SG Input 软件安全分析之逆向分析

hackedbylh / 2019-03-08 09:00:00 / 浏览数 1082 安全技术 漏洞分析 顶(0) 踩(0)

前言

通过本文介绍怎么对一个 windows 程序进行安全分析。分析的软件版本为 2018-10-9，所有相关文件的链接

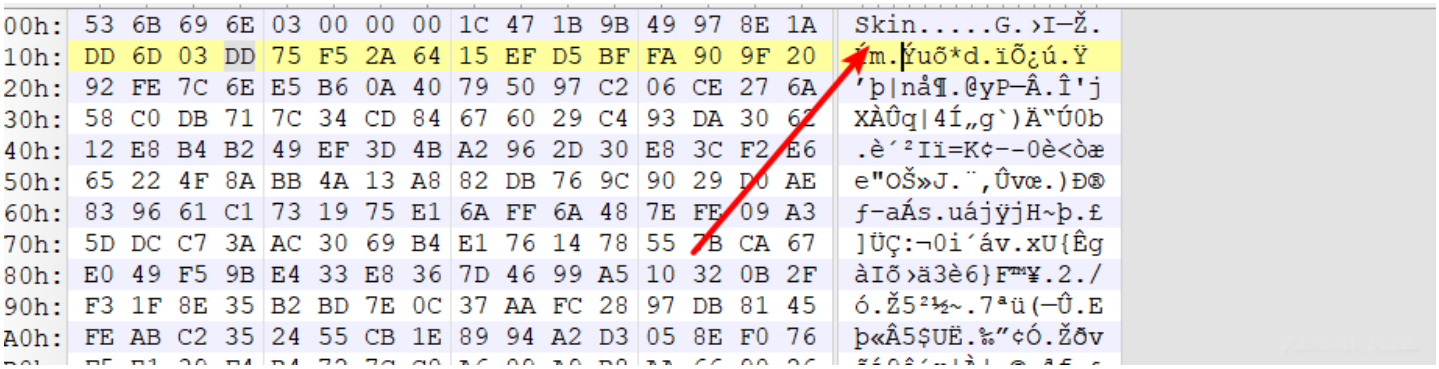
■■■https://pan.baidu.com/s/1l6BuuL-HPFdkFsVNOLpjUQ
■■■■erml

逆向分析

定位核心代码

拿到一个软件首先需要进行攻击面的探测，即找到尽可能多的可以与程序进行交互的入口点，有数据交互的地方就有可能会出现漏洞。首先对软件的功能做一个大概的了解，

先从官网随便下个皮肤，然后拿 010editor 简单看看能不能拿到一些有用的信息。



使用 binwalk 也没有识别出文件格式，于是猜测应该是输入法自己实现的格式。

后来在对皮肤相关的功能进行浏览的时候发现有皮肤编辑器这个软件

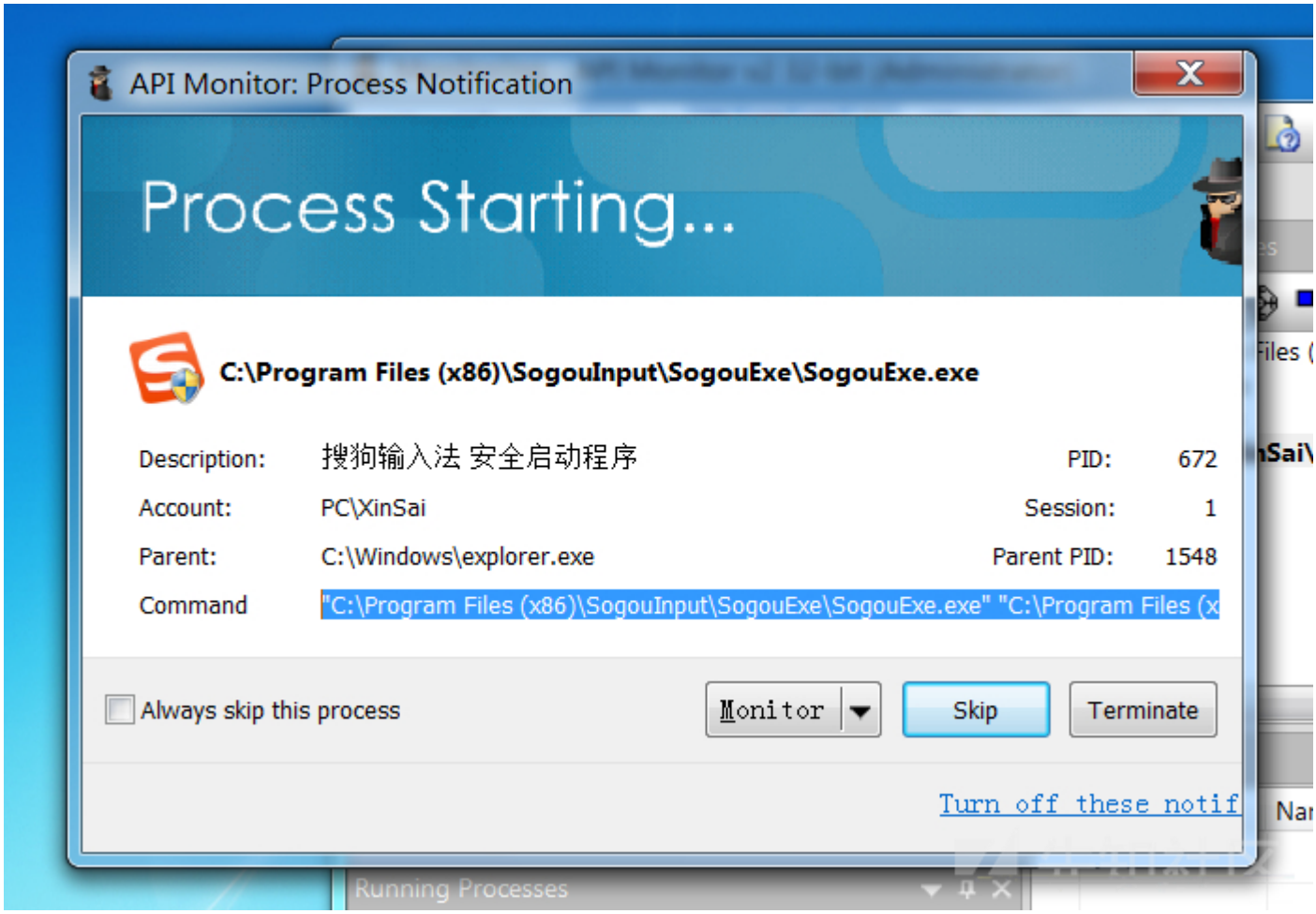
https://pinyin.sogou.com/skins/design.php

下载下来随便创建一个皮肤，发现此时的皮肤格式为 zip 格式，双击也能正常安装。皮肤编辑器的最近更新在 13 年，估计输入法是为了做兼容，同时支持两种格式的皮肤文件。

下载下来的皮肤双击就可以安装，这样的安装方式我们不好定位具体安装皮肤的程序，这时我们可以使用 api montor 监控当双击皮肤文件时系统所执行的命令，以便进行下一步的分析。

打开 api monitor ， 然后打开皮肤文件可以监控到搜狗输入法处理皮肤文件执行的命令

```
"C:\Program Files (x86)\SogouInput\SogouExe\SogouExe.exe" "C:\Program Files (x86)\SogouInput\9.1.0.2657\SGTool.exe"-line 0 -bo
```



通过使用 `Procmon.exe` 分析，其实最后调用

```
"C:\Program Files (x86)\SogouInput\9.1.0.2657\SGTool.exe" -line 0 -border --appid=skinreg -install -c  "C:\Users\XinSai\Deskto
```

通过执行这条命令就可以把皮肤安装到输入法内部。

下面把 `SGTool.exe` 拖到 IDA 里面， 使用命令行选项来搜索字符串的交叉引用去找到相关的处理代码。通过对命令行参数的交叉引用逐步向上追溯，在 `0x07A04D0` 发现程序会根据 `appid` 参数的值，决定下一步进行处理的函数

```
  return sub_8058E0(hInstance, a2, &v31);
}
if ( !_wcsicmp(&appid_1, L"skinreg") )        // 皮肤管理
{
  memset(&v31, 0, 0x800u);
  sub_7763F0(lpCmdLine, L"--appid=skinreg", &v31, 0x400u);
  return sub_780260(hInstance, a2, &v31);
}
if ( !_wcsicmp(&appid_1, L"scdreg") )
{
  memset(&v31, 0, 0x800u);
  sub_7763F0(lpCmdLine, L"--appid=scdreg", &v31, 0x400u);
  return sub_75F070(hInstance, a2, &v31);
}
```

然后调试发现一直断不到这，于是用 `drrun` 看看程序到底走了哪些路径

```
drrun.exe -t drcov --  "C:\Program Files (x86)\SogouInput\9.1.0.2657\SGTool.exe" -line 0 -border --appid=skinreg -install -c
```

发现直接点击 `ssf` 文件还是没有进入这个分支。

经过不断的尝试 + 使用一些监控软件，发现在关闭所有搜狗输入法相关进程的情况下，双击 `.ssf` 文件，会首先使用

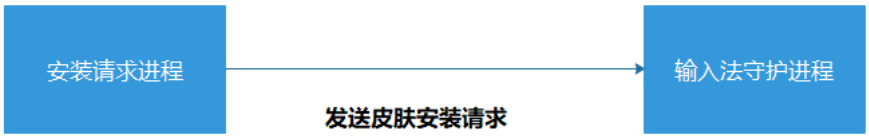
```
"C:\Program Files (x86)\SogouInput\9.1.0.2657\SGTool.exe" -daemon
```

开启一个类似于服务器的进程，然后在使用

```
"C:\Program Files (x86)\SogouInput\9.1.0.2657\SGTool.exe" -line 0 -border --appid=skinreg -install -c  "C:\Users\XinSai\Deskto
```

另外再起一个 `SGTool.exe` 的进程向刚刚启动服务进程发送消息，后续的皮肤处理在服务进程进行。猜测可能是使用了 `windows` 本地通信机制实现 C/S 架构

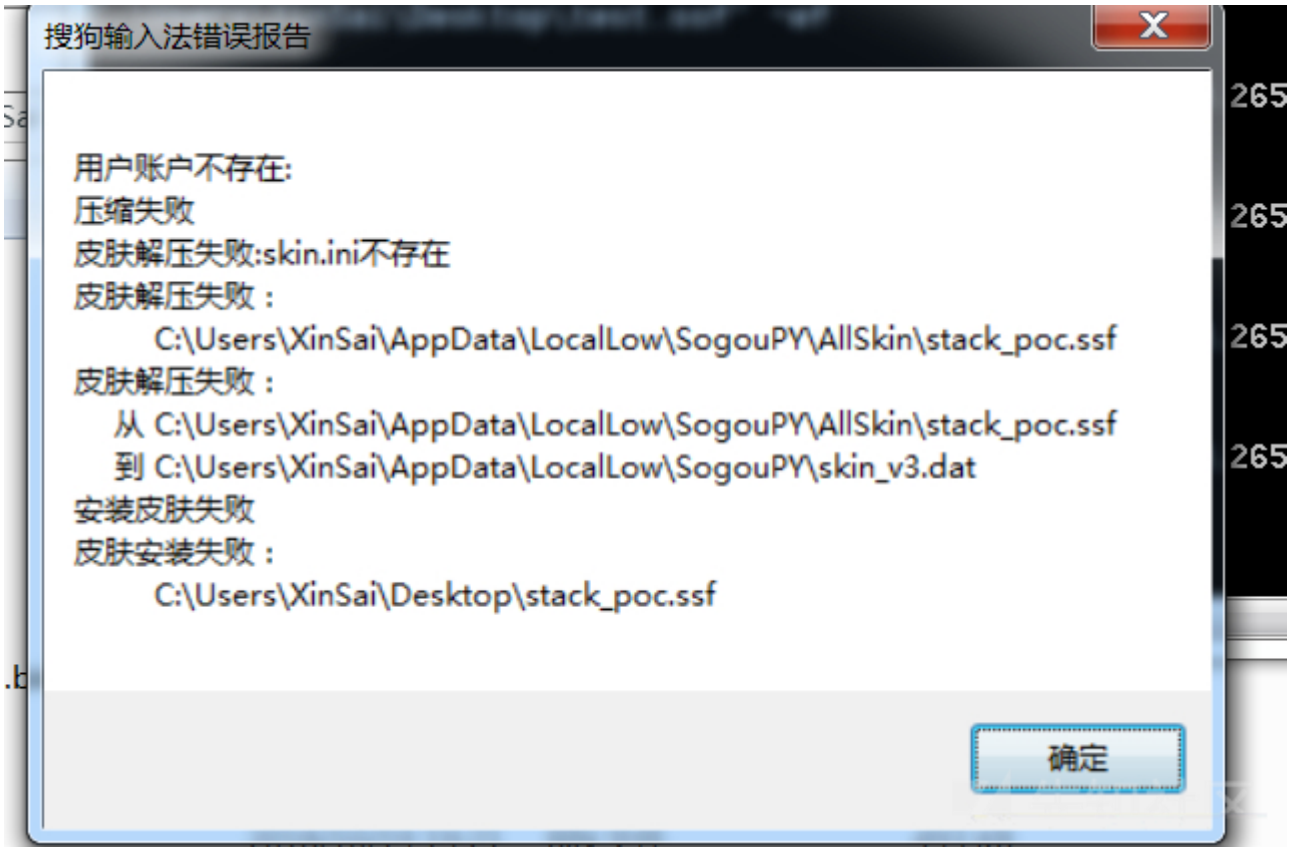
"C:\Program Files (x86)\SogouInput\9.1.0.2657\SGTool.exe" -line 0 -border --appid=skinreg -install -c "C:\Users\XinSai\Desktop\test.ssf" -q -ef



"C:\Program Files (x86)\SogouInput\9.1.0.2657\SGTool.exe" -daemon

后续在瞎

`.ssf` 后缀，然后双击会出现报错信息。

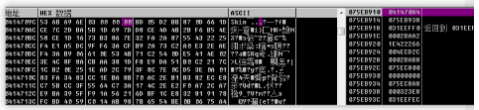


根据这些字符串在 od 里面找，可以找到一些信息，最后通过对

■■■■■■:skin.ini■■■

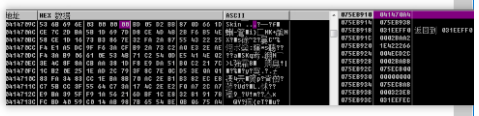
交叉引用, 然后不断回溯， 发现了一个有趣的函数 `0x07A72D0`

```
v14 = get_obj_string(&v76);
v15 = fill_string(&v77, L"确定要安装皮肤“”");
v16 = fill_string(v15, v14);
fill_string(v16, L"”吗？");
v17 = get_obj_string(&v77);
if ( MessageBoxW(0, v17, L"搜狗输入法 安装皮肤", 1u) == 1 )
{
  sub_47B1C0(&v77);
  sub_47B1C0(&v76);
  sub_47B1C0(&obj);
  LOBYTE(v80) = 0;
  sub_47C920(&v64);
LABEL_31:
  file_path = get_obj_string(&v78);
  sub_484260(&obj, file_path, 0, 0, 0, 0, 0);
  LOBYTE(v80) = 5;
  sub_4F49B0("t_envSkin");
  v20 = check_skin_version_and_decompress_by_ziputils(&obj);// 检测皮肤文件的版本，如果不是 Skin 文件就用 ziputi.dll 里面的函数解压
  LOBYTE(v80) = 0;
  v21 = v20;
```



这个函数首先 调用 0x7A84F0检测了某些参数。

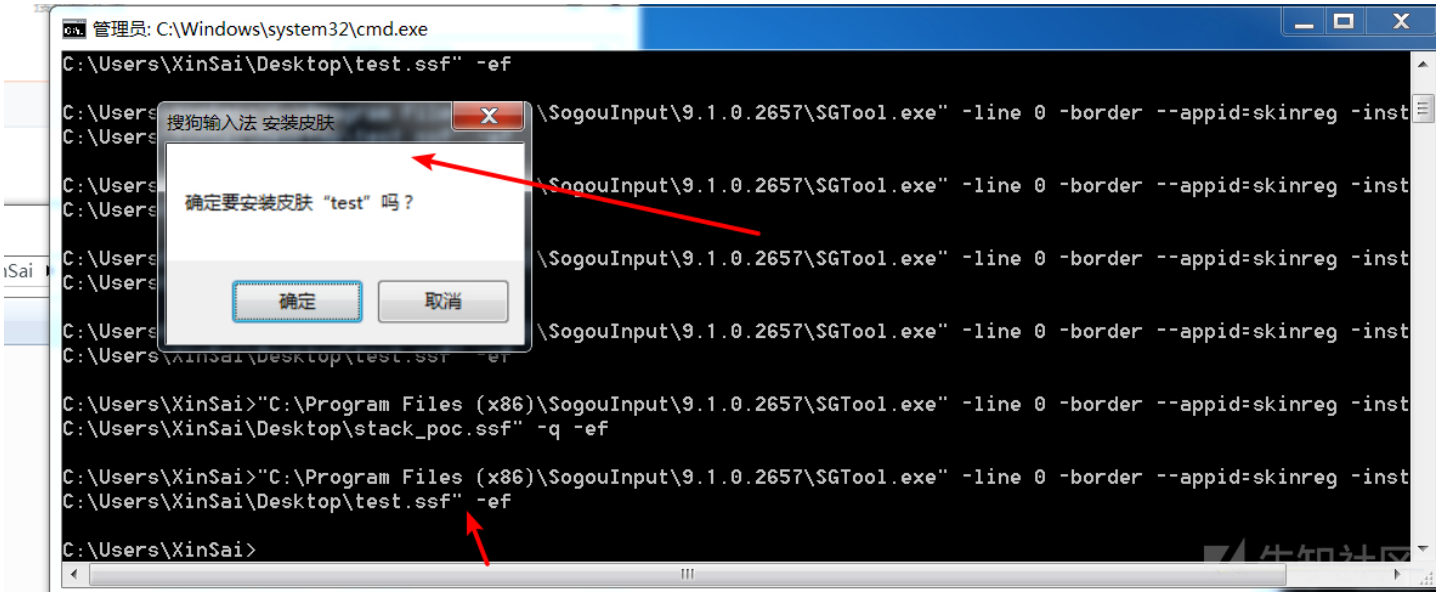
```
char __cdecl sub_7A84F0(int a1, int a2)
{
  int v2; // esi
  int v3; // eax
  int v5; // [esp+8h] [ebp-4h]

  if ( compare_skinreg_option(a1, a2, L"-q", 0, 0) )
    *(sub_7A5F80(16) + 8) = 1;
  if ( compare_skinreg_option(a1, a2, L"-b", 0, 0) )
    *(sub_7A5F80(16) + 13) = 1;
  if ( compare_skinreg_option(a1, a2, L"-d", 0, 0) )
    *(sub_7A5F80(16) + 14) = 1;
  if ( compare_skinreg_option(a1, a2, L"-c", 0, 0) )
    *(sub_7A5F80(16) + 15) = 1;
  if ( compare_skinreg_option(a1, a2, L"-h", 0, 0) )
    *(sub_7A5F80(16) + 312) = 2;
  if ( compare_skinreg_option(a1, a2, L"-hr", 0, 0) )
    *(sub_7A5F80(16) + 312) = 1;
  if ( compare_skinreg_option(a1, a2, L"-nf", 0, 0) )
    *(sub_7A5F80(16) + 324) = 0;
  if ( compare_skinreg_option(a1, a2, L"-nonotify", 0, 0) )
    *(sub_7A5F80(16) + 17) = 1;
  if ( compare_skinreg_option(a1, a2, L"-ef", 0, 0) )
    *(sub_7A5F80(16) + 18) = 1;
  v5 = -1;
  if ( compare_skinreg_option(a1, a2, L"-active", &v5, 0) )
  {
    if ( v5 >= a1 || v5 <= 0 )
003A78F0 sub_7A84F0:1 (7A84F0)
```



从我们的参数进行对比，猜测这里校验的应该是最后那两个参数

```
"C:\Program Files (x86)\SogouInput\9.1.0.2657\SGTool.exe" -line 0 -border --appid=skinreg -install -c
"C:\Users\XinSai\Desktop\test.ssf" -q -ef
```

然后猜测 -q 应该是安静模式，于是删掉 -q 试了一试



发现居然会有提示框，那这个提示框的代码附件应该离处理 皮肤文件的代码更加近了。

对字符串交叉引用，发现其实就是上面的那个代码（`0x007A75DC`）

```
fill_string(v16, L""吗？ ");
v17 = get_obj_string(&v77);
if ( MessageBoxW(0, v17, L"搜狗输入法 安装皮肤", 1u) == 1 )
{
  sub_47B1C0(&v77);
  sub_47B1C0(&v76);
  sub_47B1C0(&obj);
  LOBYTE(v80) = 0;
  sub_47C920(&v64);
LABEL_31:
  file_path = get_obj_string(&v78);
  sub_484260(&obj, file_path, 0, 0, 0, 0, 0);
  LOBYTE(v80) = 5;
  sub_4F49B0("t_envSkin");
  v20 = check_skin_version_and_decompress_by_ziputils(&obj);// 检测皮肤文件的版本，如果不是 Skin 文件就
  LOBYTE(v80) = 0;
  v21 = v20;
  sub_47B1C0(&obj);
  if ( v21 > dword_BE2C2C || v21 == 2 )
  {
    sub_7A8750(13, 0);
    sub_7A8470();
```

所以从这里开始，应该就开始对皮肤文件进行处理了。

分析皮肤处理相关的代码

经过不断的调试以及查看函数调用的参数，发现当我们点击 确认 的时候，会调用位于`0x914980` 的函数。

这个函数传入的参数是一个对象指针，对象内部有我们皮肤文件的路径，这个函数会对传入的 皮肤文件 进行第一次的判断。

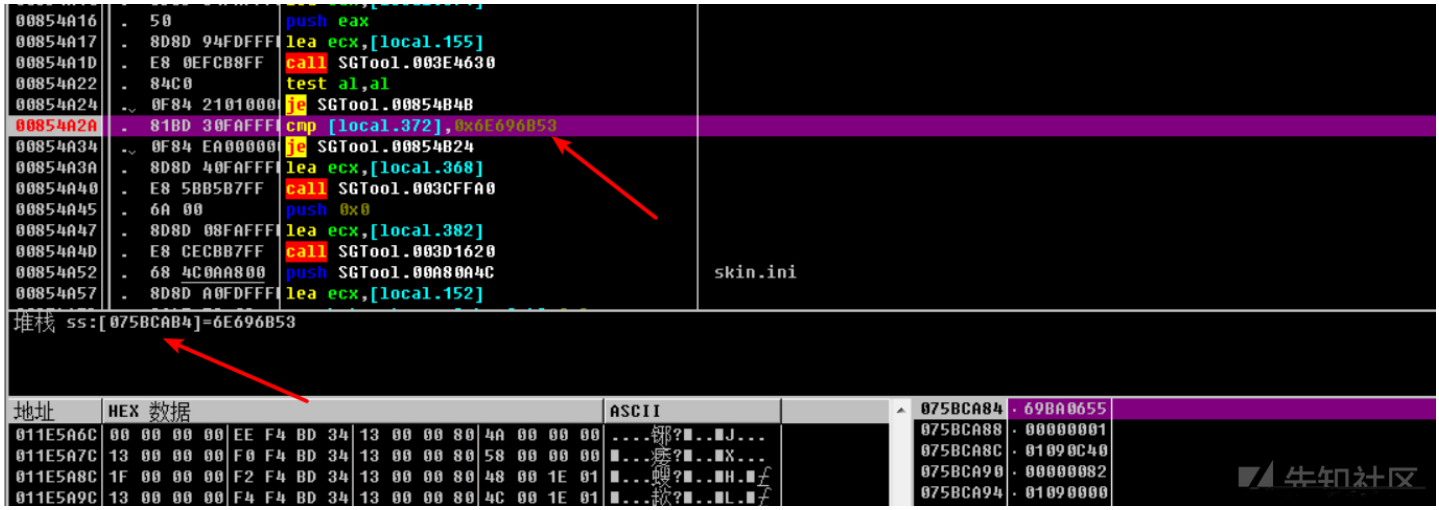
```
if ( MessageBoxW(0, v17, L"搜狗输入法 安装皮肤", 1u) == 1 )
{
  sub_47B1C0(&v77);
  sub_47B1C0(&v76);
  sub_47B1C0(&obj);
  LOBYTE(v80) = 0;
  sub_47C920(&v64);
BEL_31:
  file_path = get_obj_string(&v78);
  sub_484260(&obj, file_path, 0, 0, 0, 0, 0);
  LOBYTE(v80) = 5;
  sub_4F49B0("t_envSkin");
  v20 = check_skin_version_and_decompress_by_ziputils(&obj);// 检测皮肤文件的版本，如果不是 Skin 文件就用 ziputi.dll 里面的函数解压
  LOBYTE(v80) = 0;
  v21 = v20;
  sub_47B1C0(&obj);
  if ( v21 > dword_BE2C2C || v21 == 2 )
  {
    sub_7A8750(13, 0);
    sub_7A8470();
    v52 = malloc_for_obj(dwCreationDisposition);
    v53 = fill_string(v52, &unk_A6A83C);
```

函数首先会打开文件， 然后取出开头的4个字节作为文件类型，判断是不是Skin■如果是的话就认为是最新的皮肤格式然后进入后续的操作。

如果不是就认为是第一代皮肤格式 ，即用zip格式打包的皮肤。

```
  sub_496330(0);
  LOBYTE(v16) = 1;
  if ( !sub_4903D0(&obj, &dwCreationDisposition, fpath_obj) )// 首先打开文件
    goto LABEL_19;
  type = 0;
  if ( !get_skin_type(&v14, &dwCreationDisposition, &type) )// 获取文件的头4个字节，保存在 type
    goto LABEL_19;
  if ( type != 'nikS' )                         // 如果 type 不为 Skin 就认为是 第一代 皮肤包
  {                                             // 进入 zip 解压流程， 应该是 第一个版本的 皮肤包，按 zip 打包
    sub_48FFA0(&obj);
    sub_491620(0);
    LOBYTE(v16) = 2;
    sub_63E330(v15, L"skin.ini");
    LOBYTE(v16) = 3;
    init_some(&v9);
    LOBYTE(v16) = 4;
    if ( !decompress_skin_ini(v15, &v9, fpath_obj, &v8, 0) )// 调用 ziplib.dll 里面的函数对皮肤包进行解压
      goto LABEL_20;
    v2 = sub_4916E0(&v8, L"General", L"version");
    v3 = v2;
    if ( !v2 )
      goto LABEL_20;
    v4 = v2;
    v5 = v2 + 2;
```

调试时，可以看到 type 的值。 下面是打开的官网下载的皮肤文件，所以 type 为 Skin.



当皮肤文件的type值不为Skin时，程序会进入decompress_skin_ini (0x063F340), 这个函数里面会调用ziplib.dll 里面的函数对皮肤文件进行解压，提取并解析skin.ini文件。

```
  out[0] = 0;
  if ( v10 )
    v11 = *(v5 + 76);
  if ( !decompress_file_from_zip(obj, a2, input_file_path, (v11 + 4), &out[1], out) )// 第4个参数为 skin.ini
                                                // out[1] 保存 len
                                                // out[0] 保存 buf
                                                // 函数作用：提取出 zip包里面的 skin.ini
  {
    if ( a5 )
      *a5 = 6;
    if ( out_file_size )
      DeleteFileW(input_file_path);
    v12 = malloc_for_obj(a2);
    sprintf_to_obj(v12, L"%s", L"皮肤解压失败:skin.ini不存在");
    return 0;
  }
  decompress_content_ = out[1];
  if ( !parse_skin_ini_file(a4, out[0], out[1], 0x3A8) )// 解析 skin.ini
  {
    if ( out_file_size )
      DeleteFileW(input_file_path);
    if ( a5 )
      *a5 = 7;
    v14 = malloc_for_obj(a2);
    sprintf_to_obj(v14, L"%s", L"皮肤解压失败:skin.ini解析错误");
    return 0;
  }
0023E876 decompress_skin_ini:43 (63F476)
```

总的来说，0x914980 函数其实只是校验了皮肤文件的版本信息，对于 文件头 不是 Skin 的文件，则认为是第一代皮肤格式文件，然后会使用 ziplib.dll 里面的函数提取 skin.ini 文件并尝试解析它。

在进行完第一次的校验后，会回到 0x7A72D0, 将文件拷贝到用户的皮肤保存目录，然后对皮肤文件进行解析，提取出里面的文件。

```
    sub_47B1C0(&v68);
    if ( sub_485D00(dwCreationDisposition) )
    {
      if ( copy_skin_file_to_allskin(&v78, dwCreationDisposition, &v75, 0) )// 复制皮肤文件到 AllSkin 目录
      {
        saved_path = get_obj_string(&v75);     // 获取复制后的皮肤文件的路径
        sub_494010(saved_path, SE_FILE_OBJECT);
        sub_47B1C0(&obj);
        LOBYTE(v80) = 6;
        sub_47B1C0(&v77);
BEL_44:
        v32 = sub_7A5F80(16);
        if ( deal_skin(dwCreationDisposition, &v75, v66, 1, *(v32 + 324)) )// 处理皮肤文件，解压之类的
        {
          v34 = v66;
          init_some(&v62);
          LOBYTE(v80) = 16;
          sub_4E8B80(v56);
          LOBYTE(v80) = 17;
          sub_9126A0();
          if ( unknow(&v62, v56, v34) )
          {
```

其中 0x7A6230 deal_skin 就是解析皮肤文件的入口，它会调用 0x63E3F0 完成具体文件解析流程。

这个函数首先判断文件头 ， 如果是 Skin，表示为最新格式的皮肤文件

```
  if ( header == 'nikS' )                       // 判断文件头
  {
    sub_48FC20(&v67);
    LOBYTE(v78) = 0;
    sub_47C920(&dwCreationDisposition);
    if ( a7 )
      *a7 = 1;
    if ( a6 )
      *a6 = 0;
    v13 = file_obj + 12;
    if ( !*(file_obj + 284) )
      v13 = *(file_obj + 8);
    if ( decompress_skinv3(obj1, v65, (v13 + 4), 0) )// 对最新版本的 皮肤格式进行处理
                                                     // (v13 + 4) 为复制后的 ssf 皮肤文件的路径
    {
      v14 = 1;
    }
    else
    {
      if ( a6 )
        *a6 = 9;
      v14 = 0;
    }
    v73 = &t_str::`vftable';
    if ( !v76 && lpMem != &v75 )
      j___free_base(lpMem);
    return v14;
  }
```

则通过 decompress_skinv3 0x053B320 进行解析并提取出皮肤包里面包含的文件。否则就认为是第一代皮肤文件, 使用 ziplib 里面的函数, 把皮肤包里面的文件解压出来。

```
for ( i = v25 - 3; v28 < v27; ++v28 )
{
       ..............................................................
       ..............................................................
       ..............................................................
       ..............................................................

         if ( decompress_file_from_zip(v46, v65, &path, target, &len, &buf) )
         {
           v47 = len;
           if ( len <= 0x80000000 && len )
           {
             len = target;
             (*v51)(&v51, &len, 0);
             v48 = buf;
             v54 = v47;
             v53 = buf;
             sub_53ACE0(obj1, &v50, 0);// ■■■■■■■■
             sub_49F5C0(ziplib_obj, v65, v47, v48);
           }
           v24 = v62;
         }
       ..............................................................
       ..............................................................
       ..............................................................
       ..............................................................
}
```

下面对decompress_skinv3(0x053B320)进行分析，以便理解最新皮肤包的格式

```
  file_content = get_mem_by_obj(&v23, 0, file_size);
  if ( read_file_for_obj(&v16, dwCreationDisposition, file_content, file_size) )// 从文件中读取内容
  {
    v4 = 0;
    if ( handle_skinv3(obj, file_content, file_size) >= 0 )// 下面处理我们的输入数据
      v4 = 1;
  }
}
```

首先打开皮肤包，读取文件内容到内存，然后把文件内容，大小传给 handle_skinv3 0x0053A8C0 进行处理。

通过对这个函数的逆向，可以明白.ssf文件开头 8个字节的结构为 4字节的 Skin 和 4 字节的 version 字段

```
content_8 = (content + 1);                      // 忽略掉头 8 个字节
if ( size < 8 )
  return -1;
header = *content;
if ( header != 'nikS' )                         // 判断文件头，是否为 Skin
  return -1;
version = HIDWORD(header);                      // 这里取的是 Skin 后的4个字节
v3[12] = HIDWORD(header);
if ( version == 1 )
  goto LABEL_18;
if ( version == 2 || version < 3 || version > dword_BE2C2C )// 只处理 v3 的 Skin
  return -1;
sub_4EE3D0(&obj, version);
LOBYTE(v32) = 1;
v9 = get_mem_by_obj(v8, 0, size - 8);
HIDWORD(content_8_) = content_8;
buf_ = v9;
new_size = size - 8;
v23 = v9;
LODWORD(content_8_) = &new_size;
```

校验完版本信息后， 会对文件头部后的数据使用程序自己实现的算法进行解码，然后对解码之后的数据使用zlib再次解码。

```
obj_ = sub_53BCF0();
if ( !decode_data_to_zlib(obj_, buf, content_8_, size - 8) )// 调用的参数为:  obj,buf,file_content+8 file_size-8 .
                                                // 会把文件内容 + 8 开始的数据解码，解码后的数据为 zlib 格式，  保存在 buf。
  goto LABEL_29;
size_4 = new_size - 4;
decoded_data_size = *buf_;                      // 从解码后的数据里，取开头4字节
                                                // 貌似是新的 size
size_ = decoded_data_size + 8;
buffer = get_mem_by_obj(&lpMem, 0, decoded_data_size + 8);
v16 = header;                                   // 取到原始的头，  Skin
*(buffer + 1) = HIDWORD(header);
buf__ = v23;
*buffer = v16;                                  // 到现在为止，在新分配的内存开始，设置好了 Skin 的头部字段，  8 个字节 Skin + ver
if ( zlib_decompress((buffer + 1), &decoded_data_size, buf__ + 4, size_4) )// 对刚 解码过的 zlib 数据，zlib 解压，
                                                // buf__ 为 刚刚解码后的数据.
                                                // 解压后的数据保存在 buffer + 8 出
                                                // 解压后的长度保存在 decoded_data_size
{
ABEL_29:
  sub_482C20(&obj);
  goto LABEL_10;
```

zlib 解码完成后，把解码后的数据传入 0x53bf50 （ 调试时确认 ）对文件进行提取

```
{
ABEL_11:
  if ( !v31 && v5 != &v29 )
    j___free_base(v5);
  return -1;
}
v18 = *v21;
v21[12] = HIDWORD(header);
v19 = (*v18)((buffer + 1), size_ - 8, (v21 + 6));// 对 zlib 解压过的数据进行下一步的处理
                                                // 调用的函数为 0x53bf50L
                                                // 传入的参数为 zlib解压后的数据，  解压后数据的长度
                                                // 从zlib解码后的数据里面提取出文件
if ( v19 < 0 )
{
ABEL_10:
```

这个函数对传入的数据进行解析，提取出相关的文件

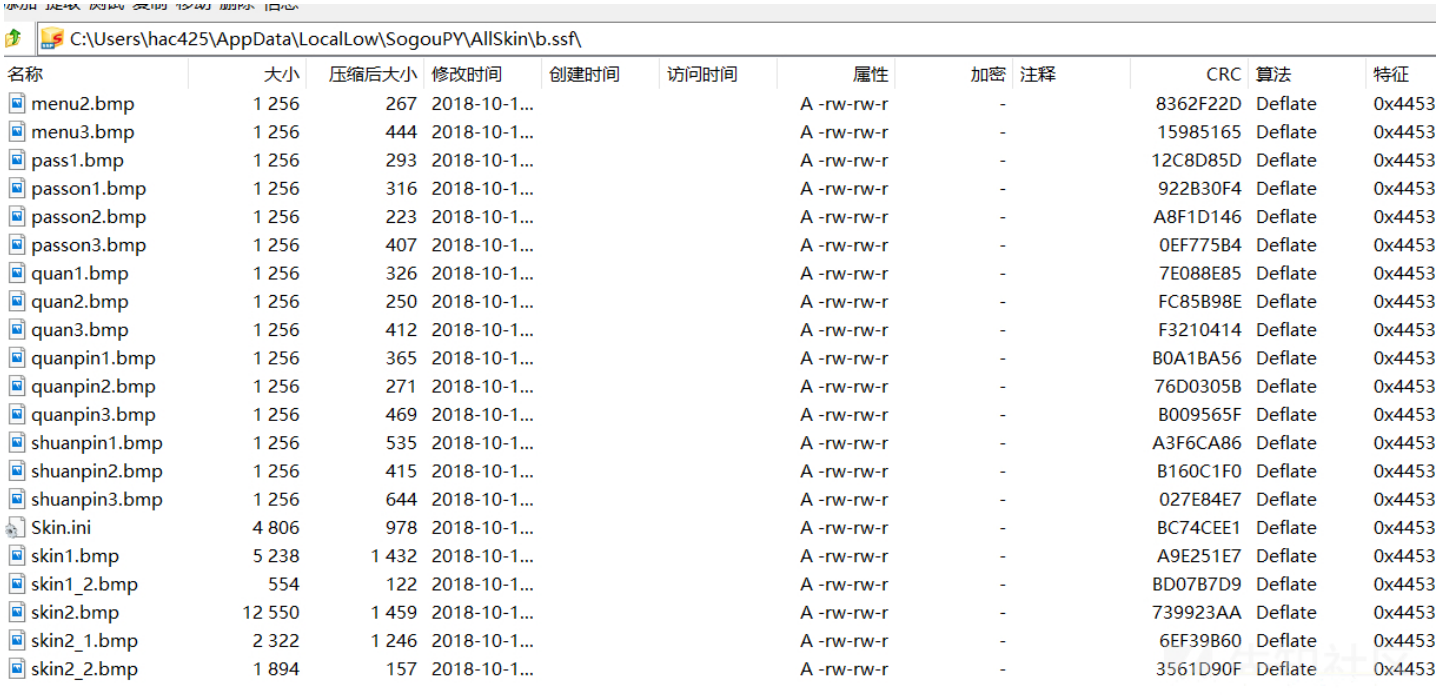
```
// ■■■ 0x53AAE1
unsigned int __thiscall skin_v3_step2(int this, unsigned int *buf, unsigned int size, int a4)
{

len = *buf;                                     // ■■ buf ■■■ 4 ■■■, ■■■■■■■■
if ( *buf > size )
  return -1;
new_size = size - 4;
```

```
sizea = size - 4;
if ( *buf )
{
  if ( (**(this + 4))(this + 4, buf + 1, new_size, a4) < 0 )// ■■0x53c110L■■ 2■ 3■■■■ buf + 4, size-4
                                          // ■■■■■■■4■■■
                                          // ■■■■■■■ zlib ■■■■■■■ 0x8-0x40 ■■■■■■

      do                                  // ■■■■ zlib ■■■■■■■■■■■■■■
      {
        offset = *(v12 + 4 * v9);
        if ( offset < data_len )
        {

          v17 = (**v15)(cur, sizea, a4);      // ■■ 0x923f70L, ■■■■■■
          if ( v17 < 0 )
            goto LABEL_19;
          v18 = &cur[v17];
          v19 = (*v22[3])(v18, sizea - v17, a4);// ■■■■ 0x53c1c0L■ ■■■■■■■

        }
        ++v9;
      }
      while ( v9 < v11 );
```

文件格式汇总

老版本的皮肤格式

其实就是一个 zip 包， 里面有配置文件和一些图片。

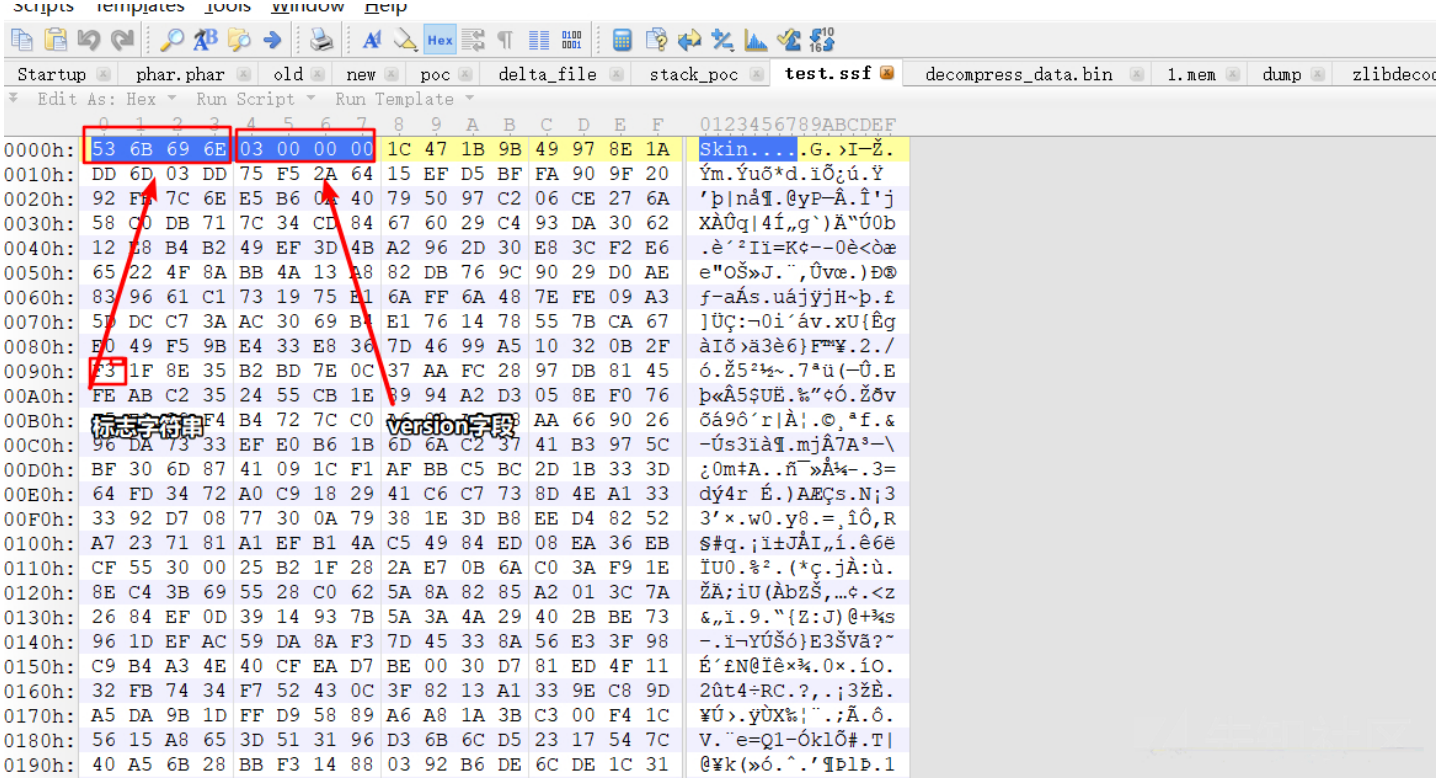


C:\Users\hac425\AppData\LocalLow\SogouPY\AllSkin\b.ssf\

名称	大小	压缩后大小	修改时间	创建时间	访问时间	属性	加密	注释	CRC	算法	特征
menu2.bmp	1 256	267	2018-10-1...			A -rw-rw-r	-		8362F22D	Deflate	0x4453
menu3.bmp	1 256	444	2018-10-1...			A -rw-rw-r	-		15985165	Deflate	0x4453
pass1.bmp	1 256	293	2018-10-1...			A -rw-rw-r	-		12C8D85D	Deflate	0x4453
passon1.bmp	1 256	316	2018-10-1...			A -rw-rw-r	-		922B30F4	Deflate	0x4453
passon2.bmp	1 256	223	2018-10-1...			A -rw-rw-r	-		A8F1D146	Deflate	0x4453
passon3.bmp	1 256	407	2018-10-1...			A -rw-rw-r	-		0EF775B4	Deflate	0x4453
quan1.bmp	1 256	326	2018-10-1...			A -rw-rw-r	-		7E088E85	Deflate	0x4453
quan2.bmp	1 256	250	2018-10-1...			A -rw-rw-r	-		FC85B98E	Deflate	0x4453
quan3.bmp	1 256	412	2018-10-1...			A -rw-rw-r	-		F3210414	Deflate	0x4453
quanpin1.bmp	1 256	365	2018-10-1...			A -rw-rw-r	-		B0A1BA56	Deflate	0x4453
quanpin2.bmp	1 256	271	2018-10-1...			A -rw-rw-r	-		76D0305B	Deflate	0x4453
quanpin3.bmp	1 256	469	2018-10-1...			A -rw-rw-r	-		B009565F	Deflate	0x4453
shuanpin1.bmp	1 256	535	2018-10-1...			A -rw-rw-r	-		A3F6CA86	Deflate	0x4453
shuanpin2.bmp	1 256	415	2018-10-1...			A -rw-rw-r	-		B160C1F0	Deflate	0x4453
shuanpin3.bmp	1 256	644	2018-10-1...			A -rw-rw-r	-		027E84E7	Deflate	0x4453
Skin.ini	4 806	978	2018-10-1...			A -rw-rw-r	-		BC74CEE1	Deflate	0x4453
skin1.bmp	5 238	1 432	2018-10-1...			A -rw-rw-r	-		A9E251E7	Deflate	0x4453
skin1_2.bmp	554	122	2018-10-1...			A -rw-rw-r	-		BD07B7D9	Deflate	0x4453
skin2.bmp	12 550	1 459	2018-10-1...			A -rw-rw-r	-		739923AA	Deflate	0x4453
skin2_1.bmp	2 322	1 246	2018-10-1...			A -rw-rw-r	-		6EF39B60	Deflate	0x4453
skin2_2.bmp	1 894	157	2018-10-1...			A -rw-rw-r	-		3561D90F	Deflate	0x4453

新版本的皮肤格式

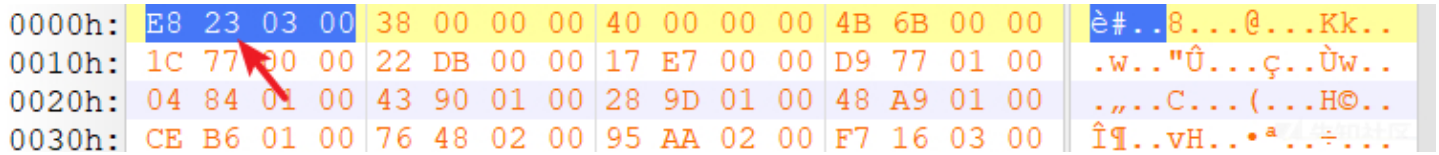
首先是 .ssf 文件
开头 8个字节为 4字节的 Skin 和 4 字节的 version



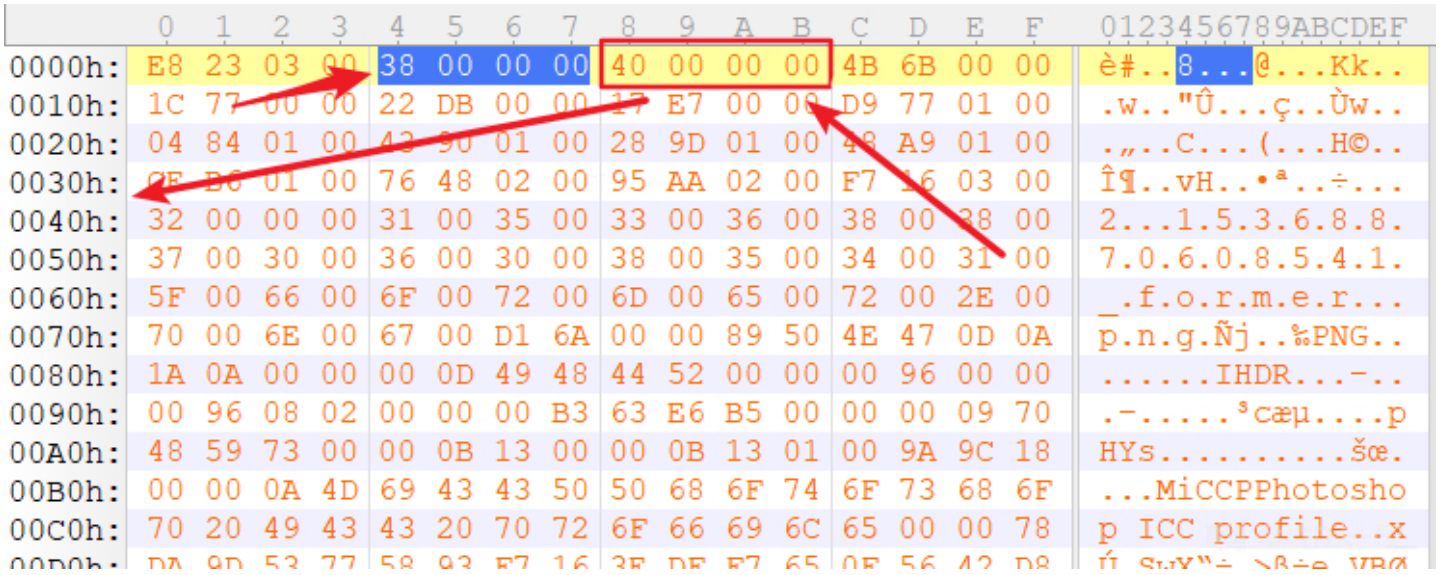
- 然后调用 0x0639610 对 文件偏移 8开始进行解码， 解码后的数据 A 偏移 4 字节开始 为 zlib 压缩的数据。
- 然后对 A + 4 使用 zlib 解码，得到 zlib 解码后的数据 B
- 然后把 B 和 B 的长度传入 0x53bf50 继续处理

B 的结构为

开头4个字节为数据的总长度



然后根据 0x53C110 ， 后面紧跟着的 4 个字节为文件映射表的长度，即 0x38
字节。每一个表项4个字节，代表表项指示其所表示的文件在整个文件中的偏移地址。



如图所示， 第一个表项的值为 0x40 ， 所以第一个文件应该在 0x40 处。

经过一定的观察发现一个文件的表示方式为

- 4字节： 文件名的长度
- 文件名(unicode 编码)

- 4字节： 文件数据的长度
- 文件数据

发现漏洞

分析完整个皮肤处理的代码后发现整个代码的逻辑还是不怎么复杂的。于是可以直接读反编译的代码来找找看是否存在什么漏洞。在读代码找漏洞时重点关注缓冲区的操作，内存的分配大小以及对文件中表示长度的字段的使用是否合理。

经过一番仔细的走查发现在对皮肤文件第一步用自己实现的解密算法解密后的开始 4 个字节为 `deocded_data_size` ， 之后会把它加上 8 然后去分配内存。

```
  buf = ptr;
  obj_ = sub_53BCF0();
  if ( !decode_data_to_zlib(obj_, buf, &new_size, content_off_8, fsize_sub_8) )// 调用的参数为： obj,buf,file_content+8 file_size-8
                                                // 会把文件内容 + 8 开始的数据解码，解码后的数据为 zlib 格式， 保存在 buf。
    goto LABEL_29;
  size_4 = new_size - 4;
  decoded_data_size = *buf_;                    // 通过解码后的数据的开始4字节，确定 zlib 解压的数据大小， 然后进行内存分配
                                                // 这个 size 通过动态调试，应该是动态解密出来的
  v6 = decoded_data_size + 8;
  v4 = get_mem_by_obj(&lpMem, 0, decoded_data_size + 8);
  v16 = header;                                 // 取到原始的头， Skin
  *(v4 + 1) = HIDWORD(header);
  buf__ = v24;
  *v4 = v16;                                    // 到现在为止，在新分配的内存开始，设置好了 Skin 的头部字段， 8 个字节 Skin + version
  if ( zlib_decompress((v4 + 1), &decoded_data_size, buf__ + 4, size_4) )// 对刚 解码过的 zlib 数据，zlib 解压，
                                                // buf__ 为 刚刚解码后的数据.
                                                // 解压后的数据保存在 buffer + 8 出
                                                // 解压后的长度保存在 decoded_data_size
  {
LABEL_29:
```

`deocded_data_size` 是从解密后的文件中取出的，当把 `deocded_data_size` 改成 `0xffffffff`
时，在分配内存时会整数溢出导致分配比较小的内存块，然后后续的代码在使用这个缓冲区时会造成一个堆溢出。漏洞已于3个月前提交并修复。

总结

在分析软件功能实现时，可以采用一些监控软件比如 `api monitor`
来辅助定位关键代码。一些程序中的提示，报错信息也可以用来定位。最重要的就是多调试，多调试。程序从文件内容中取 `size` 时要注意校验。

上一篇：一个JS沙箱逃逸漏洞 下一篇：session，cookie认证会...

1. 0 条回复
   - 动动手指，沙发就是你的了！

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板