ISO/IEC JTC 1/SC 22/WG 23 N 0278

Revision of C annex portion of N0270

Date	10 September 2010
Contributed by	John Benito
Original file name	C - HCB.pdf
Notes	This is a revision of the C annex portion of N0270

C.HCB Buffer Boundary Violation [HCB]

C.HCB.1 Terminology and features

C.HCB.2 Description of vulnerability

A buffer boundary violation condition occurs when an array is indexed outside its bounds, or pointer arithmetic results in an access to storage that occurs outside the bounds of the object accessed.

In C, the subscript operator `[]` is defined such that `E1[E2]` is identical to `(*((E1)+(E2)))`, so that in either representation, the value in location `(E1+E2)` is returned. C does not perform bounds checking on arrays, so the following code:

```
int foo(const int i) {
      int x[] = {0,0,0,0,0,0,0,0,0,0};
      return x[i];
}
```

will return whatever is in location `x[i]` even if, `i` were equal to -10 or 10 (assuming either subscript was still within the address space of the program). This could be sensitive information or even a return address, which if altered by changing the value of `x[-10]`or `x[10]`, could change the program flow.

The following code is more appropriate and would not violate the boundaries of the array `x`:

```
int foo( const int i) {
      int x[X_SIZE] = {0};
      if (i < 0 || i >= X_SIZE) {
        return ERROR_CODE;
      }

      else {
            return x[i];
      }
}
```

A buffer boundary violation may also occur when copying, initializing, writing or reading a buffer if attention to the index or addresses used are not taken. For example, in the following move operation there is a buffer boundary violation:

```
char buffer_src[]={“abcdefg”};
char buffer_dest[5]={0};

strcpy(buffer_dest, buffer_src);
```

the `buffer_src` is longer than the `buffer_dest`, and the code does not check for this before the actual copy operation is invoked. A safer way to accomplish this copy would be:

```
char buffer_src[]={“abcdefg”];
char buffer_dest[5]={0};

strncpy(buffer_dest, buffer_src, sizeof(buffer_dest) -1);
```

this would not cause a buffer bounds violation, however, because the destination buffer is smaller than the source buffer, the destination buffer will now hold “abcd”, the 5th element of the array would hold the null character.

C.HCB.3 Avoiding the vulnerability or mitigating its effects

- Validate all input values.
- Check any array index before use if there is a possibility the value could be outside the bounds of the array.
- Use length restrictive functions such as `strncpy()`instead of `strcpy()`.
- Use stack guarding add-ons to detect overflows of stack buffers.
- Do not use the deprecated functions or other language features such as `gets()`.
- Be aware that the use of all of these measures may still not be able to stop all buffer overflows from happening. However, the use of them can make it much rarer for a buffer overflow to occur and much harder to exploit it.
- Use alternative functions as specified in ISO/IEC TR 24731-1:2007 or TR 24731-2:2010. These Technical Reports provides alternative functions for the C Library (as defined in ISO/IEC 9899:1999) that promotes safer, more secure programming. The functions verify that output buffers are large enough for the intended result and return a failure indicator if they are not. Optionally, failing functions call a“"runtime-constraint handler"” to report the error. Data is never written past the end of an array. All string results are null terminated. In addition, the functions in ISO/IEC TR 24731-1:2007 are re-entrant: they never return pointers to static objects owned by the function. ISO/IEC TR 24731-1:2007 also contains functions that address insecurities with the C input-output facilities.

C.HCB.4 Implications for standardization

Future standardization efforts should consider:

- Defining an array type that does automatic bounds checking.
- Deprecating less safe functions such as `strcpy()` and `strcat()` where a more secure alternative is available.
- Defining safer and more secure replacement functions such as `memncpy()` and `memncmp()` to complement the `memcpy()` and `memcmp()` functions (see in Implications for standardization.XYW).
- Adopting one of the Technical Reports on safer C library functions, Extensions to the C Library (TR 24731-1: Part I: *Bounds-checking interfaces* or TR 24731-2: Part II: *Dynamic allocation functions*, that have been developed by WG 14.[1]

---

[1] TR 24731-1 has been added to the WG 14 working paper as an optionally normative Annex.