SC22/WG14/N1332

August 6, 2008

Reply to: Thomas Plum, tplum@plumhall.com
Arjun Bijanki, Arjun.Bijanki@microsoft.com

Encoding and Decoding Function Pointers

In many systems and applications, pointers to functions are stored un-encrypted in locations addressable by exploit code. A hacker exploiting a vulnerability in a program could potentially overwrite the function pointer and thereby hijack the process when the function is called. Note that this attack can even occur on systems where the stack is not executable.

Instead of storing a function pointer, the program can store an encrypted version of the pointer. An attacker would need to break the encryption in order to redirect the pointer to other code. This is similar to what's recommended when dealing with other sensitive data (e.g. passwords).

We propose to add to the C1x standard library two functions whose names might be encode_pointer and decode_pointer. These functions are similar in purpose, but slightly different in details, from two functions in Microsoft Windows (EncodePointer and DecodePointer), which are used by Visual C++'s C runtime libraries.

Note that this process of pointer-encoding does not prevent buffer overruns or arbitrary-memory-write attacks, but it does make such attacks more difficult to exploit.

First draft of proposed wording:

7.20.9 Pointer encoding and decoding functions

7.20.9.1 The encode_pointer function

Synopsis

```
#include <stdlib.h>
void (*)() encode_pointer(void(*pf)());
```

Description

The `encode_pointer` function shall perform a transformation on the `pf` argument, such that the `decode_pointer` function shall reverse that transformation. Thus, for any pointer to function `pfn`, `decode_pointer(encode_pointer((void(*)())pfn)`, when converted to the type of `pfn`, shall equal `pfn`.

However, this inverse relationship between `encode_pointer` and `decode_pointer` shall not be valid if the invocations of `encode_pointer` and `decode_pointer` take place under certain implementation-defined conditions.

[Footnote: For example, if the invocations take place in different execution processes, then the inverse relationship is not valid. In that implementation, the transformation method could encode the process number in the encode/decode algorithm.]

Returns

The result of the transformation.

7.20.9.2 The decode_pointer function

Synopsis

```
#include <stdlib.h>
void (*)() decode_pointer(void(*epf)());
```

Description

The `decode_pointer` function shall perform a transformation on the `epf` argument, such that it shall reverse the transformation performed by the `encode_pointer` function. Thus, for any pointer to function `pfn`, `decode_pointer(encode_pointer((void(*)())pfn)`, when converted to the type of `pfn`, shall equal `pfn`.

However, this inverse relationship between `encode_pointer` and `decode_pointer` shall not be valid if the invocations of `encode_pointer` and `decode_pointer` take place under certain implementation-defined conditions.

[Footnote: For example, if the invocations take place in different execution processes, then the inverse relationship is not valid. In that implementation, the transformation method could encode the process number in the encode/decode algorithm.]

Returns

The result of the inverse transformation.