6.x Library Signature [NSQ]

6.x.0 Status and history

2009-02-14 – Edits from editor's teleconference 28th Jan
2008-10-05: Revised draft provided by Dan Nagle
2008-09: Considered at Stuttgart meeting

6.x.1 Description of application vulnerability

~~Some older libraries were coded before the value of subprogram signatures was recognized, and added to language standards.~~ Programs written in modern languages may use libraries written in other languages than the program implementation language. If the library is large, the effort of adding ~~those~~ signatures for all of the functions use by hand may be tedious and error-prone. Portable cross-language signatures ~~may~~ will require detailed understanding of both languages, which ~~one~~ a programmer may lack.

Integrating two or more programming languages into a single executable relies upon knowing how to interface the function calls, argument list and global data structures so the symbols match in the object code during linking.

Byte alignment can be a source of data corruption if memory boundaries between the programming languages are different. Each language may also align structure data differently.

6.x.2 Cross reference

6.x.3 Mechanism of failure

~~When an older software library lacks the language-specified signatures, due to its being prepared prior to the requirement that signatures be used, the signature must be created. If this is done manually, it may be tedious and error-prone. Furthermore, if~~When the library and the application in which it is to be used are written in different languages, the specification of signatures is complicated by inter-language issues ~~as well~~.

As used in this vulnerability description, the term library includes the interface to the operating system, which may be specified only for the language used to code the operating system itself. In this case, any program written in any other language faces the inter-language interoperability issue of creating a fully-functional signature.

~~Automated methods may exist. Or, translators may have options to create the signatures as they compile the older library. However, neither of these remedies might be required by the language standard and so may not be universally available.~~

~~If~~ When the application language and the library language are different, then the ability to specify signatures according to either standard may not exist, or be very difficult. Thus, a translator-by-translator solution may be needed,
which maximizes the probability of incorrect signatures (since the solution must be recreated for each translator pair). Incorrect signatures may or may not be caught during the linking phase.

6.x.4 Applicable language characteristics

~~Languages where older versions of the language standard did not specify that subprogram signatures be supplied for all subprogram references.~~

Languages that do not specify how to describe signatures for subprograms written in other languages.

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:

- ~~•Use translator options to create the needed signatures when compiling the library~~
- Use tools to create the signatures
- Avoid using translator options or language features to reference library subprograms without proper signatures
- ~~•Try to find a later version of the library that has the signatures~~

6.x.6 Implications for standardization

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

- ~~•Require translators to create signatures when needed when translating older libraries that lack them.~~
- Provide correct linkage even in the absence of correctly specified procedure signatures. (Note that this may be very difficult where the original source code is unavailable.)
- Provide specified means to describe the signatures of subprograms ~~written using other languages~~.

6.x.7 Bibliography