POS34-C. Do not call putenv() with a pointer to an automatic variable as the argument

The POSIX function `putenv()` is used to set environment variable values. The `putenv()` function does not create a copy of the string supplied to it as an argument; rather, it inserts a pointer to the string into the environment array. If a pointer to a buffer of automatic storage duration is supplied as an argument to `putenv()`, the memory allocated for that buffer may be overwritten when the containing function returns and stack memory is recycled. This behavior is noted in the Open Group Base Specifications, Issue 6 [Open Group 2004]:

> *A potential error is to call `putenv()` with an automatic variable as the argument, then return from the calling function while `string` is still part of the environment.*

The actual problem occurs when passing a *pointer* to an automatic variable to `putenv()`. An automatic pointer to a static buffer would work as intended.

Noncompliant Code Example

In this noncompliant code example, a pointer to a buffer of automatic storage duration is used as an argument to `putenv()` [Dowd 2006]. The `TEST` environment variable may take on an unintended value if it is accessed after `func()` has returned and the stack frame containing `env` has been recycled.

Note that this example also violates DCL30-C. Declare objects with appropriate storage durations.

```
int func(const char *var) {
  char env[1024];
  int retval = snprintf(env, sizeof(env),"TEST=%s", var);
  if (retval < 0 || (size_t)retval >= sizeof(env)) {
    /* Handle error */
  }

  return putenv(env);
}
```

Compliant Solution (`static`)

This compliant solution uses a static array for the argument to `putenv()`.

```
int func(const char *var) {
  static char env[1024];

  int retval = snprintf(env, sizeof(env),"TEST=%s", var);
  if (retval < 0 || (size_t)retval >= sizeof(env)) {
    /* Handle error */
  }

  return putenv(env);
}
```

Compliant Solution (Heap Memory)

This compliant solution dynamically allocates memory for the argument to `putenv()`:

```
int func(const char *var) {
  static char *oldenv;
  const char *env_format = "TEST=%s";
  const size_t len = strlen(var) + strlen(env_format);
  char *env = (char *) malloc(len);
  if (env == NULL) {
    return -1;
  }
  int retval = snprintf(env, len, env_format, var);
  if (retval < 0 || (size_t)retval >= len) {
    /* Handle error */
  }
  if (putenv(env) != 0) {
    free(env);
    return -1;
  }
  if (oldenv != NULL) {
    free(oldenv); /* avoid memory leak */
  }
  oldenv = env;
  return 0;
}
```

The POSIX `setenv()` function is preferred over this function [Open Group 2004].

Compliant Solution (`setenv()`)

The `setenv()` function allocates heap memory for environment variables, which eliminates the possibility of accessing volatile stack memory:

```
int func(const char *var) {
  return setenv("TEST", var, 1);
}
```

Using `setenv()` is easier and consequently less error prone than using `putenv()`.

Risk Assessment

Providing a pointer to a buffer of automatic storage duration as an argument to `putenv()` may cause that buffer to take on an unintended value. Depending on how and when the buffer is used, it can cause unexpected program behavior or possibly allow an attacker to run arbitrary code.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
POS34-C	high	unlikely	medium	P6	L2

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC-POS34**	
CodeSonar	5.1p0	**(customization)**	Users can add a custom check for all uses of `putenv()`.
Compass/ROSE			
Parasoft C/C++test	10.4.2	**CERT_C-POS34-a** **CERT_C-POS34-b**	Usage of system properties (environment variables) should be restricted Do not call putenv() with a pointer to an automatic variable as the argument
Polyspace Bug Finder	R2019b	CERT C: Rule POS34-C	Checks for use of automatic variable as putenv-family function argument (rule fully covered)
PRQA QA-C	9.5	**5024**	Partially implemented

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

CERT-CWE Mapping Notes

Key here for mapping notes

CWE-252/CWE-253/CWE-391 and ERR33-C/POS34-C

Independent(ERR33-C, POS54-C, FLP32-C, ERR34-C)

Intersection(CWE-252, CWE-253) = Ø

CWE-391 = Union(CWE-252, CWE-253)

CWE-391 = Union(ERR33-C, POS34-C, list) where list =

- Ignoring return values of functions outside the C or POSIX standard libraries

Bibliography

[Dowd 2006]	Chapter 10, "UNIX Processes"
[ISO/IEC 9899:2011]	Section 6.2.4, "Storage Durations of Objects" Section 7.22.3, "Memory Management Functions"
[Open Group 2004]	`putenv()` `setenv()`