API01-C. Avoid laying out strings in memory directly before sensitive data

Strings (both character and wide-character) are often subject to buffer overflows, which will overwrite the memory immediately past the string. Many rules warn against buffer overflows, including STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator. Sometimes the danger of buffer overflows can be minimized by ensuring that arranging memory such that data that might be corrupted by a buffer overflow is not sensitive.

Noncompliant Code Example

This noncompliant code example stores a set of strings using a linked list:

```
const size_t String_Size = 20;
struct node_s {
  char name[String_Size];
  struct node_s* next;
}
```

A buffer overflow on `name` would overwrite the `next` pointer, which could then be used to read or write to arbitrary memory.

Compliant Solution

This compliant solution creates a linked list of strings but stores the `next` pointer before the string:

```
const size_t String_Size = 20;
struct node_s {
  struct node_s* next;
  char name[String_Size];
}
```

If buffer overflow occurs on `name`, the `next` pointer remains uncorrupted.

Compliant Solution

In this compliant solution, the linked list stores pointers to strings that are stored elsewhere. Storing the strings elsewhere protects the `next` pointer from buffer overflows on the strings.

```
const size_t String_Size = 20;
struct node_s {
  struct node_s* next;
  char* name;
}
```

Exceptions

API01-C-EX1: Using a string before sensitive data such as pointers is permitted when it is not practical to segregate the strings from the sensitive data.

Each of the following code examples creates a linked list of strings, but each node is actually stored inside an array. This practice ensures that the string is always in front of a `next` pointer regardless of how they are ordered in the struct.

```
const size_t String_Size = 20;
struct node_s {
  char name[String_Size];
  struct node_s* next;
}
struct node_s list[10];
```

```
const size_t String_Size = 20;
struct node_s {
  struct node_s* next;
  char name[String_Size];
}
struct node_s list[10];
```

Risk Assessment

Failure to follow this recommendation can result in memory corruption from buffer overflows, which can easily corrupt data or yield remote code execution.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
API01-C	High	Likely	High	P9	L2

Automated Detection

Tool	Version	Checker	Description
Parasoft C/C++test	10.4.2	**CERT_C-API01-a** **CERT_C-API01-b**	Avoid overflow when writing to a buffer Avoid using unsafe string functions which may cause buffer overflows