STR06-C. Do not assume that strtok() leaves the parse string unchanged

The C function `strtok()` is a string tokenization function that takes two arguments: an initial string to be parsed and a `const`-qualified character delimiter. It returns a pointer to the first character of a token or to a null pointer if there is no token.

The first time `strtok()` is called, the string is parsed into tokens and a character delimiter. The `strtok()` function parses the string up to the first instance of the delimiter character, replaces the character in place with a null byte (`'\0'`), and returns the address of the first character in the token. Subsequent calls to `strtok()` begin parsing immediately after the most recently placed null character.

Because `strtok()` modifies the initial string to be parsed, the string is subsequently unsafe and cannot be used in its original form. If you need to preserve the original string, copy it into a buffer and pass the address of the buffer to `strtok()` instead of the original string.

Noncompliant Code Example

In this example, the `strtok()` function is used to parse the first argument into colon-delimited tokens; it outputs each word from the string on a new line. Assume that `PATH` is `"/usr/bin:/usr/sbin:/sbin"`.

```
char *token;
char *path = getenv("PATH");

token = strtok(path, ":");
puts(token);

while (token = strtok(0, ":")) {
  puts(token);
}

printf("PATH: %s\n", path);
/* PATH is now just "/usr/bin" */
```

After the loop ends, `path` is modified as follows: `"/usr/bin\0/bin\0/usr/sbin\0/sbin\0"`. This is an issue because the local `path` variable becomes `/usr/bin` and because the environment variable `PATH` has been unintentionally changed, which can have unintended consequences. (See ENV 30-C. Do not modify the object referenced by the return value of certain functions.)

Compliant Solution

In this compliant solution, the string being tokenized is copied into a temporary buffer that is not referenced after the call to `strtok()`:

```
char *token;
const char *path = getenv("PATH");
/* PATH is something like "/usr/bin:/bin:/usr/sbin:/sbin" */

char *copy = (char *)malloc(strlen(path) + 1);
if (copy == NULL) {
  /* Handle error */
}
strcpy(copy, path);
token = strtok(copy, ":");
puts(token);

while (token = strtok(0, ":")) {
  puts(token);
}

free(copy);
copy = NULL;

printf("PATH: %s\n", path);
/* PATH is still "/usr/bin:/bin:/usr/sbin:/sbin" */
```

Another possibility is to provide your own implementation of `strtok()` that does not modify the initial arguments.

Risk Assessment

The *Linux Programmer's Manual* (man) page on `strtok(3)` [Linux 2008] states:

> *Never use this function. This function modifies its first argument. The identity of the delimiting character is lost. This function cannot be used on constant strings.*

The improper use of `strtok()` is likely to result in truncated data, producing unexpected results later in program execution.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
STR06-C	Medium	Likely	Medium	P12	L1

Automated Detection

Tool	Version	Checker	Description
CodeSonar	5.2p0	**(customization)**	Users who wish to avoid using `strtok()` entirely can add a custom check for all uses of `strtok()`.
Compass/ROSE			
LDRA tool suite	9.7.1	**602 S**	Enhanced Enforcement
PRQA QA-C	9.7	**5007**	

Related Vulnerabilities

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

Related Guidelines

SEI CERT C++ Coding Standard	VOID STR06-CPP. Do not assume that strtok() leaves the parse string unchanged
MITRE CWE	CWE-464, Addition of data structure sentinel

Bibliography

[Linux 2008]	strtok(3)