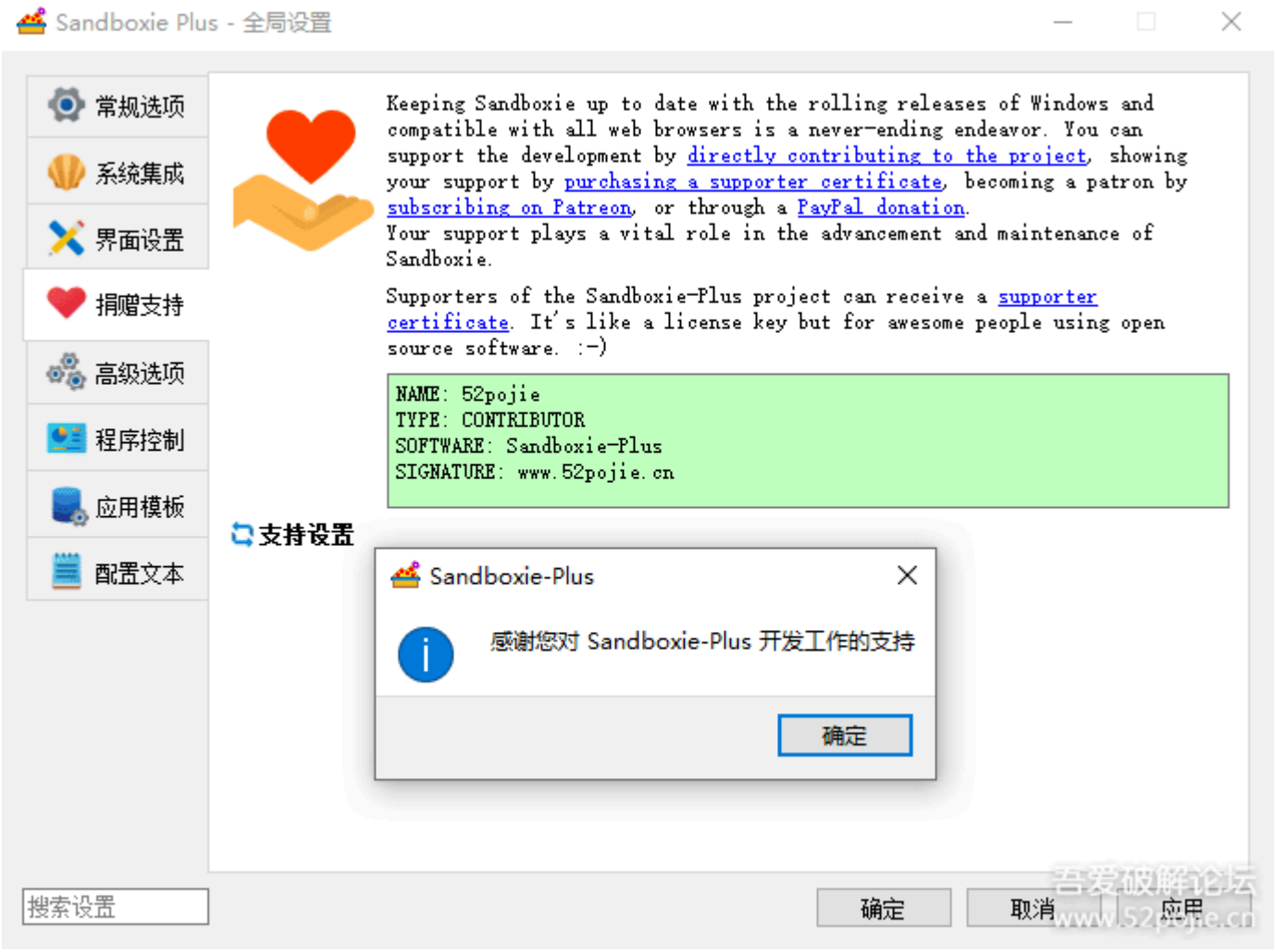
复制代码 隐藏代码

```
NAME: 52pojie  //任意
DATE: 01.04.2099  //非必需
TYPE: CONTRIBUTOR  //或PERSONAL-HUGE
SOFTWARE: Sandboxie-Plus
UPDATEKEY: 123456789  //非必需
SIGNATURE: www.52pojie.cn  //任意
```

最后还有一个问题，Windows内核驱动必须签名才能加载，签名是需要花钱从微软买的。这个问题确实没有什么好办法，所以可以猜到，作者为什么要把验证放到驱动层了吧。

一个临时方案是暂时关闭Windows的驱动签名验证，方法是以管理员身份打开命令行，输入 `bcdedit /set testsigning on` ，重启即可。但这带来一些问题，比如游戏反作弊系统检测到禁止驱动签名则不能启动，桌面右下角会有测试模式的水印，安全性下降等等。

先不管那么多，看看修改后的效果如何吧：



验证成功，所有功能都正常使用

备注一下，在 `verify.c` 中还发现两个名称相近的函数 `KphVerifyFile` 和 `KphVerifyCurrentProcess` ，逻辑与 `KphVerifySignature` 相似，暂时没发现具体作用，不过也进行了修改。许可证的 `SIGNATURE` 属性少于6个字符会验证失败，猜测与hash函数的计算方式有关，暂时没做进一步分析。

四、总结

这次分析的软件虽然验证方式简单，但从中能总结出常见的验证流程以及破解方法。通常，破解从验证提示处入手，一步步跟踪找到验证函数的位置，大体是这样的流程：

复制代码 隐藏代码

```
提示窗口 -> 验证函数1 -> ... -> 验证函数n -> 算法函数
```

而某个需要验证的功能函数是：

复制代码 隐藏代码

```
功能函数 -> 验证函数1 -> ... -> 验证函数n -> 算法函数
```

很多时候，输入许可证和调用功能，验证时中间过程是不一样的，但如果能找到并修改最底层的算法函数，那么无论验证的中间过程是什么样的，最后进行计算时总能得到我们想要的结果。就像能控制一加一等于几，那么再复杂的算法结果也能操纵，比起修改中间过程更加彻底。

假如不修改算法函数，那就必须逐一分析验证过程中调用链的每个函数。修改返回值有时候是不行的，因为它们常常有副作用，创建或修改了一些外部变量，然后在另一个隐蔽的地方进行验证导致失败。另外，输入许可证时和调用功能时的验证函数可能是不同的，但大多数时候默认两者相同或部分相同。而遇到防护严密或者验证过程不寻常的软件，这个假设就不成立了。

在实际应用中，分析反编译的代码比源代码困难的多，需要经验和技巧的积累。而Sandboxie Plus作为开源的付费软件，很适合新手用来学习和练手，了解常见验证机制的流程。

版权信息

本文涉及的源代码使用GPLv3授权，在以GPLv3发布的前提下可以任意修改，来自 `SandboxiePlus/LICENSE` ：

复制代码 隐藏代码

```
Sandboxie-Plus is made up of the following components, governed under various licenses:
* MiscHelpers a generic Qt based helper library, license under the LGPL.
* The Qt Framework which is license under the LGPL.
* SandMan the primary Sandboxie-Plus UI component, provided under a custom license.
* QSbieAPI a stand alone re implementation of sandboxie's API using IPC mechanisms to. cc
* Sandboxie core components, licensed under the GPL v3.
* UglobalHotkey is an extension for Qt framework, which implements global hotkeys functic
* QtSingleApp a Qt Solutions Component that provides support for applications that can be

ALL THIS SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```