Analysis of Microsoft VML Integer Wrap / Heap Overflow Vulnerability, CVE-2007-0024

What we know:

1) Vulnerable file: vgx.dll
2) Vulnerable function: exact name unknown, but contains “recolorinfo”
3) Details:
    a. integer overflow per unchecked multiplication result
    b. heap overflow due to insufficient allocation

Procedure:

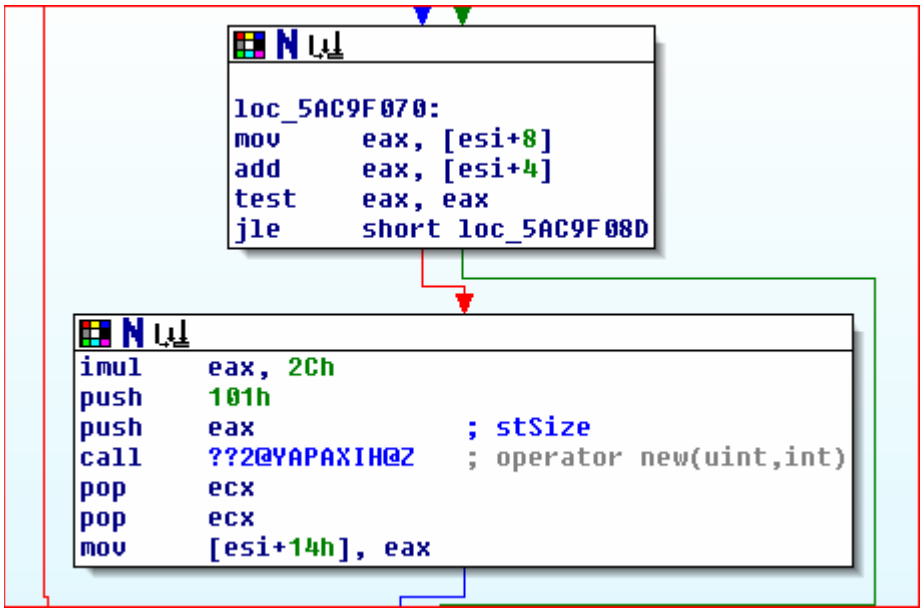
1) Open vgx.dll in IDA, search for “recolorinfo” in Names pane
2) From those results, search for text “imul”
3) From those results, search for calls to malloc or new

Results:

Surprisingly, there is only one function that fits the criteria in vgx.dll:

CVMLRecolorinfo::InternalLoad()

This function is 0x5AC9F00D in vgx.dll version 6.0.2800.1106 and here is the code:



There apparently is a structure pointed to by esi, which has visible members at offsets 8h, 4h, and 14h. The signed integers at 8h and 4h are added together. If the result is less than or equal to zero, the function returns. Otherwise, the result is multiplied by 2Ch and passed to malloc() as the number of bytes to allocate.

Here is some code:

```
struct st_t
{
	int foo, bar;
	char *ptr; //maybe not even char, but ptr to something
};

bool f()
{
	st_t st;
	//members initialized by code not shown
	int size = st.foo + st.bar;

	if (size <= 0)
		return(true);
	else
	{
		size *= 44;
		st.ptr = (char *)malloc(size);
	}
	return(true);
}
```

So, foo and bar are returned from GetIntValue(), which we can assume is supplied parameters from user input. Therefore, a user can control the value of foo and bar, which are both integers. Four distinct conditions can occur:

foo+bar is less than zero, e.g. 80000000h. In this case, the function never calls malloc(), because it doesn't pass the first size test.

foo+bar equals zero, e.g. 00000000h. In this case, the function never calls malloc(), because it doesn't pass the first size test.

foo+bar is greater than zero. This case is special, and there are two possible outcomes.

The safe outcome is if foo+bar is greater than zero, but small enough to not wrap around when multiplied by 2Ch. For example, something like 1000h is safe. This is multiplied by 2Ch, which results in 2C000h bytes being allocated.

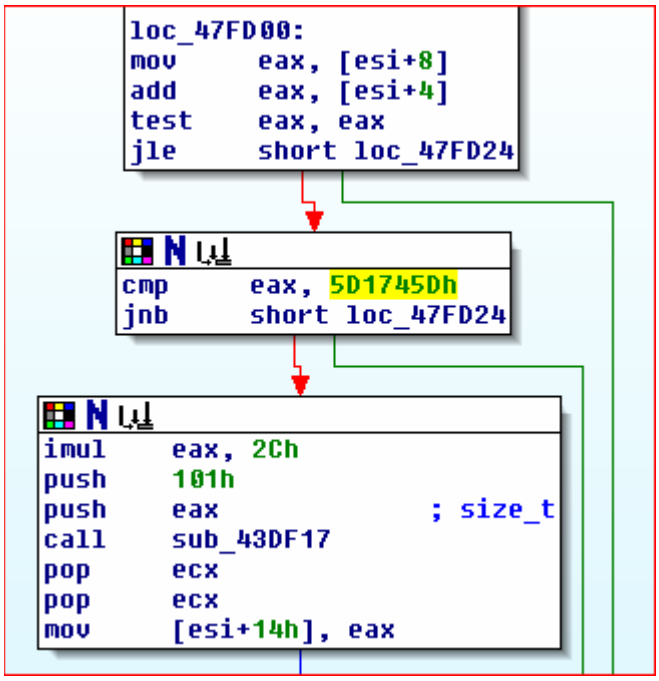
The unsafe outcome is if foo+bar is greater than zero, but wraps when multiplied by 2Ch. For example, 0x5D1745Eh is not safe. It is not less than or equal to zero, so it will pass the first size check. But, it is multiplied by 2Ch, which results in 28h bytes being allocated (5D1745Eh * 2Ch = 28h).

It is just my *assumption* that 2Ch is the size of a structure or class, and 5D1745Eh is the number of structures or classes that the program needs. After this allocation, the program thinks it has room for 5D1745Eh items, but really it can't even hold one.

To fix this, the patched vgx.dll added this code after the first size check:

```
else if (size >= 0x5D1745D) // ((0x80000000 / 0x2C) * 2)
      return(true);
```

And here is the proof:



Note that foo+bar can end up like 5D1745Ah which will pass both size checks, but still result in an error. However, in this case the error is likely to just be ERROR_NOT_ENOUGH_MEMORY because malloc() will be trying to allocate 4+ GB of memory. Interestingly, the return value of malloc() is not checked before it is copied into the st.ptr member; and the function returns true either way. Most likely, the caller checks if st.ptr is NULL before using it and even if not, there will only be an on-write access violation and not a heap overflow.