ISO/IEC JTC 1/SC 22/OWGV N 0229

Revised proposal for a vulnerability description on namespace issues

Date	19 October 2009
Contributed by	Erhard Ploedereder (Ada-Europe)
Original file name	beaujolais-V2.doc
Notes	Replace N0197

6.C Namespace Issues [BJL]

6.C.1 Description of Application Vulnerability

If a language provides separate, non-hierarchical namespaces and a means to make names declared in these name spaces directly visible to an application, the potential of unintentional and possible disastrous change in application behavior can arise, when names are added to a namespace during maintenance.

Namespaces include constructs like packages, modules, libraries, classes or any other means of grouping declarations for import into other program units.

6.C.2 Cross References

6.C.3 Mechanism of Failure

The failure is best illustrated by an example. Namespace N1 provides the name A but not B; Namespace N2 provides the name B but not A. The application wishes to use A from N1 and B from N2. At this point, there are no obvious issues. The application chooses (or needs to) import the namespaces to obtain names for direct usage, e.g.,

```
Use N1, N2; -- presumed to make all names in N1 and N2 directly visible
… X := A + B;
```

The semantics of the above example are intuitive and unambiguous.

Later, during maintenance, the name B is added to N1. The change to the namespace usually implies a recompilation of dependent units. At this point, two declarations of B are applicable for the use of B in the above example.

Some languages try to disambiguate the above situation by stating preference rules in case of such ambiguity among names provided by different name spaces. If, in the above example, N1 is preferred over N2, the meaning of the use of B changes silently, presuming that no typing error arises. Consequently the semantics of the program change silently and assuredly unintentionally, since the implementor of N1 can not assume that all users of N1 would prefer to take any declaration of B from N1 rather than its previous namespace.

It does not matter what the preference rules actually is, as long as the namespaces are mutable. The above example is easily extended by adding A to N2 to show a symmetric error situation for a different precedence rule.

If a language supports overloading of subprograms, the notion of “same name” used in the above example is extended to mean not only the same name, but also the same signature of the subprogram. For vulnerabilities associated with overloading and overriding, see section @ref(OAO). In the context of namespaces, however, adding signature matching to the name binding process, merely extends the described problem from simple names to full signatures, but does not alter the mechanism or quality of the described vulnerability. In particular, overloading does not introduce more ambiguity for binding to declarations in different name spaces.

This vulnerability not only creates unintentional errors. It also can be exploited maliciously, if the source of the application and of the namespaces is known to the aggressor and one of the namespaces is mutable by him or her.

6.C.4 Applicable Language Characteristics

The vulnerability is applicable to languages with the following characteristics:

- Languages that support non-hierarchical separate name-spaces, have means to import all names of a namespace “wholesale” for direct use, and have preference rules to chose among multiple imported direct homographs. All three conditions needs to be satisfied for the vulnerability to arise.

6.C.5 Avoiding the Vulnerability or Mitigating its Effects

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

- Avoiding “wholesale” import directives
- Using only selective “single name” import directives or using fully qualified names (in both cases, provided that the language offers the respective capabilities)

6.C.6 Implications for Standardization

- Languages should not have preference rules among mutable namespaces. Ambiguities should be illegal and avoidable by the user, e.g., by using names qualified by their originating namespace.