Hex-Rays Decompiler 7.4

Better array detection

The text produced by v7.3 is not quite correct because the array at [ebp-128] was not recognized. Overall determining the array is a tough task but we can handle simple cases automatically now.

PSEUDOCODE V7.3

```
_QWORD *v5; // r4
int v7; // [sp+0h] [bp-128h]
__int64 v8; // [sp+120h] [bp-8h]

  v8 = a2;
  v4 = a2;
  v5 = a1;
  memcpy(&v7, &v8, 0x100u);
  memcpy(v5, &v7, 0x100u);
```

PSEUDOCODE V7.4

```
_BYTE v7[256]; // [sp+0h] [bp-128h]
  __int64 v8; // [sp+120h] [bp-8h]

  v8 = a2;
  v4 = a2;
  memcpy(v7, &v8, sizeof(v7));
  memcpy(a1, v7, 0x100u);
```

Support for more floating-point helpers

On the left there is a mysterious call to **_extendsfdf2**. In fact this is a compiler helper function that just converts a single precision floating point value into a double precision value. However, we do not want to see this call as is. It is much better to translate it into the code that looks more like C. Besides, there is a special treatment for printf-like functions.

PSEUDOCODE V7.3

```
void __cdecl printf_float(float a)
{
  double v1; // r0
```

```
  v1 = COERCE_DOUBLE(_extendsfdf2(LODWORD(a)));
  printf("%f\n", v1);
}
```

PSEUDOCODE V7.4

```
void __cdecl printf_float(float a)
{
  printf("%f\n", a);
}
```

Automatic variable mapping

In some cases we can easily prove that one variable can be mapped into another. The new version automatically creates a variable mapping in such cases. This makes the output shorter and easier to read. Needless to say that the user can revert the mapping if necessary.

PSEUDOCODE V7.3

```
__int64 sprintf_s(
        char *__ptr64 const _Buffer,
        const unsigned __int64 _BufferCount,
        const char *__ptr64 const _Format,
        ...)
{
  char *v3; // x21
  unsigned __int64 v4; // x20
  const char *v5; // x19
  unsigned __int64 *v6; // x0
  __int64 result; // x0
  va_list va; // [xsp+38h] [xbp+38h]

  va_start(va, _Format);
  v3 = _Buffer;
  v4 = _BufferCount;
  v5 = _Format;
  v6 = _local_stdio_printf_options();
  return _stdio_common_vsprintf_s(*v6, v3, v4, v5, 0i64, (char
*__ptr64)va);
}
```

PSEUDOCODE V7.4

```
__int64 sprintf_s(
        char *__ptr64 const _Buffer,
```

```
        const unsigned __int64 _BufferCount,
        const char *__ptr64 const _Format,
        ...)
{
  unsigned __int64 *v6; // x0
  __int64 result; // x0
  va_list va; // [xsp+38h] [xbp+38h]

  va_start(va, _Format);
  v6 = _local_stdio_printf_options();
  return _stdio_common_vsprintf_s(*v6, _Buffer, _BufferCount,
_Format, 0i64,
                                  (char *__ptr64)va);
}
```

Automatic symbolic names

The new version automatically applies symbolic constants when necessary. Less manual work.

PSEUDOCODE V7.3

```
if ( operation == 4 )
    return BaseDllReadVariableNames(v1, v2);
  if ( operation != 6 )
  {
    if ( operation == 2 || operation == 3 )
      return BaseDllWriteVariableValue(v1, v2, 0, 0);
    if ( operation == 7 || operation == 8 )
      return BaseDllWriteApplicationVariables(v1, v2);
```

PSEUDOCODE V7.4

```
  if ( operation == ReadKeyNames )
    return BaseDllReadVariableNames(v1, v2);
  if ( operation != ReadSection )
  {
    if ( operation == WriteKeyValue || operation == DeleteKey )
      return BaseDllWriteVariableValue(v1, v2, 0, 0);
    if ( operation == WriteSection || operation == DeleteSection )
      return BaseDllWriteApplicationVariables(v1, v2);
```

Simplified C++ names

This is not the longest C++ function name one may encounter but just compare the left and right sides. In fact the right side could even fit into one line easily, we just kept it multiline to be consistent. By the way, all names in IDA benefit from this simplification, not only the ones displayed by the decompiler. And it is configurable!

PSEUDOCODE V7.3

```
std::basic_string<char,std::char_traits<char>,std::allocator<char> >
*
__fastcall
std::_System_error::_Makestr(
  std::basic_string<char,std::char_traits<char>,std::allocator<char>
> *result,
  std::error_code _Errcode,
  std::basic_string<char,std::char_traits<char>,std::allocator<char>
> _Message)
```

PSEUDOCODE V7.4

```
std::string *
__fastcall
std::_System_error::_Makestr(
  std::string *result,
  std::error_code _Errcode,
  std::string _Message)
```

Improved handling of 64-bit arithmetics

The battle is long but we do not give up. More 64-bit patterns are recognized now.

PSEUDOCODE V7.3

```
v0 = h();
return (__int16)((((v0 ^ (SHIDWORD(v0) >> 31)) - (SHIDWORD(v0) >>
31)) & 0x3FF ^ (SHIDWORD(v0) >> 31))
                  - (SHIDWORD(v0) >> 31));
```

PSEUDOCODE V7.4

```
  return h() % 1024;
```

Better detection of 64-bit decrements

Yet another example of 64-bit arithmetics. The code on the left is correct but not useful at all. It can and should be converted into the simple equivalent text on the right.

PSEUDOCODE V7.3

```
v1 = a1 + 0xFFFFFFFFLL;
  HIDWORD(v1) = ((unsigned __int64)(a1 + 0xFFFFFFFFLL) >> 32) - 1;
```

PSEUDOCODE V7.4

```
return a1 - 1;
```

More meaningful variable names

Currently we support only GetProcAddress but we are sure that we will expand this feature in the future.

PSEUDOCODE V7.3

```
    dword_12313BA8 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD,
_DWORD))
                       GetProcAddress(v4, "MessageBoxA");
    if ( !dword_12313BA8 )
      return 0;
    dword_12313BAC = GetProcAddress(v5, "GetActiveWindow");
    dword_12313BB0 = (int (__stdcall *)(_DWORD))GetProcAddress(v5,
"GetLastActivePopup");
  }
  if ( dword_12313BAC )
  {
    v3 = dword_12313BAC();
    if ( v3 )
    {
      if ( dword_12313BB0 )
        v3 = dword_12313BB0(v3);
    }
  }
  return dword_12313BA8(v3, a1, a2, a3);
```

PSEUDOCODE V7.4

```
    MessageBoxA_0 = (int (__stdcall *)(HWND, LPCSTR, LPCSTR, UINT))
                    GetProcAddress(v4, "MessageBoxA");
    if ( !MessageBoxA_0 )
      return 0;
    GetActiveWindow = (HWND (__stdcall *)())GetProcAddress(v5,
"GetActiveWindow");
```

```
    GetLastActivePopup = (HWND (__stdcall *)(HWND))GetProcAddress(v5,
"GetLastActivePopup");
  }
  if ( GetActiveWindow )
  {
    v3 = GetActiveWindow();
    if ( v3 )
    {
      if ( GetLastActivePopup )
        v3 = GetLastActivePopup(v3);
    }
  }
  return MessageBoxA_0(v3, a1, a2, a3);
```