MEM05-C. Avoid large stack allocations

Avoid excessive stack allocations, particularly in situations where the growth of the stack can be controlled or influenced by an attacker. See INT04-C. Enforce limits on integer values originating from tainted sources for more information on preventing attacker-controlled integers from exhausting memory.

Noncompliant Code Example

The C Standard includes support for variable length arrays (VLAs). If the array length is derived from an untrusted data source, an attacker can cause the process to perform an excessive allocation on the stack.

This noncompliant code example temporarily stores data read from a source file into a buffer. The buffer is allocated on the stack as a VLA of size `bufsize`. If `bufsize` can be controlled by a malicious user, this code can be exploited to cause a denial-of-service attack:

```
int copy_file(FILE *src, FILE *dst, size_t bufsize) {
  char buf[bufsize];

  while (fgets(buf, bufsize, src)) {
    if (fputs(buf, dst) == EOF) {
      /* Handle error */
    }
  }

  return 0;
}
```

The BSD extension function `alloca()` behaves in a similar fashion to VLAs; its use is not recommended [Loosemore 2007].

Compliant Solution

This compliant solution replaces the VLA with a call to `malloc()`. If `malloc()` fails, the return value can be checked to prevent the program from terminating abnormally.

```
int copy_file(FILE *src, FILE *dst, size_t bufsize) {
  if (bufsize == 0) {
    /* Handle error */
  }
  char *buf = (char *)malloc(bufsize);
  if (!buf) {
    /* Handle error */
  }

  while (fgets(buf, bufsize, src)) {
    if (fputs(buf, dst) == EOF) {
      /* Handle error */
    }
  }
  /* ... */
  free(buf);
  return 0;
}
```

Noncompliant Code Example

Recursion can also lead to large stack allocations. Recursive functions must ensure that they do not exhaust the stack as a result of excessive recursions.

This noncompliant implementation of the Fibonacci function uses recursion:

```
unsigned long fib1(unsigned int n) {
  if (n == 0) {
    return 0;
  }
  else if (n == 1 || n == 2) {
    return 1;
  }
  else {
    return fib1(n-1) + fib1(n-2);
  }
}
```

The amount of stack space needed grows linearly with respect to the parameter n. Large values of n have been shown to cause abnormal program termination.

Compliant Solution

This implementation of the Fibonacci functions eliminates the use of recursion:

```
unsigned long fib2(unsigned int n) {
  if (n == 0) {
    return 0;
  }
  else if (n == 1 || n == 2) {
    return 1;
  }

  unsigned long prev = 1;
  unsigned long cur = 1;

  unsigned int i;

  for (i = 3; i <= n; i++) {
    unsigned long tmp = cur;
    cur = cur + prev;
    prev = tmp;
  }

  return cur;
}
```

Because there is no recursion, the amount of stack space needed does not depend on the parameter n, greatly reducing the risk of stack overflow.

Risk Assessment

Program stacks are frequently used for convenient temporary storage because allocated memory is automatically freed when the function returns. Generally, the operating system grows the stack as needed. However, growing the stack can fail because of a lack of memory or a collision with other allocated areas of the address space (depending on the architecture). When the stack is exhausted, the operating system can terminate the program abnormally. This behavior can be exploited, and an attacker can cause a denial-of-service attack if he or she can control or influence the amount of stack memory allocated.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
MEM05-C	Medium	Likely	Medium	P12	L1

Automated Detection

Tool	Version	Checker	Description
CodeSonar	5.1p0	**IO.TAINT.SIZE** **MISC.MEM.SIZE.BAD**	Tainted Allocation Size Unreasonable Size Argument
Coverity	2017.07	**STACK_USE**	Can help detect single stack allocations that are dangerously large, although it will not detect excessive stack use resulting from recursion

Klocwork	2018	**MISRA.FUNC.RECUR**	
LDRA tool suite	9.7.1	**44 S**	Enhanced Enforcement
Parasoft C/C++test	10.4.2	**CERT_C-MEM05-a** **CERT_C-MEM05-b**	Do not use recursion Ensure the size of the variable length array is in valid range
Polyspace Bug Finder	R2019b	CERT C: Rec. MEM05-C	Checks for: • Direct or indirect function call to itself • Variable length array with nonpositive size • Tainted size of variable length array Rec. partially covered.
PRQA QA-C	9.7	**1051, 1520, 3670**	Partially implemented
PVS-Studio	6.23	**V505**	

Related Vulnerabilities

Stack overflow has been implicated in Toyota unintended acceleration cases, where Camry and other Toyota vehicles accelerated unexpectedly. Michael Barr testified at the trial that a stack overflow could corrupt the critical variables of the operating system, because they were located in memory adjacent to the top of the stack [Samek 2014].

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

Related Guidelines

SEI CERT C++ Coding Standard	VOID MEM05-CPP. Avoid large stack allocations
ISO/IEC TR 24772:2013	Recursion [GDL]
MISRA C:2012	Rule 17.2 (required)

Bibliography

[Loosemore 2007]	Section 3.2.5, "Automatic Storage with Variable Size"
[Samek 2014]	Are We Shooting Ourselves in the Foot with Stack Overflow? Monday, February 17th, 2014 by Miro Samek
[Seacord 2013]	Chapter 4, "Dynamic Memory Management"