ISO/IEC JTC 1/SC 22/WG 23 N 0309

Proposed vulnerability description on Inter-language calling

Date	11 March 2011
Contributed by	John Benito
Original file name	djs.docx
Notes	

6.X Inter-language Calling [DJS]

6.x.1 Description of application vulnerability

When an application is developed using more than one programming language, complications arise. The calling conventions, data layout, error handing and return conventions all differ between languages, if these are not addressed correctly, stack overflow/underflow, data corruption, and memory corruption are possible.

In multi-language development environment it is also difficult to reuse code across the languages.

6.x.2 Cross reference

[None**]**

6.x.3 Mechanism of failure

When calling a function that has been developed using a language different from the calling language, the call convention and the return convention used must be taken into account. If these conventions are not handled correctly, there is a good chance the calling stack will be corrupted, see [OTR]. The call convention covers how the language invokes the call, see [CJS], but how the parameters are handled.

Many software languages have restriction on length of identifiers, the type of characters that can be used as the first character, and the case of the characters used. All of these need to be taken into account when invoking a routine written in a language other than the calling language.

Character and aggregate data types require special treatment in a multi-language development environment, the data layout of all languages that are to be used must be taken into consideration, this includes padding and alignment. If these data types are not handled correctly, the data could be corrupted, the memory could be corrupted, or both may become corrupt. This can happen by writing/reading past either end of the data structure, see [HCB]. For example, a Pascal's `STRING` data type

```
VAR str: STRING(10);
```

corresponds to a C structure

```
struct {
  int length;
  char str [10];
};
```

where `length` contains the actual length of `STRING`.

Most numeric data types have counterparts across languages, but again the layout should be understood, and only those types that match the languages should be used. For example, in some implementations of C++ a

```
signed char
```

would match a Fortran

```
INTEGER*1
```

and would match a Pascal

```
PACKED -128..127
```

these can be implementation-defined and should be verified.

6.x.4 Applicable language characteristics

The vulnerability is applicable to languages with the following characteristics:

- All high level programming languages and low level programming languages are susceptible to this vulnerability when used in a multi-language development environment.

6.x.5 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Understand the calling convenes of all languages used.
- Understand the data layout of all data types used.
- Understand the return conventions of all languages used.
- Ensure that the language in which error check occurs is the one that handles the error.
- Avoid using uppercase letters in identifiers.
- Avoid using the underscore (_) and dollar sign ($) as the first character in identifiers.
- Avoid using long identifier names.

6.x.6 Implications for standardization

In future standardization activities, the following items should be considered:

- Standards committees should consider developing guides for inter-language calling with languages most often used with their programming language.