API04-C. Provide a consistent and usable error-checking mechanism

Functions should provide consistent and usable error-checking mechanisms. Complex interfaces are sometimes ignored by programmers, resulting in code that is not error checked. Inconsistent interfaces are frequently misused and difficult to use, resulting in lower-quality code and higher development costs.

Noncompliant Code Example (`strlcpy()`)

The `strlcpy()` function copies a null-terminated source string to a destination array. It is designed to be a safer, more consistent, and less error-prone replacement for `strcpy()`.

The `strlcpy()` function returns the total length of the string it tried to create (the length of the source string).

To detect truncation, perhaps while building a path name, code such as the following might be used:

```
char *dir, *file, pname[MAXPATHLEN];

/* ... */

if (strlcpy(pname, dir, sizeof(pname)) >= sizeof(pname)) {
  /* Handle source-string-too long error */
}
```

Compliant Solution (`strcpy_m()`)

The managed string library [Burch 2006] handles errors by consistently returning the status code in the function return value. This approach encourages status checking because the user can insert the function call as the expression in an `if` statement and take appropriate action on failure.

The `strcpy_m()` function is an example of a managed string function that copies a source-managed string into a destination-managed string:

```
errno_t retValue;
string_m dest, source;

/* ... */

if (retValue = strcpy_m(dest, source)) {
  fprintf(stderr, "Error %d from strcpy_m.\n", retValue);
}
```

The greatest disadvantage of this approach is that it prevents functions from returning any other value. Consequently, all values (other than the status) returned by a function must be returned as a pass-by-reference parameter, preventing a programmer from nesting function calls. This trade-off is necessary because nesting function calls can conflict with a programmer's willingness to check status codes.

Risk Assessment

Failure to provide a consistent and usable error-checking mechanism can result in type errors in the program.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
API04-C	Medium	Unlikely	Medium	**P4**	**L3**

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

Key here (explains table format and definitions)

Taxonomy	Taxonomy item	Relationship
ISO/IEC 9945:2003		Prior to 2018-01-12: CERT: Unspecified Relationship

ISO/IEC 23360-1:2006		Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TR 24731-1		Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TR 24731-2		Prior to 2018-01-12: CERT: Unspecified Relationship
MISRA C:2012	Rule 21.3 (required)	Prior to 2018-01-12: CERT: Unspecified Relationship
MISRA C:2012	Directive 4.12 (required)	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-754, Improper check for unusual or exceptional conditions	Prior to 2018-01-12: CERT:

Bibliography

[Burch 2006]	
[CERT 2006c]	
[Miller 1999]	
[Seacord 2013]	Chapter 2, "Strings"