black hat USA 2021

August 4-5, 2021

BRIEFINGS

ERROR: BadAlloc! - Broken Memory Allocators Led to Millions of Vulnerable IoT and Embedded Devices

Omri Ben-Bassat

Security Researcher

Azure Defender for IoT

Microsoft

Tamir Ariel

Security Researcher

Azure Defender for IoT

Microsoft

Index

Quick Reminder
Integer Overflows

Sum

$8 + 8 = ??$

Sum

8 + 8 = 88

Sum

8 + 8 = 88

Sum

8 + 8 = 16

Sum

8 + 8 = 16

4,294,967,295$(2^{32} - 1)$ **+ 8 = 7**

Sum

8 + 8 = 16

4,294,967,295$(2^{32} - 1)$ + 8 = 7

(on 32-bit systems)

Multiplication

$$2 * 2 = 4$$

$$2{,}147{,}483{,}649\left(\frac{2^{32}}{2} + 1\right) * 2 = 2$$

Multiplication

$$2 * 2 = 4$$

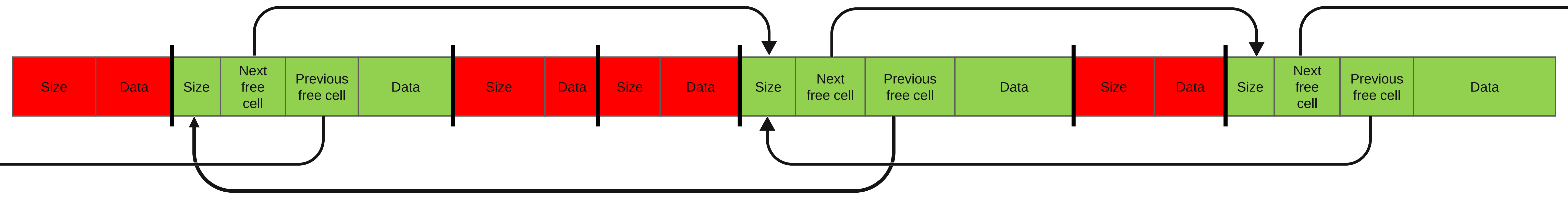
$$2{,}147{,}483{,}649\left(\frac{2^{32}}{2} + 1\right) * 2 = 2$$

(on 32-bit systems)

Memory Allocator 101

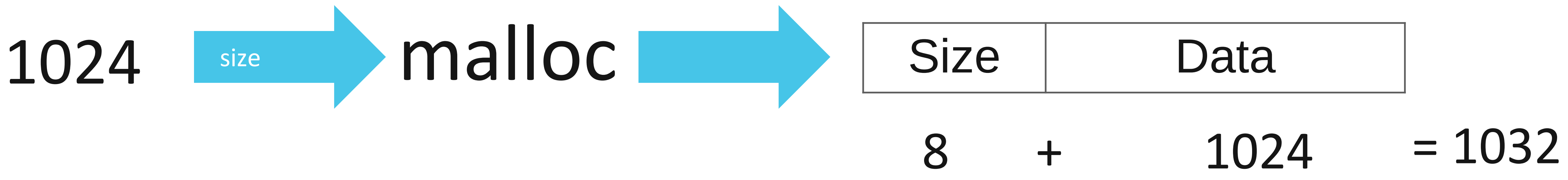
Heap layout

- Free memory is managed by the allocator using a single/double linked list of free blocks.

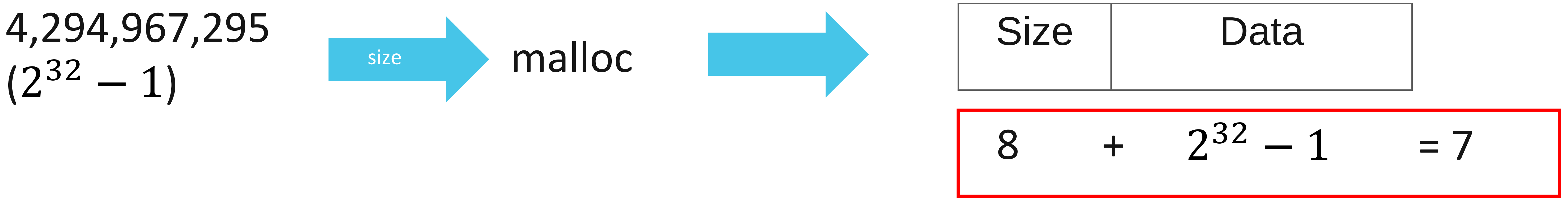


🟥 Allocated 🟩 Free

Calculating total block size

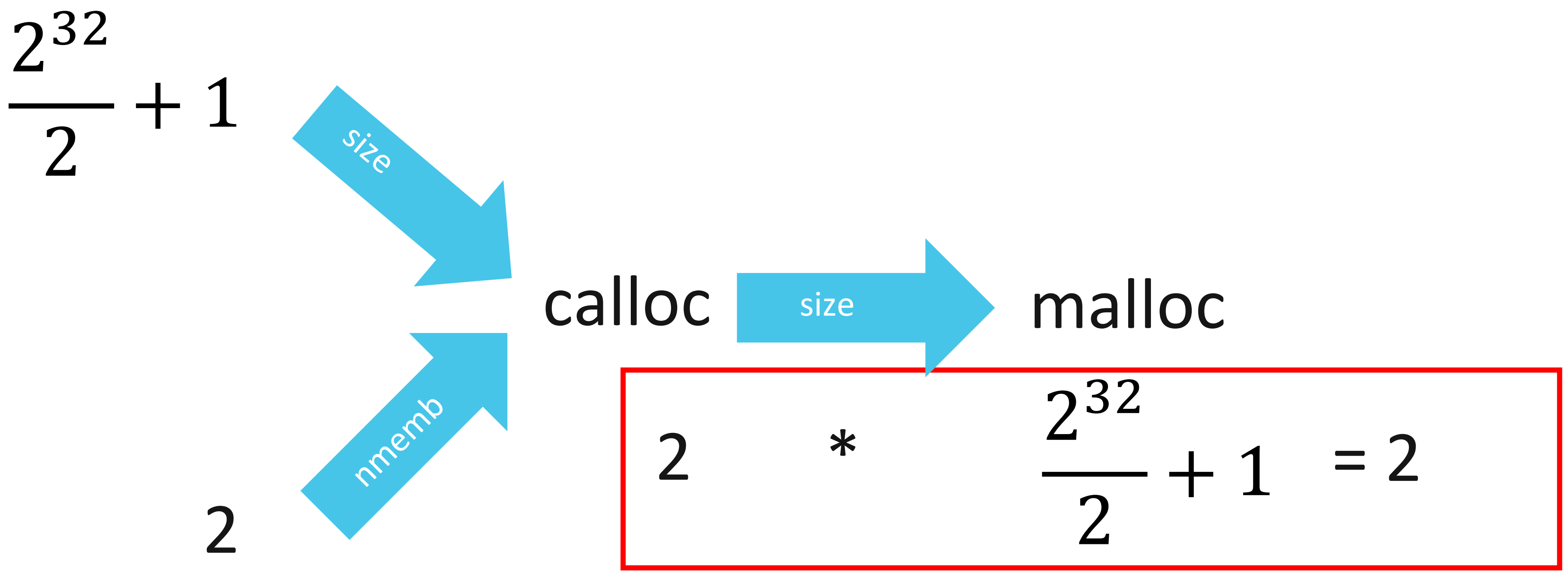


What will happen if I ask for a large amount of space?



$$8 \quad + \quad 2^{32} - 1 \quad = 7$$

Calloc



$$2 \quad * \quad \frac{2^{32}}{2}+1 = 2$$

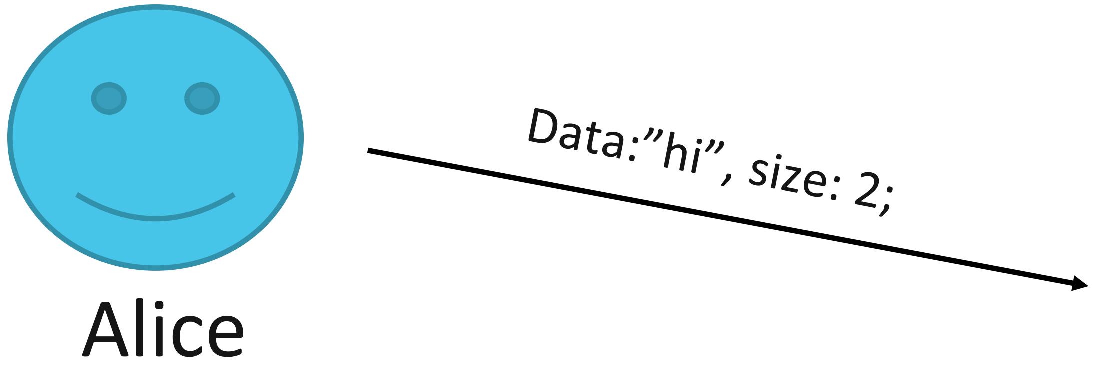
Good alloc

Alice

Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(**size**)
        If (buf != NULL)
                memcpy(buf, user_data, **size**)
                return “ok, thank you!”
        else
                return “sorry too much data”
…
```

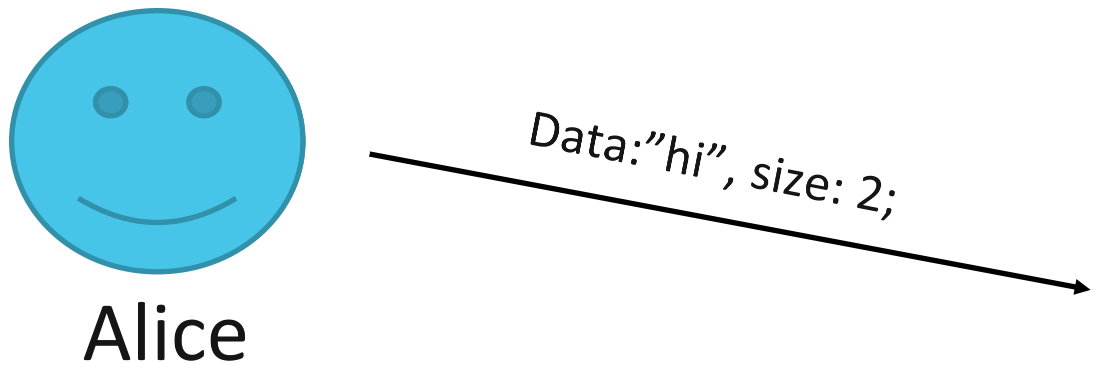
Good alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(**size**)
        If (buf != NULL)
                memcpy(buf, user_data, **size**)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

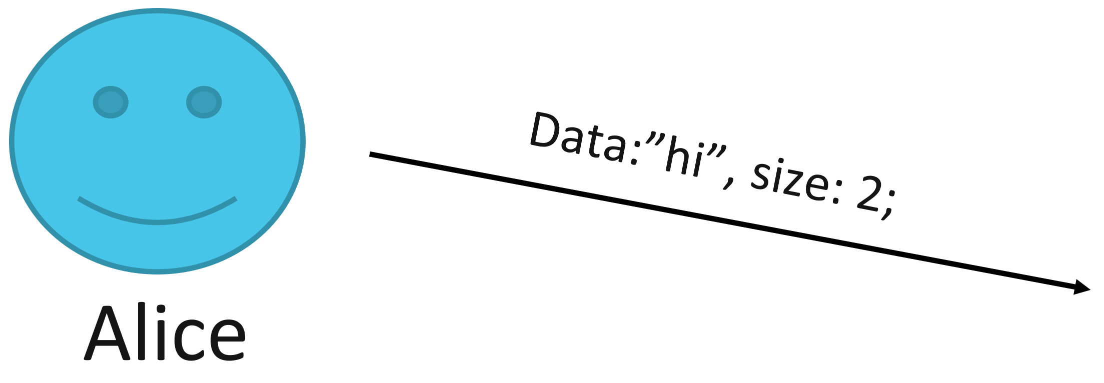
Good alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(**size**)
        If (buf != NULL)
                memcpy(buf, user_data, **size**)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

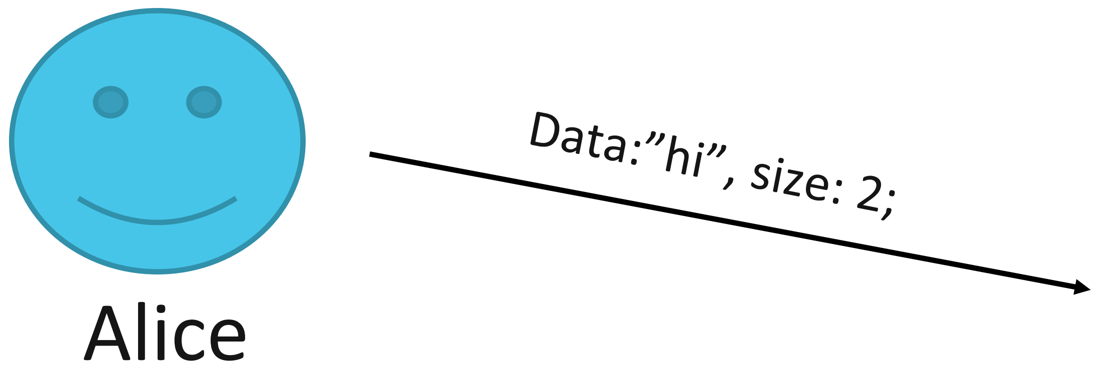
Good alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(size)
        If (buf != NULL)
                memcpy(buf, user_data, size)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

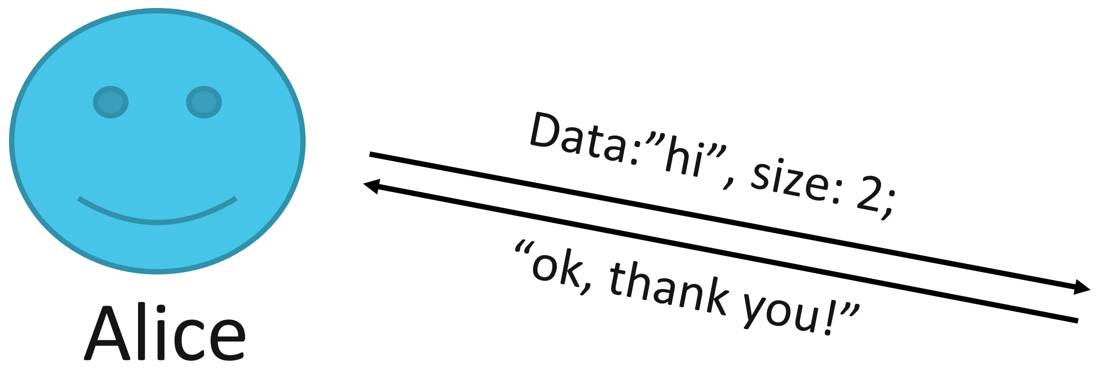
Good alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(**size**)
        If (buf != NULL)
                memcpy(buf, user_data, **size**)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

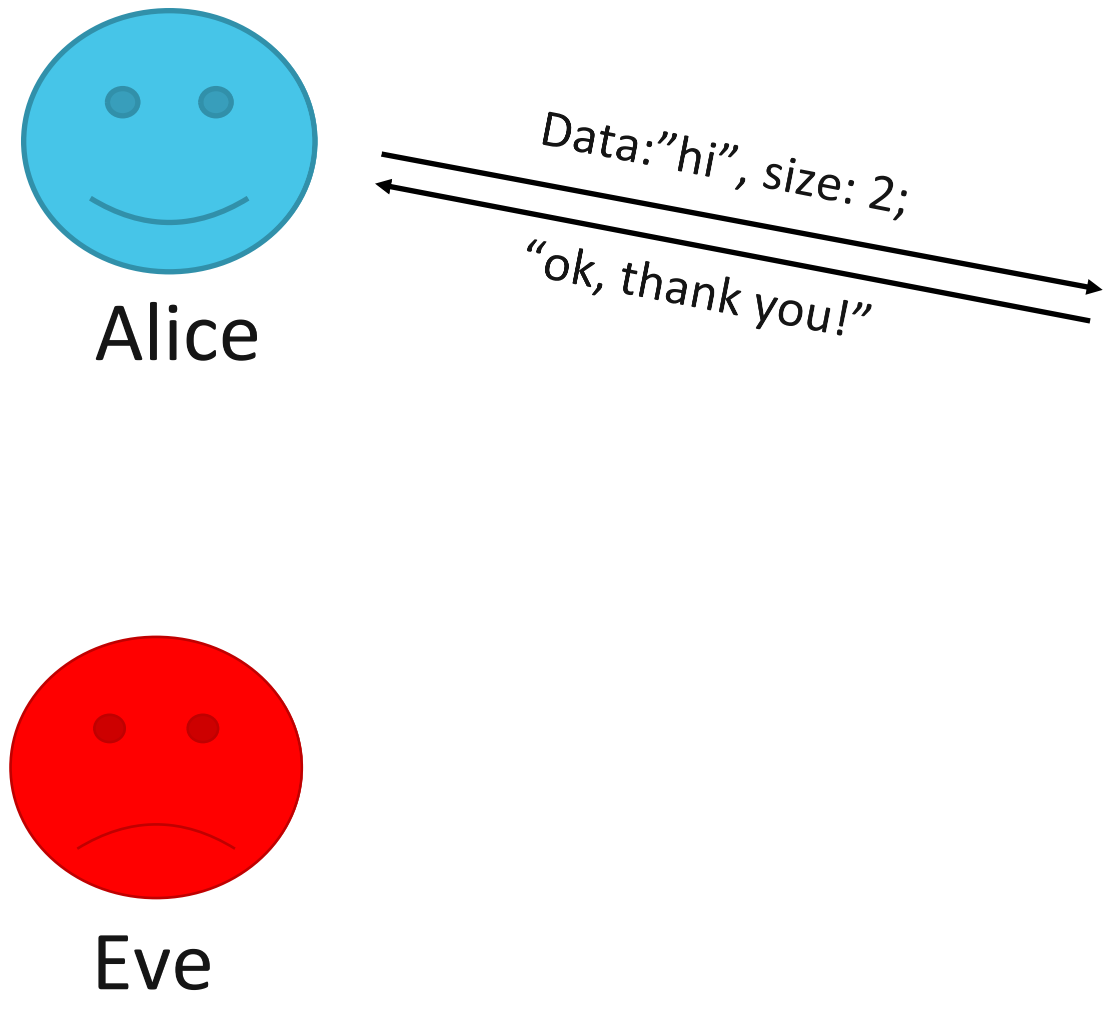
Good alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(size)
        If (buf != NULL)
                memcpy(buf, user_data, size)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

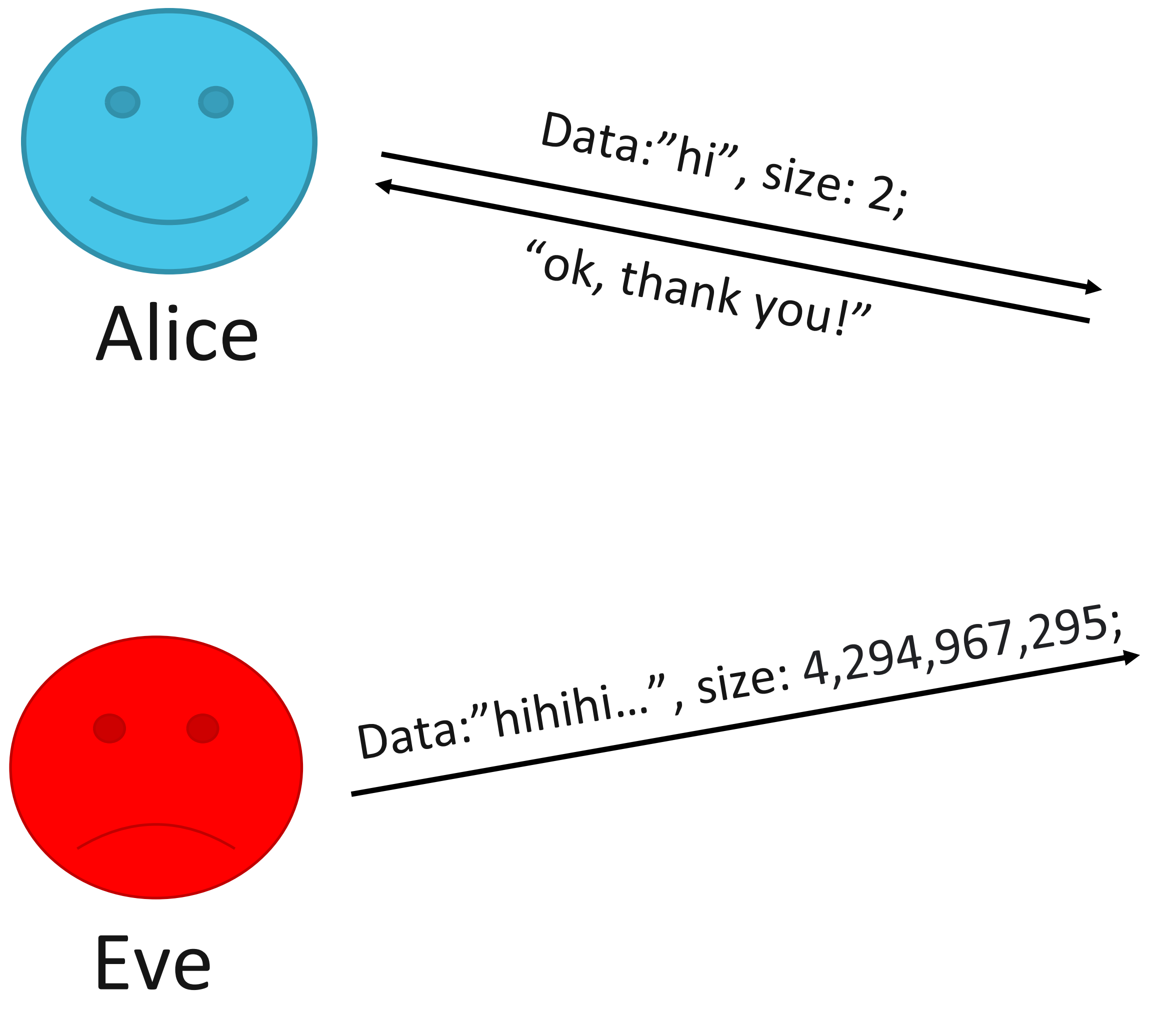
Good alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(**size**)
        If (buf != NULL)
                memcpy(buf, user_data, **size**)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

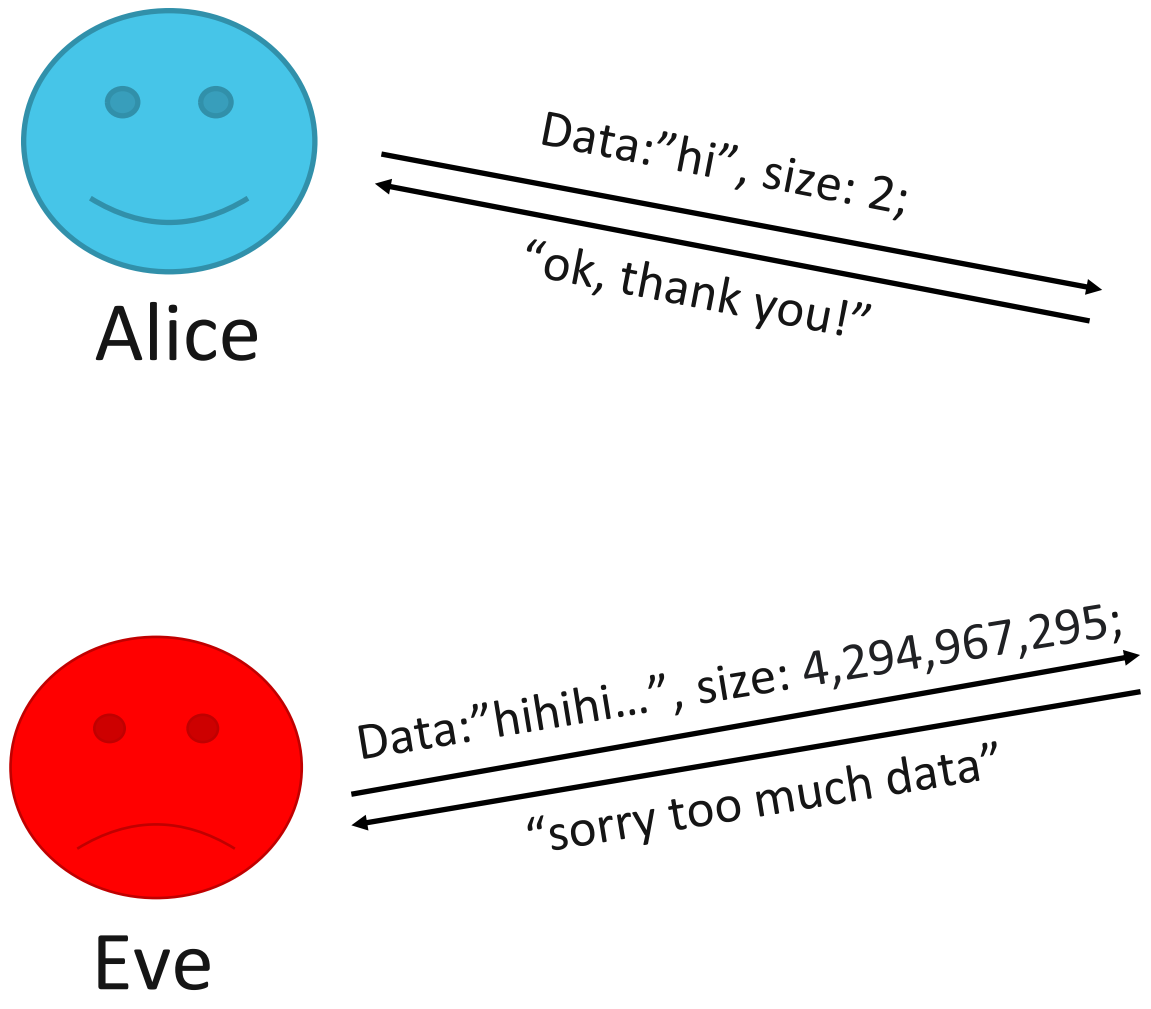
Good alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(size)
        If (buf != NULL)
                memcpy(buf, user_data, size)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

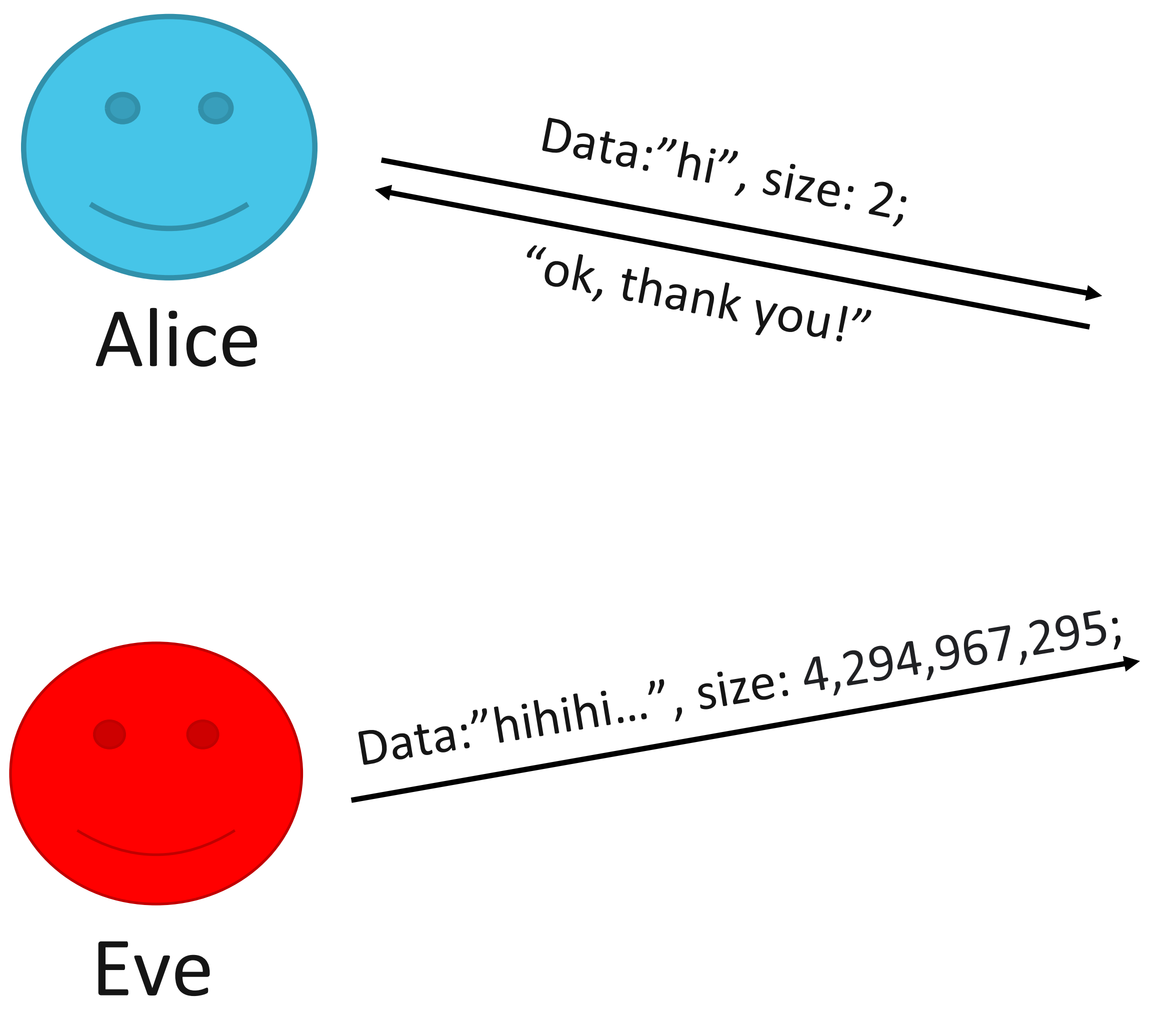
Good alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = malloc(size)
        If (buf != NULL)
                memcpy(buf, user_data, size)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

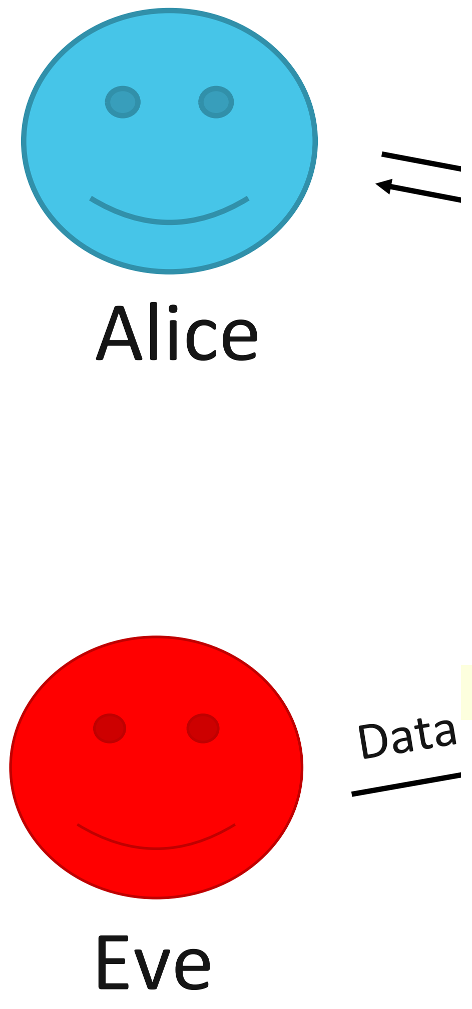
Bad alloc

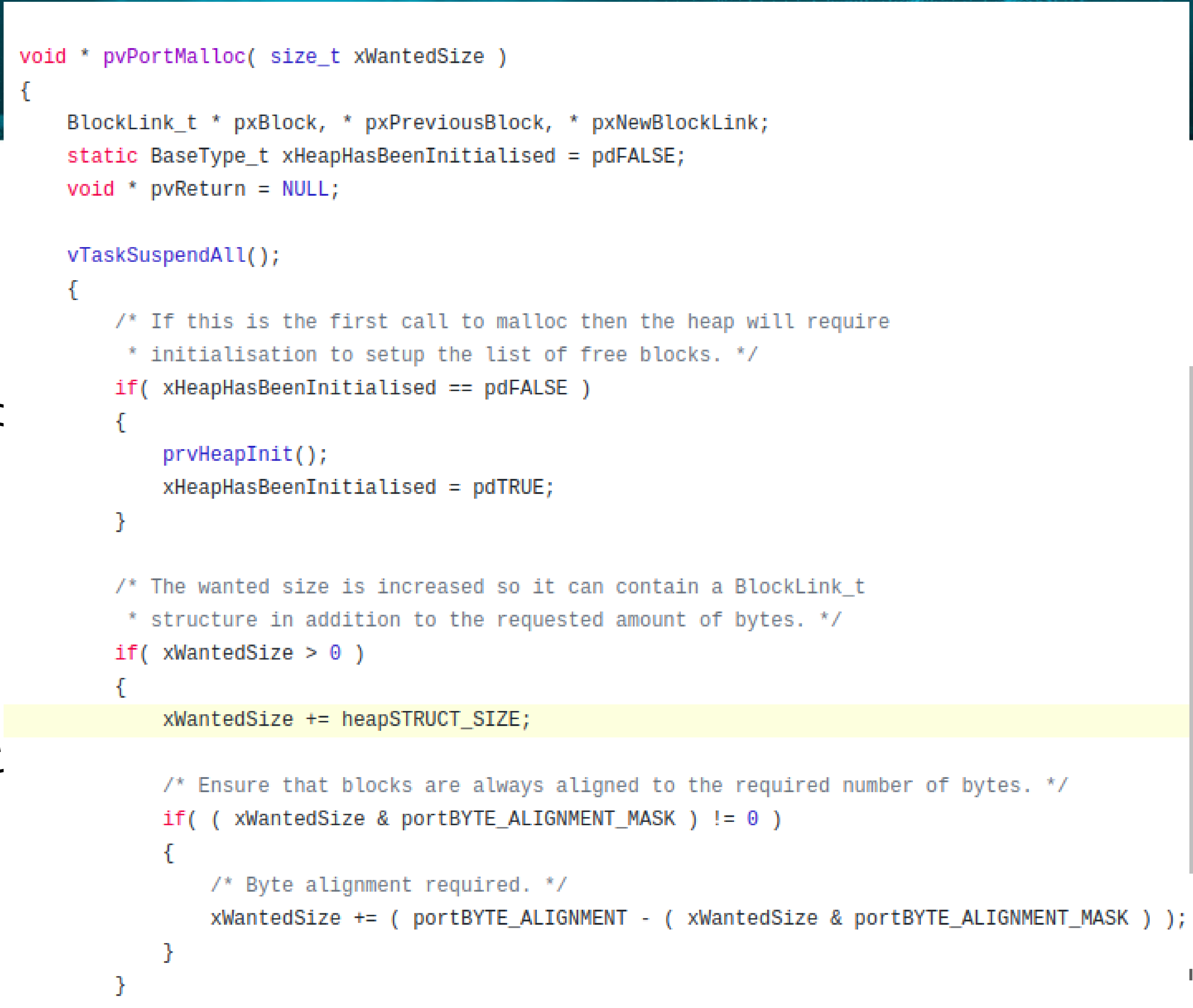


Server

```
Read_user_data(user_data, size)
…
        Buf = bad_malloc(size)
        If (buf != NULL)
                memcpy(buf, user_data, size)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

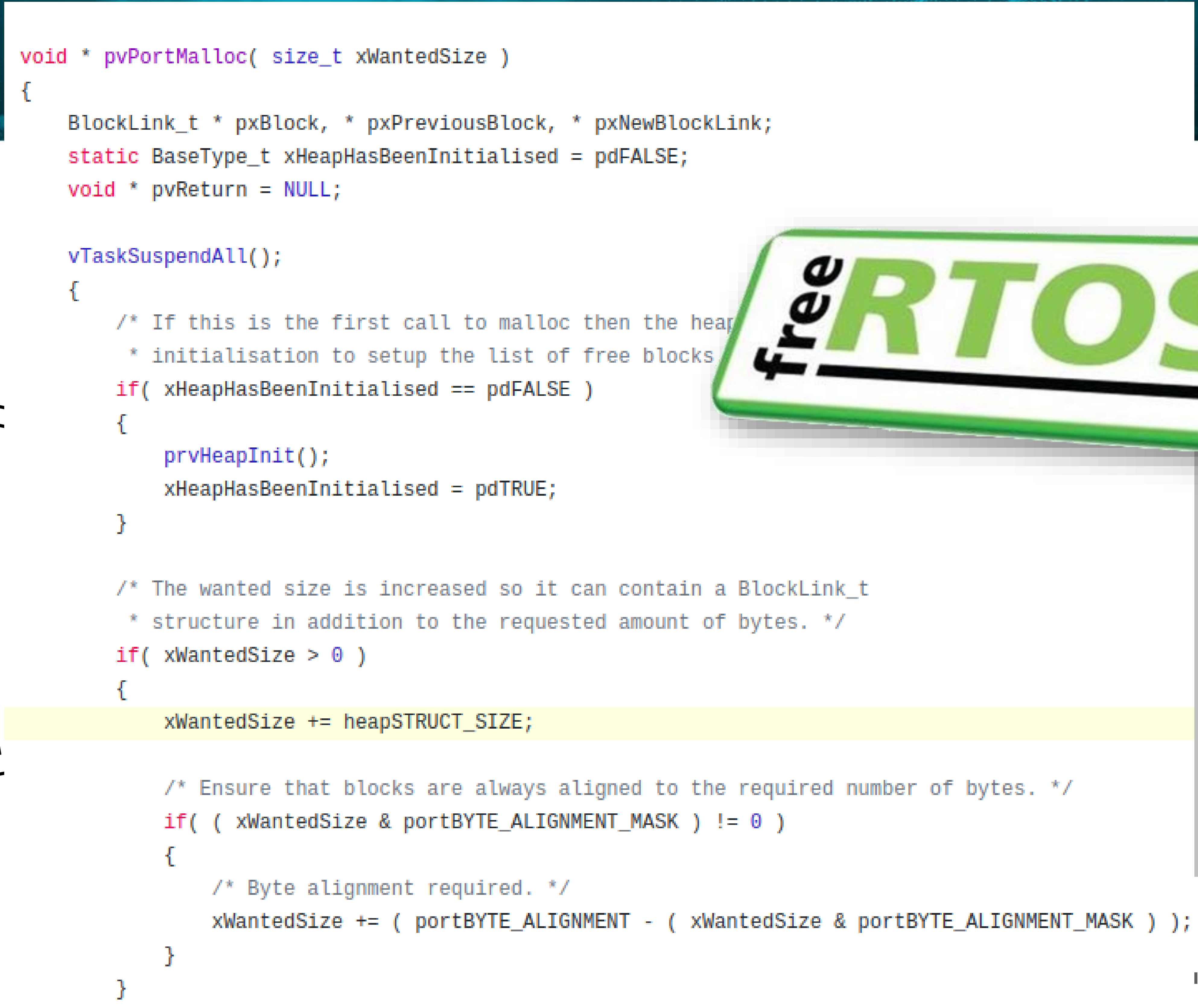
Bad alloc





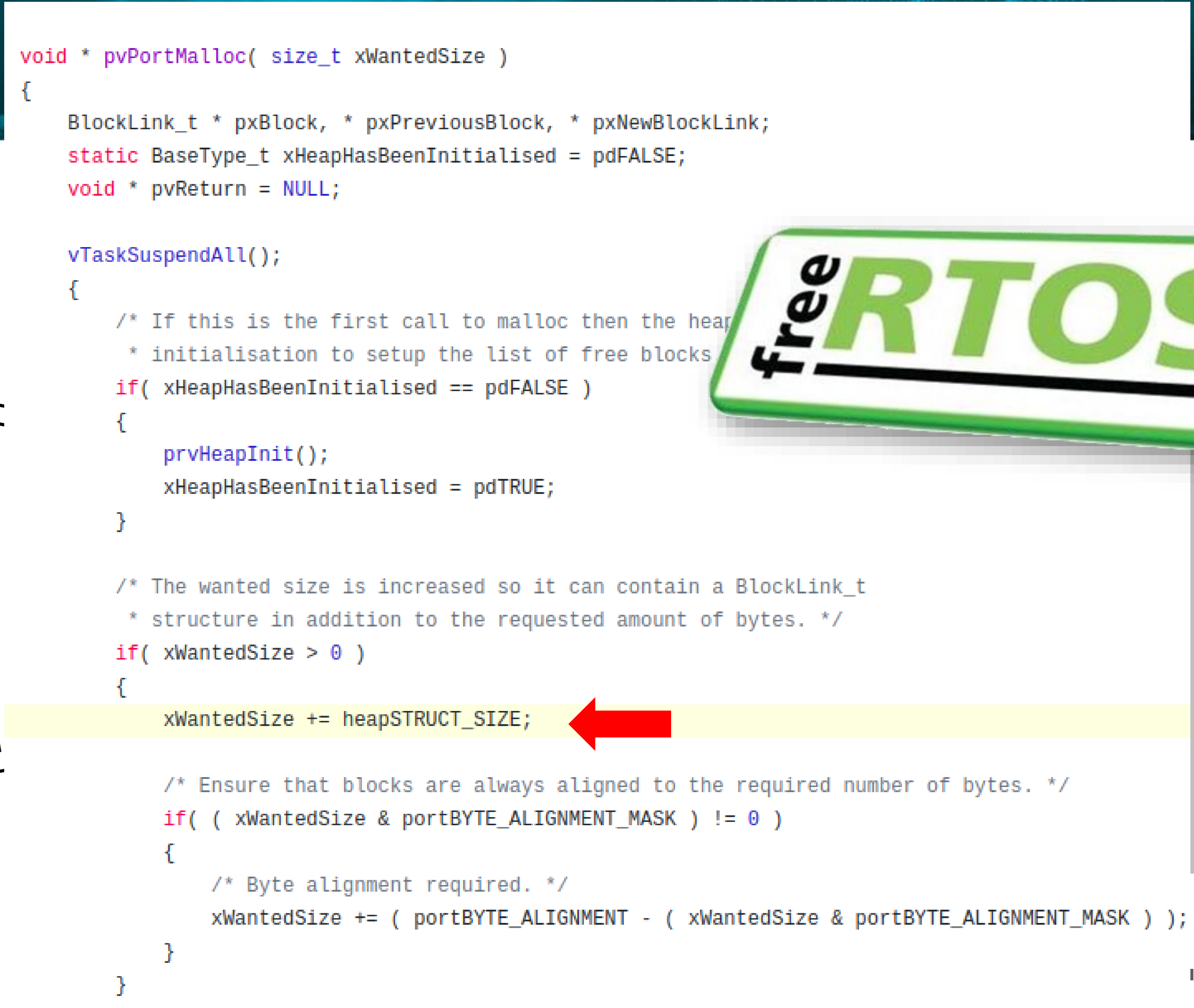
```
void * pvPortMalloc( size_t xWantedSize )
{
    BlockLink_t * pxBlock, * pxPreviousBlock, * pxNewBlockLink;
    static BaseType_t xHeapHasBeenInitialised = pdFALSE;
    void * pvReturn = NULL;

    vTaskSuspendAll();
    {
        /* If this is the first call to malloc then the heap will require
         * initialisation to setup the list of free blocks. */
        if( xHeapHasBeenInitialised == pdFALSE )
        {
            prvHeapInit();
            xHeapHasBeenInitialised = pdTRUE;
        }

        /* The wanted size is increased so it can contain a BlockLink_t
         * structure in addition to the requested amount of bytes. */
        if( xWantedSize > 0 )
        {
            xWantedSize += heapSTRUCT_SIZE;

            /* Ensure that blocks are always aligned to the required number of bytes. */
            if( ( xWantedSize & portBYTE_ALIGNMENT_MASK ) != 0 )
            {
                /* Byte alignment required. */
                xWantedSize += ( portBYTE_ALIGNMENT - ( xWantedSize & portBYTE_ALIGNMENT_MASK ) );
            }
        }
```

Bad alloc



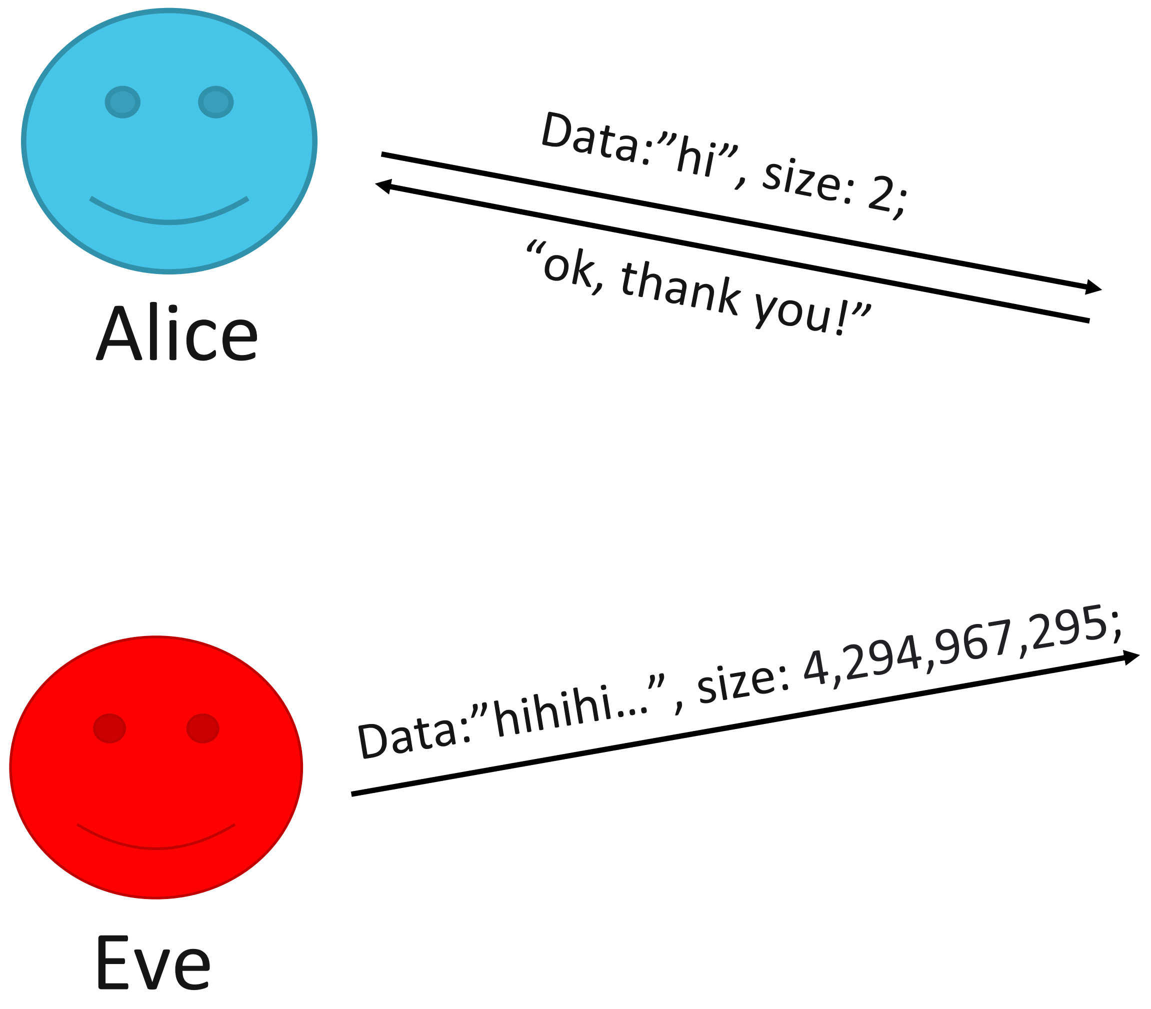
```
void * pvPortMalloc( size_t xWantedSize )
{
    BlockLink_t * pxBlock, * pxPreviousBlock, * pxNewBlockLink;
    static BaseType_t xHeapHasBeenInitialised = pdFALSE;
    void * pvReturn = NULL;

    vTaskSuspendAll();
    {
        /* If this is the first call to malloc then the heap
         * initialisation to setup the list of free blocks
        if( xHeapHasBeenInitialised == pdFALSE )
        {
            prvHeapInit();
            xHeapHasBeenInitialised = pdTRUE;
        }

        /* The wanted size is increased so it can contain a BlockLink_t
         * structure in addition to the requested amount of bytes. */
        if( xWantedSize > 0 )
        {
            xWantedSize += heapSTRUCT_SIZE;

            /* Ensure that blocks are always aligned to the required number of bytes. */
            if( ( xWantedSize & portBYTE_ALIGNMENT_MASK ) != 0 )
            {
                /* Byte alignment required. */
                xWantedSize += ( portBYTE_ALIGNMENT - ( xWantedSize & portBYTE_ALIGNMENT_MASK ) );
            }
        }
```

Bad alloc



```
void * pvPortMalloc( size_t xWantedSize )
{
    BlockLink_t * pxBlock, * pxPreviousBlock, * pxNewBlockLink;
    static BaseType_t xHeapHasBeenInitialised = pdFALSE;
    void * pvReturn = NULL;

    vTaskSuspendAll();
    {
        /* If this is the first call to malloc then the heap
         * initialisation to setup the list of free blocks
        if( xHeapHasBeenInitialised == pdFALSE )
        {
            prvHeapInit();
            xHeapHasBeenInitialised = pdTRUE;
        }

        /* The wanted size is increased so it can contain a BlockLink_t
         * structure in addition to the requested amount of bytes. */
        if( xWantedSize > 0 )
        {
            xWantedSize += heapSTRUCT_SIZE;

            /* Ensure that blocks are always aligned to the required number of bytes. */
            if( ( xWantedSize & portBYTE_ALIGNMENT_MASK ) != 0 )
            {
                /* Byte alignment required. */
                xWantedSize += ( portBYTE_ALIGNMENT - ( xWantedSize & portBYTE_ALIGNMENT_MASK ) );
            }
        }
```

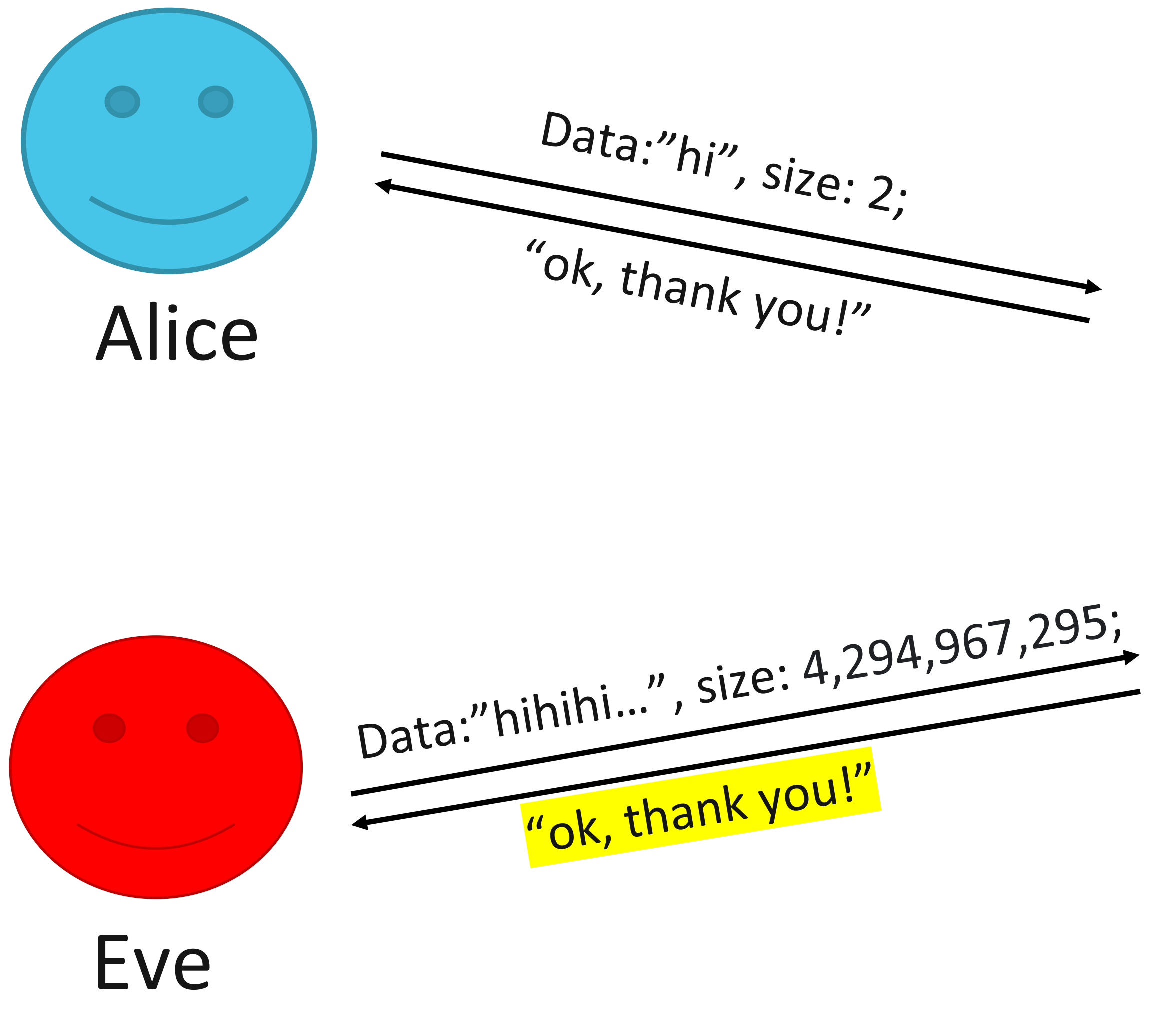
Bad alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = bad_malloc(**size**)
        If (buf != NULL)
                memcpy(buf, user_data, **size**)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

Bad alloc



Server

```
Read_user_data(user_data, size)
…
        Buf = bad_malloc(size)
        If (buf != NULL)
                memcpy(buf, user_data, size)
                return "ok, thank you!"
        else
                return "sorry too much data"
…
```

Bad a

Alic

Eve

```
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
hihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihihi
```

Affected Products

eCosCentric®

MEDIATEK

Tencent 腾讯

Google

arm

freeRTOS

NXP

MONGOOSE OS

RIOT

VxWorks

Texas Instruments

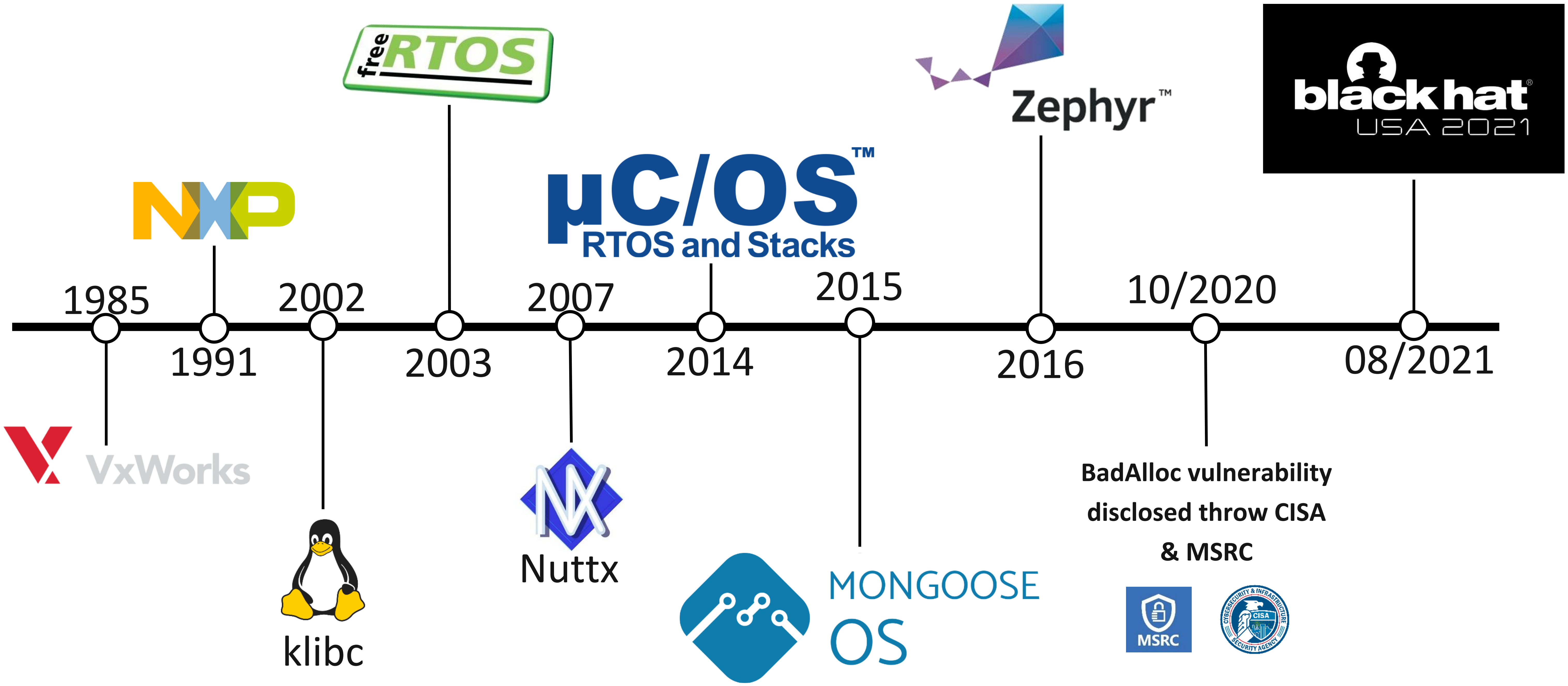
redis

Zephyr™

µC/OS™ RTOS and Stacks

Red Hat

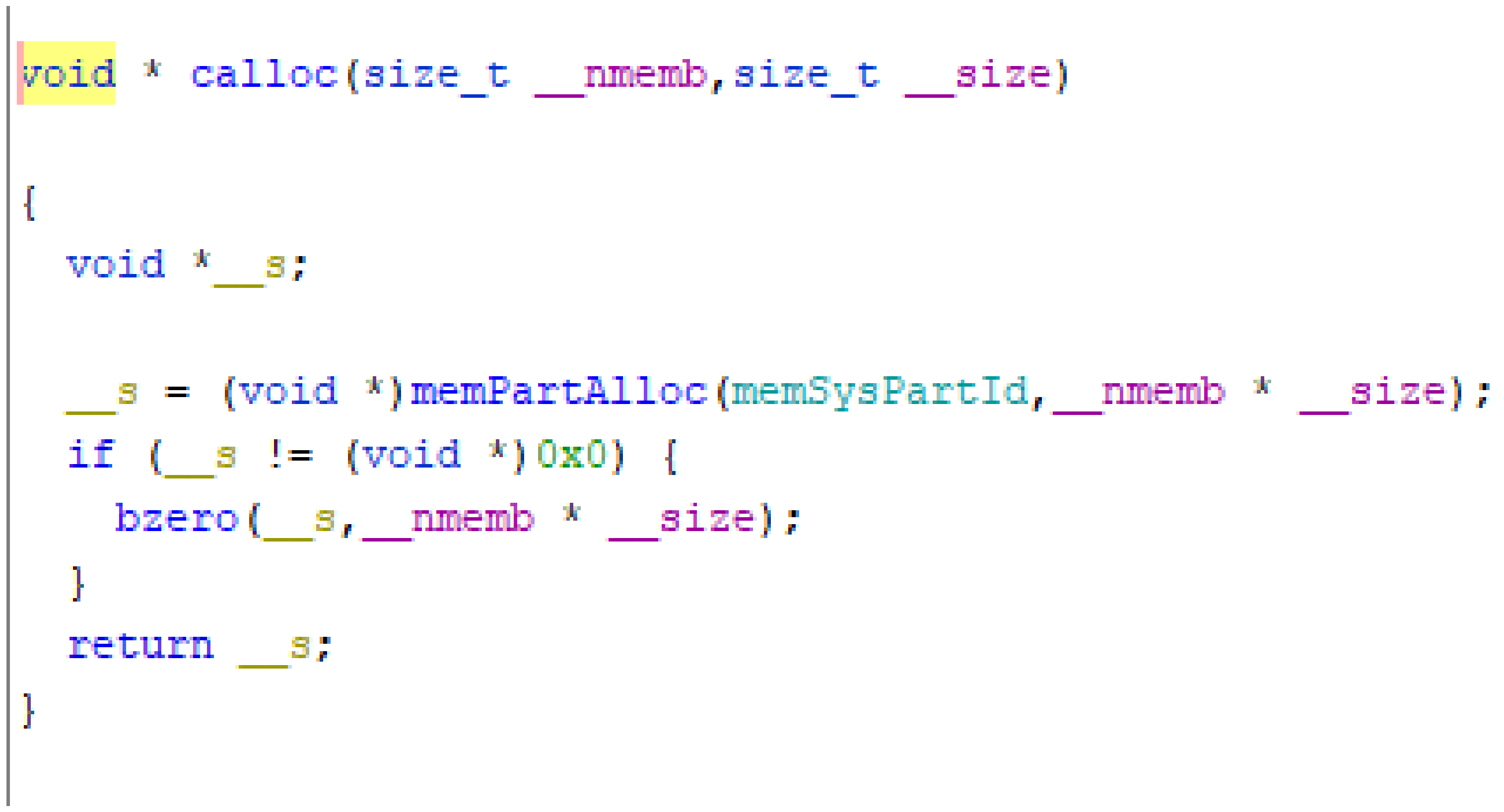
SAMSUNG



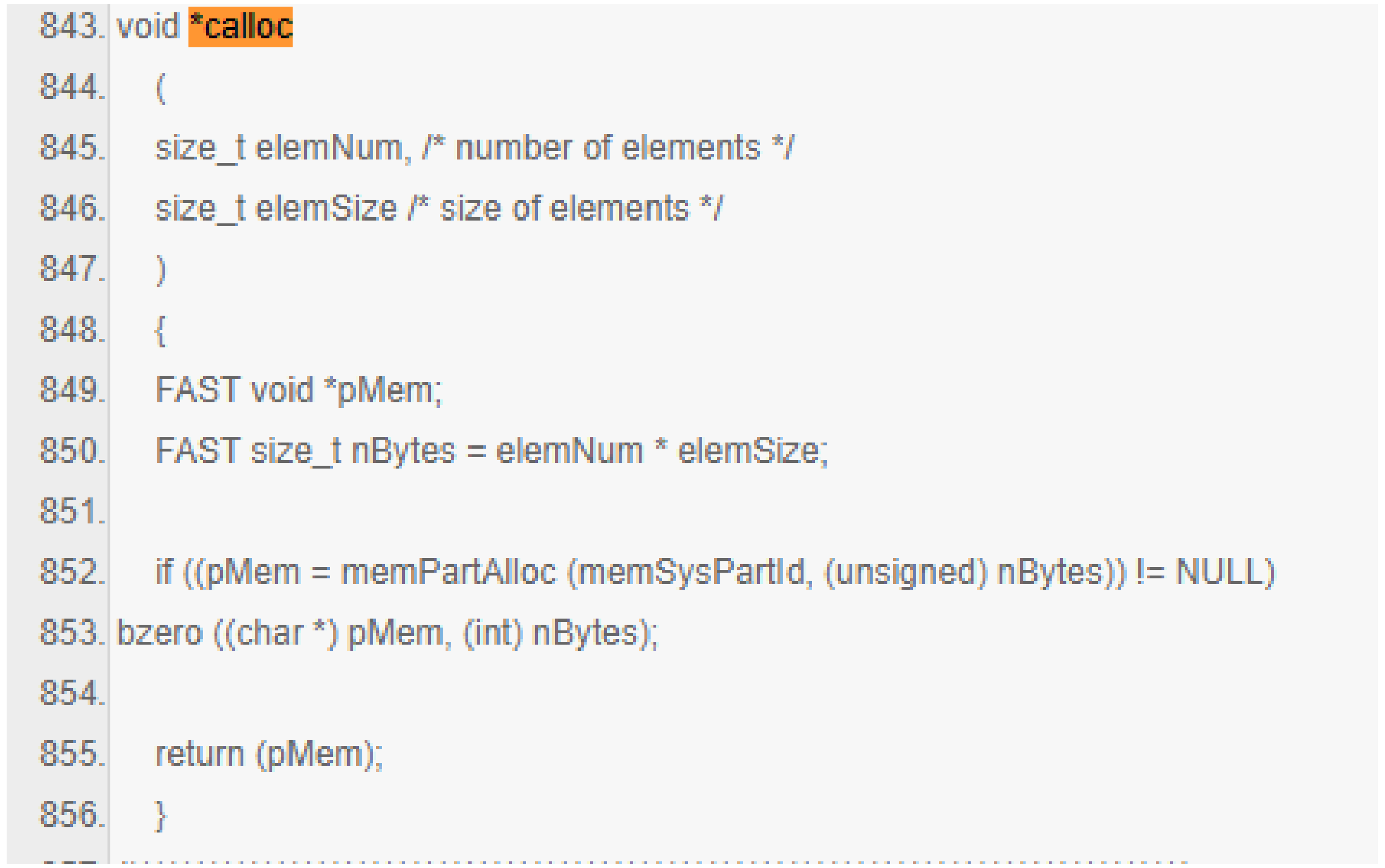
BadAlloc vulnerability disclosed throw CISA & MSRC

Notable Examples

VxWorks 5.1 - 1993

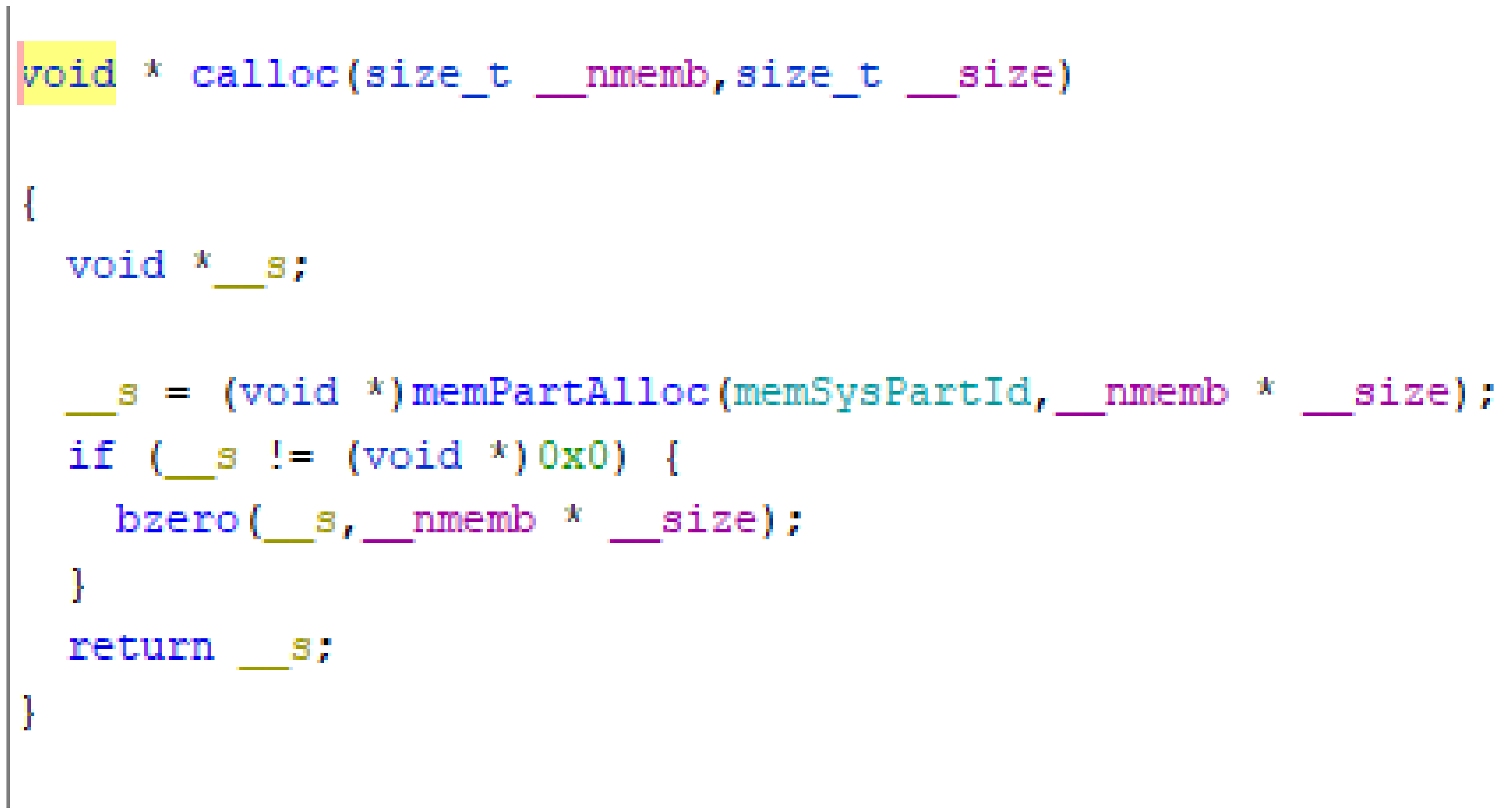


```
void * calloc(size_t __nmemb,size_t __size)

{
  void *__s;

  __s = (void *)memPartAlloc(memSysPartId,__nmemb * __size);
  if (__s != (void *)0x0) {
    bzero(__s,__nmemb * __size);
  }
  return __s;
}
```

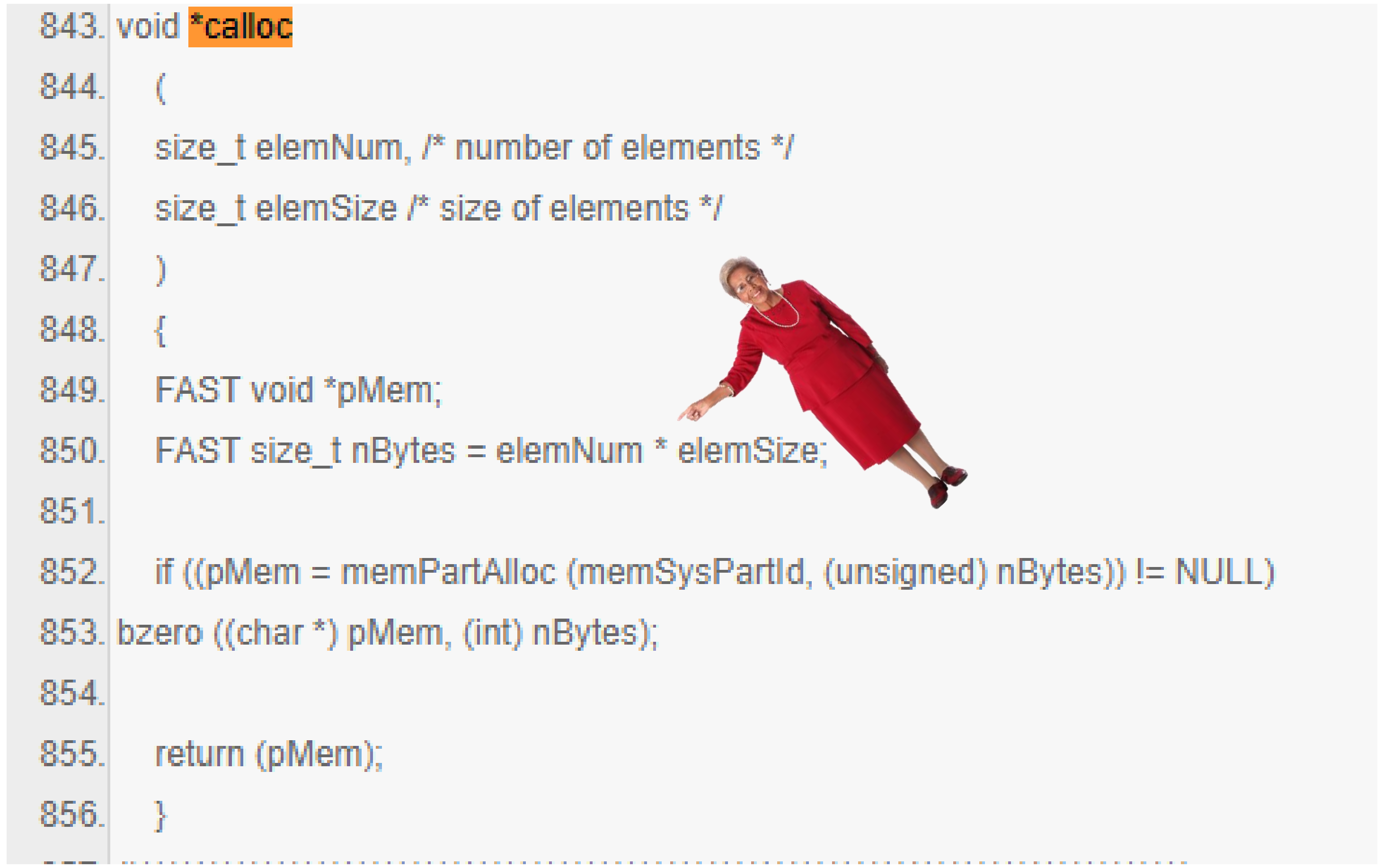


```
void *calloc
    (
    size_t elemNum, /* number of elements */
    size_t elemSize /* size of elements */
    )
    {
    FAST void *pMem;
    FAST size_t nBytes = elemNum * elemSize;

    if ((pMem = memPartAlloc (memSysPartId, (unsigned) nBytes)) != NULL)
bzero ((char *) pMem, (int) nBytes);

    return (pMem);
    }
```

VxWorks 5.1 - 1993

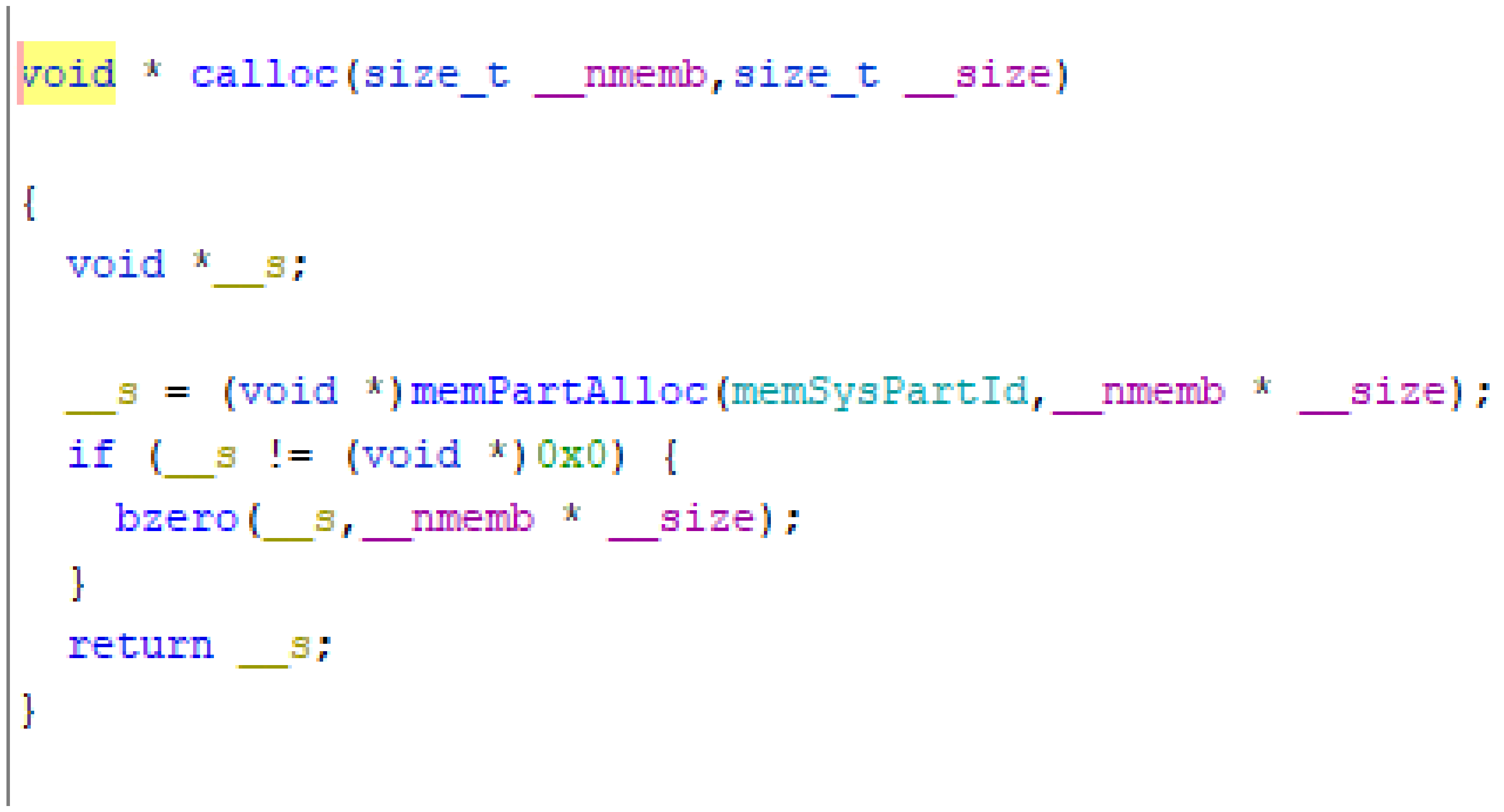


```
void * calloc(size_t __nmemb,size_t __size)

{
  void *__s;

  __s = (void *)memPartAlloc(memSysPartId,__nmemb * __size);
  if (__s != (void *)0x0) {
    bzero(__s,__nmemb * __size);
  }
  return __s;
}
```

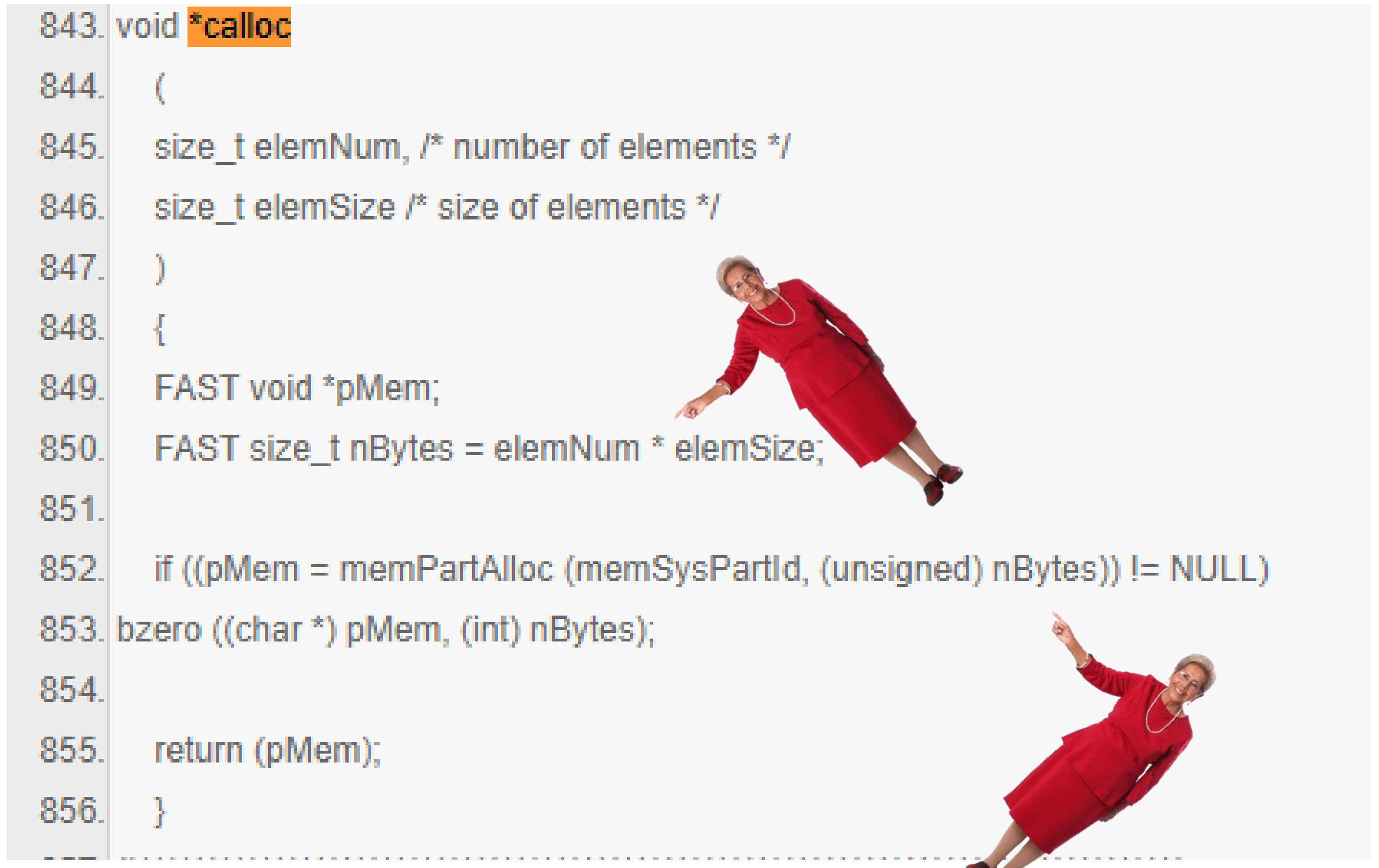


```
void *calloc
    (
    size_t elemNum, /* number of elements */
    size_t elemSize /* size of elements */
    )
    {
    FAST void *pMem;
    FAST size_t nBytes = elemNum * elemSize;

    if ((pMem = memPartAlloc (memSysPartId, (unsigned) nBytes)) != NULL)
bzero ((char *) pMem, (int) nBytes);

    return (pMem);
    }
```

VxWorks 5.1 - 1993

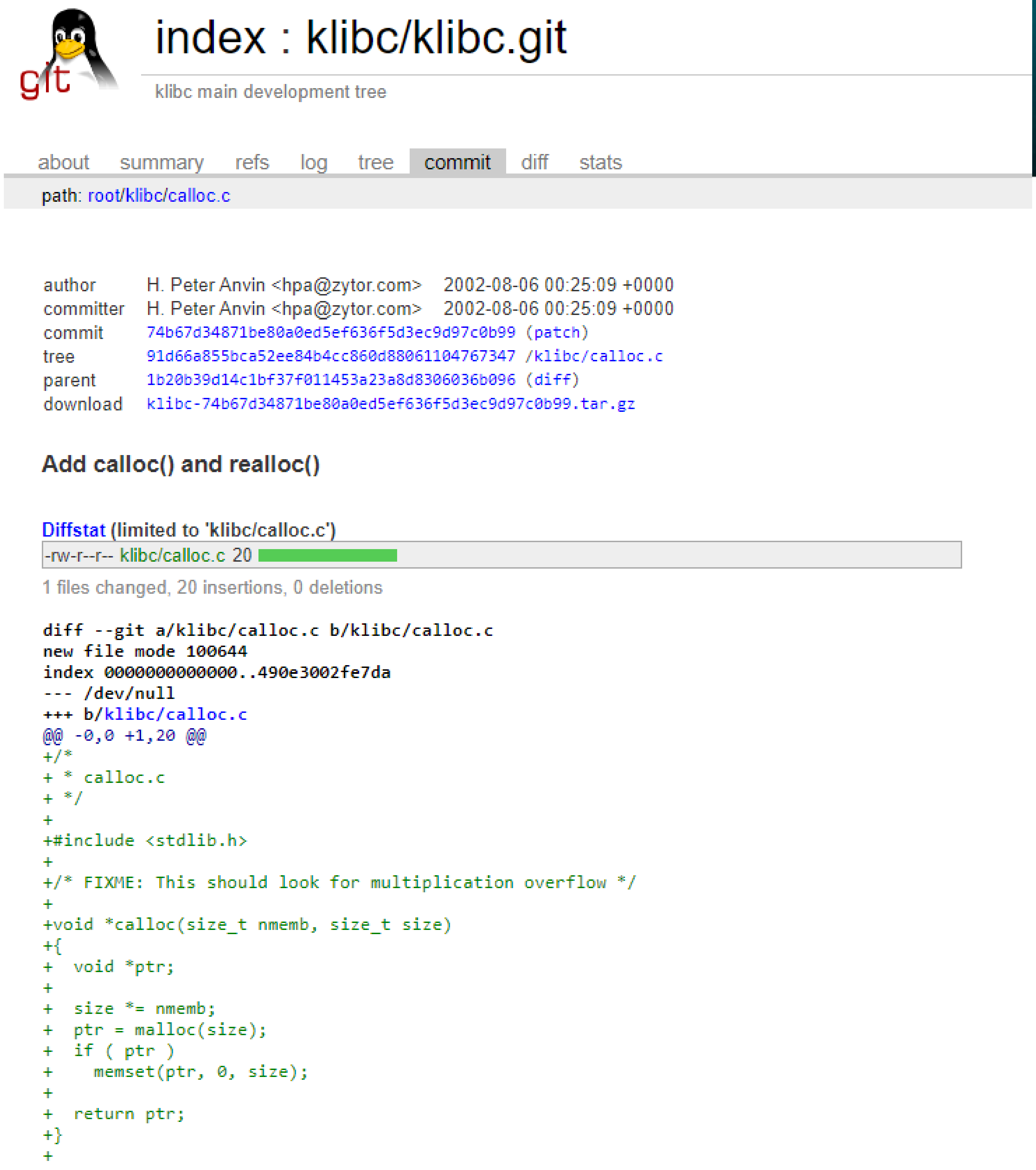


```
void * calloc(size_t __nmemb,size_t __size)

{
  void *__s;

  __s = (void *)memPartAlloc(memSysPartId,__nmemb * __size);
  if (__s != (void *)0x0) {
    bzero(__s,__nmemb * __size);
  }
  return __s;
}
```



```
void *calloc
    (
    size_t elemNum, /* number of elements */
    size_t elemSize /* size of elements */
    )
    {
    FAST void *pMem;
    FAST size_t nBytes = elemNum * elemSize;

    if ((pMem = memPartAlloc (memSysPartId, (unsigned) nBytes)) != NULL)
bzero ((char *) pMem, (int) nBytes);

    return (pMem);
    }
```

Klibc – 2002



index : klibc/klibc.git

klibc main development tree

path: root/klibc/calloc.c

author	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
committer	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
commit	74b67d34871be80a0ed5ef636f5d3ec9d97c0b99 (patch)	
tree	91d66a855bca52ee84b4cc860d88061104767347 /klibc/calloc.c	
parent	1b20b39d14c1bf37f011453a23a8d8306036b096 (diff)	
download	klibc-74b67d34871be80a0ed5ef636f5d3ec9d97c0b99.tar.gz	

Add calloc() and realloc()

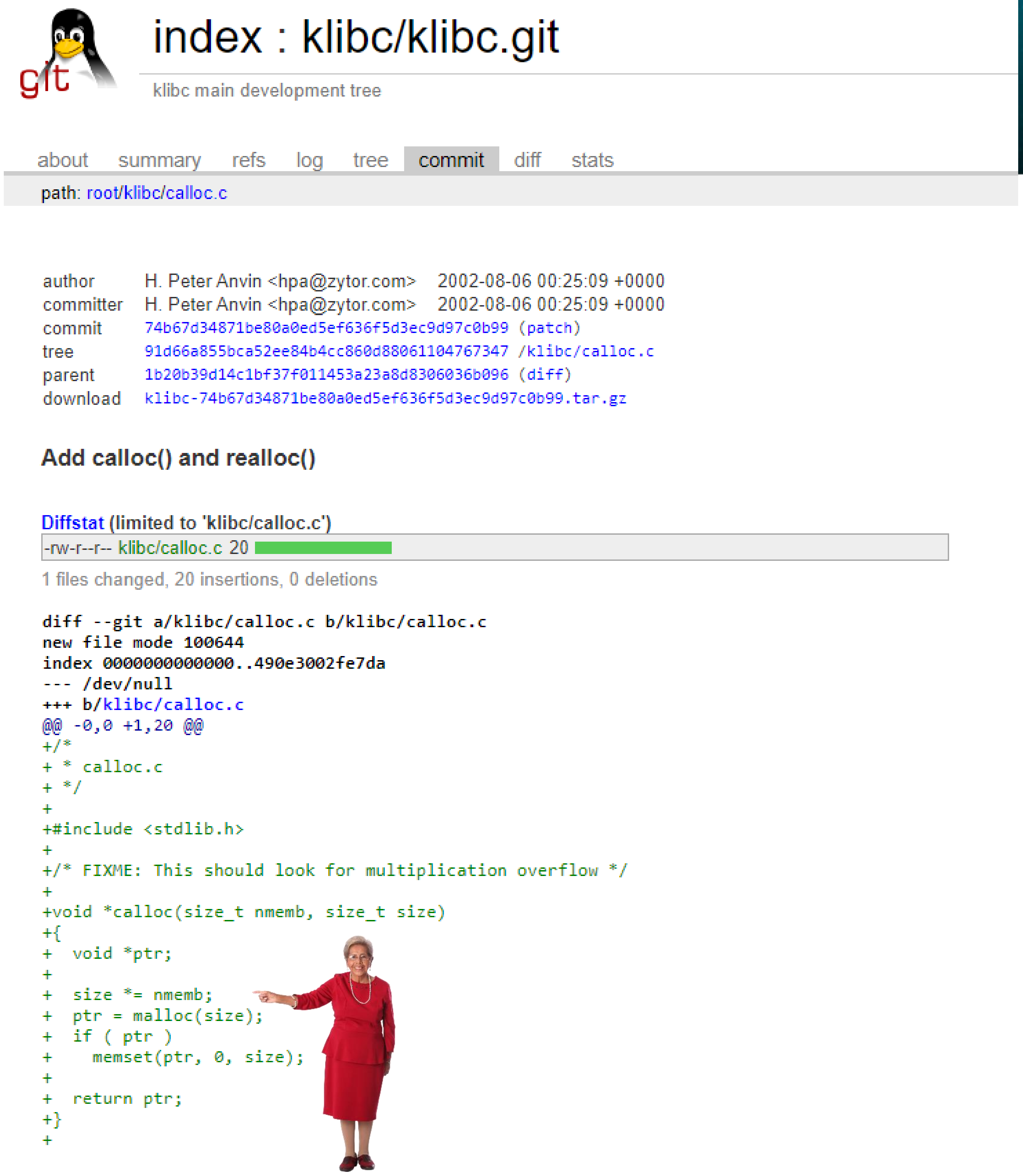
Diffstat (limited to 'klibc/calloc.c')

-rw-r--r-- klibc/calloc.c 20

1 files changed, 20 insertions, 0 deletions

```
diff --git a/klibc/calloc.c b/klibc/calloc.c
new file mode 100644
index 0000000000000..490e3002fe7da
--- /dev/null
+++ b/klibc/calloc.c
@@ -0,0 +1,20 @@
+/*
+ * calloc.c
+ */
+
+#include <stdlib.h>
+
+/* FIXME: This should look for multiplication overflow */
+
+void *calloc(size_t nmemb, size_t size)
+{
+  void *ptr;
+
+  size *= nmemb;
+  ptr = malloc(size);
+  if ( ptr )
+    memset(ptr, 0, size);
+
+  return ptr;
+}
+
```

Klibc – 2002


index : klibc/klibc.git

klibc main development tree

about summary refs log tree commit diff stats

path: root/klibc/calloc.c

author	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
committer	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
commit	74b67d34871be80a0ed5ef636f5d3ec9d97c0b99 (patch)	
tree	91d66a855bca52ee84b4cc860d88061104767347 /klibc/calloc.c	
parent	1b20b39d14c1bf37f011453a23a8d8306036b096 (diff)	
download	klibc-74b67d34871be80a0ed5ef636f5d3ec9d97c0b99.tar.gz	

Add calloc() and realloc()

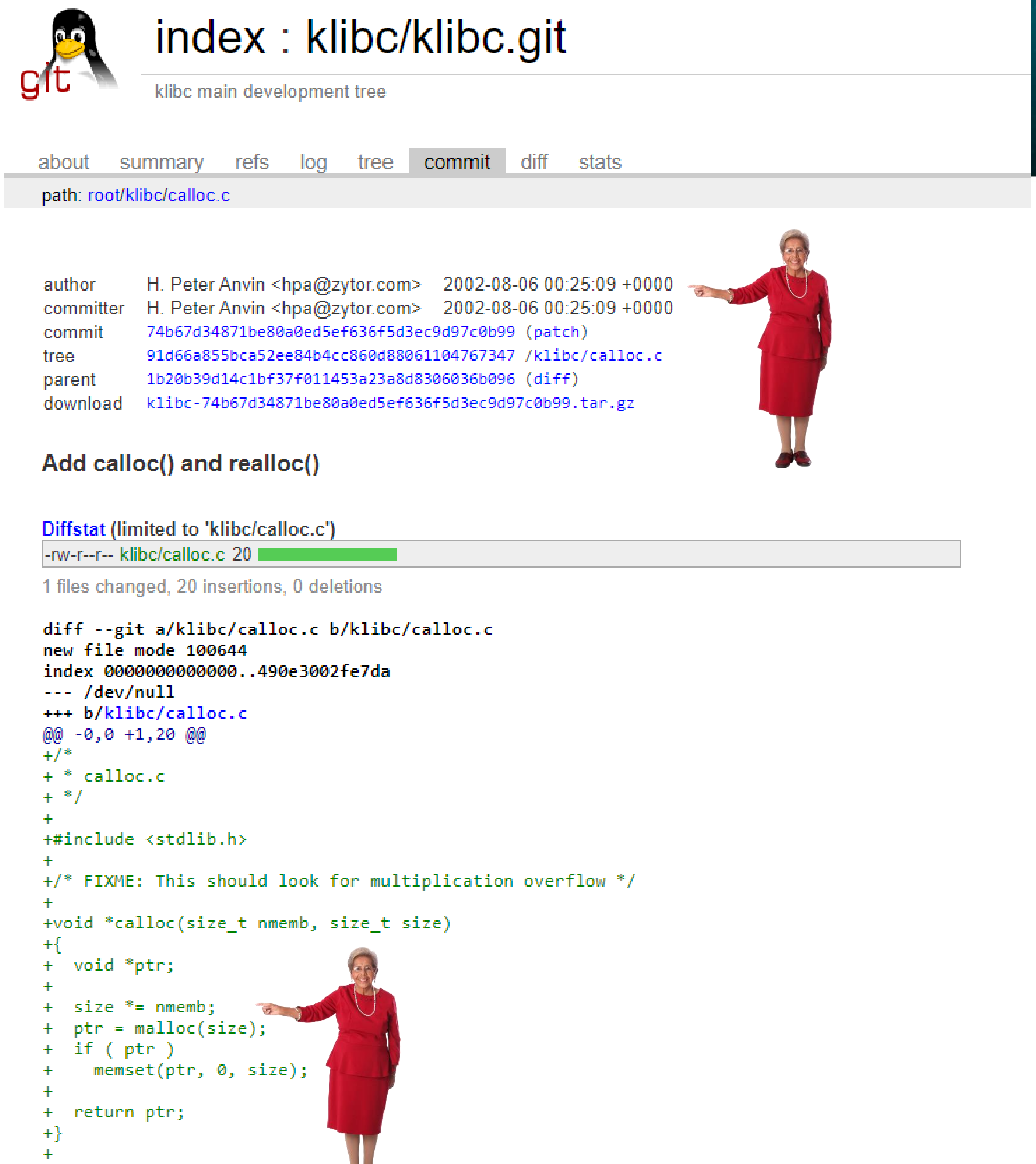
Diffstat (limited to 'klibc/calloc.c')

-rw-r--r-- klibc/calloc.c 20

1 files changed, 20 insertions, 0 deletions

```
diff --git a/klibc/calloc.c b/klibc/calloc.c
new file mode 100644
index 0000000000000..490e3002fe7da
--- /dev/null
+++ b/klibc/calloc.c
@@ -0,0 +1,20 @@
+/*
+ * calloc.c
+ */
+
+#include <stdlib.h>
+
+/* FIXME: This should look for multiplication overflow */
+
+void *calloc(size_t nmemb, size_t size)
+{
+  void *ptr;
+
+  size *= nmemb;
+  ptr = malloc(size);
+  if ( ptr )
+    memset(ptr, 0, size);
+
+  return ptr;
+}
+
```

Klibc – 2002



index : klibc/klibc.git

klibc main development tree

about summary refs log tree commit diff stats

path: root/klibc/calloc.c

author	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
committer	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
commit	74b67d34871be80a0ed5ef636f5d3ec9d97c0b99 (patch)	
tree	91d66a855bca52ee84b4cc860d88061104767347 /klibc/calloc.c	
parent	1b20b39d14c1bf37f011453a23a8d8306036b096 (diff)	
download	klibc-74b67d34871be80a0ed5ef636f5d3ec9d97c0b99.tar.gz	

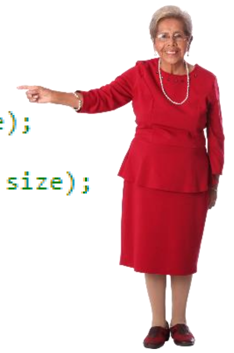
Add calloc() and realloc()

Diffstat (limited to 'klibc/calloc.c')

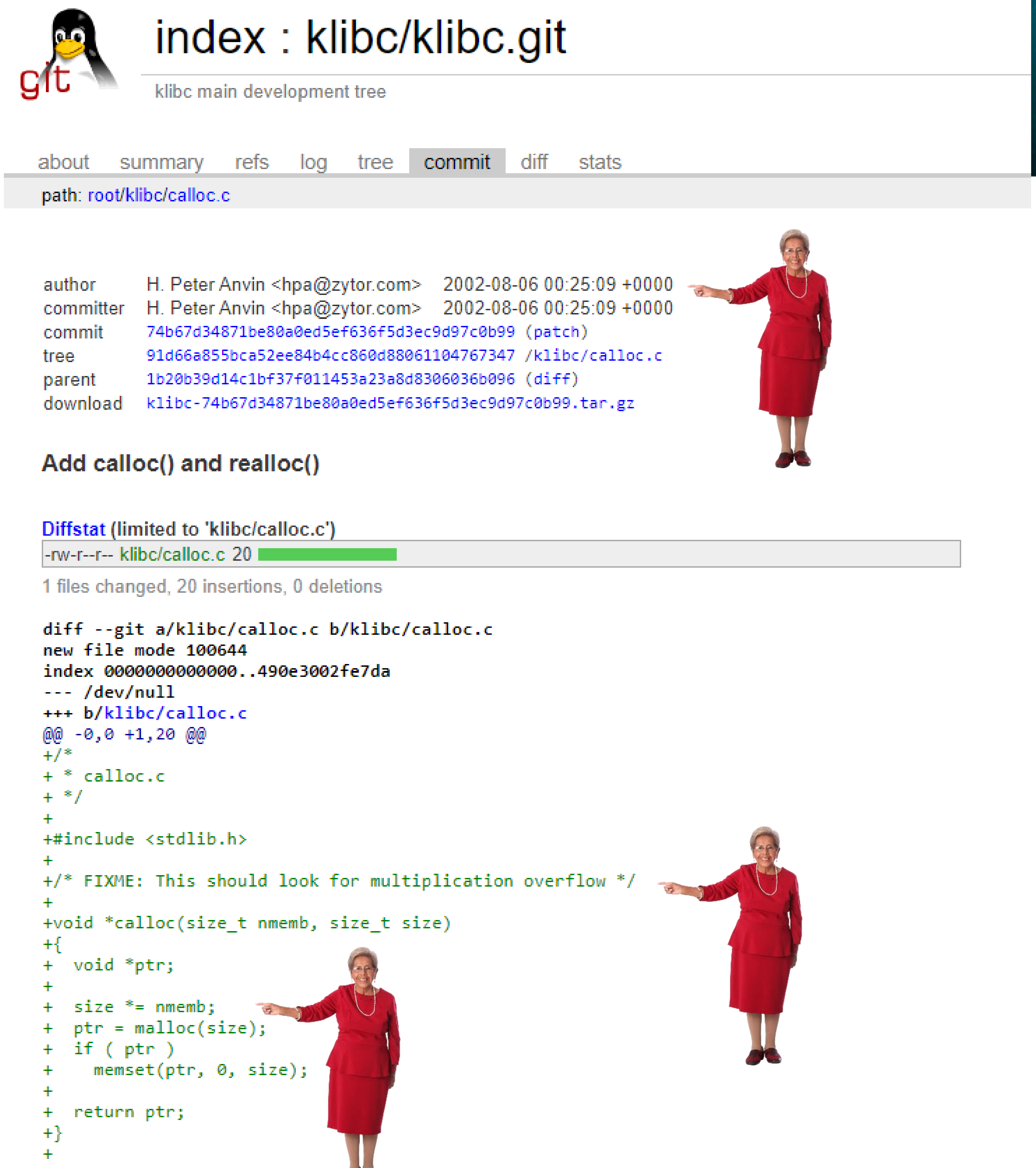
-rw-r--r-- klibc/calloc.c 20

1 files changed, 20 insertions, 0 deletions

```
diff --git a/klibc/calloc.c b/klibc/calloc.c
new file mode 100644
index 0000000000000..490e3002fe7da
--- /dev/null
+++ b/klibc/calloc.c
@@ -0,0 +1,20 @@
+/*
+ * calloc.c
+ */
+
+#include <stdlib.h>
+
+/* FIXME: This should look for multiplication overflow */
+
+void *calloc(size_t nmemb, size_t size)
+{
+  void *ptr;
+
+  size *= nmemb;
+  ptr = malloc(size);
+  if ( ptr )
+    memset(ptr, 0, size);
+
+  return ptr;
+}
+
```



Klibc – 2002



index : klibc/klibc.git

klibc main development tree

about summary refs log tree commit diff stats

path: root/klibc/calloc.c

author	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
committer	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
commit	74b67d34871be80a0ed5ef636f5d3ec9d97c0b99 (patch)	
tree	91d66a855bca52ee84b4cc860d88061104767347 /klibc/calloc.c	
parent	1b20b39d14c1bf37f011453a23a8d8306036b096 (diff)	
download	klibc-74b67d34871be80a0ed5ef636f5d3ec9d97c0b99.tar.gz	

Add calloc() and realloc()

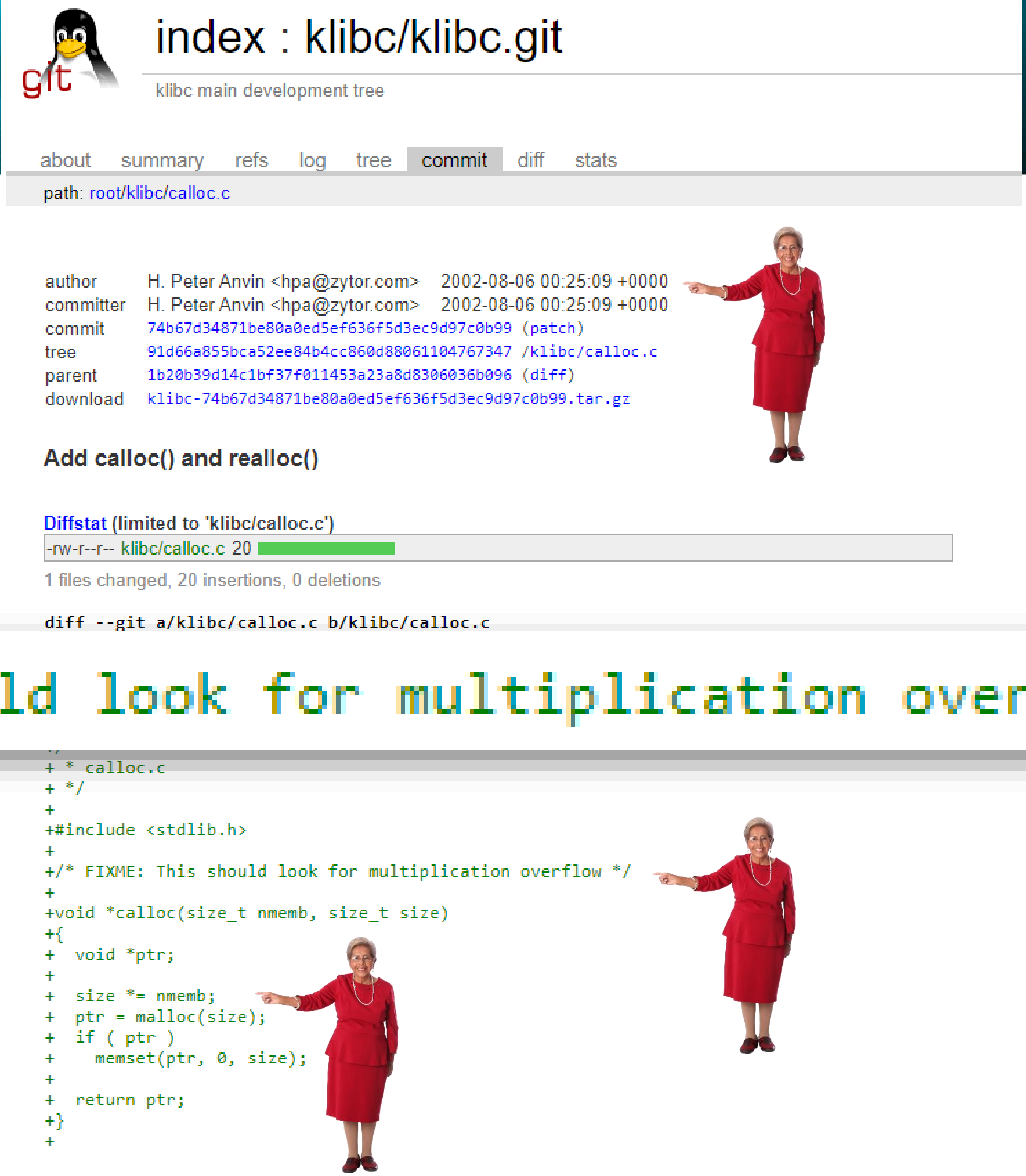
Diffstat (limited to 'klibc/calloc.c')

-rw-r--r-- klibc/calloc.c 20

1 files changed, 20 insertions, 0 deletions

```
diff --git a/klibc/calloc.c b/klibc/calloc.c
new file mode 100644
index 0000000000000..490e3002fe7da
--- /dev/null
+++ b/klibc/calloc.c
@@ -0,0 +1,20 @@
+/*
+ * calloc.c
+ */
+
+#include <stdlib.h>
+
+/* FIXME: This should look for multiplication overflow */
+
+void *calloc(size_t nmemb, size_t size)
+{
+  void *ptr;
+
+  size *= nmemb;
+  ptr = malloc(size);
+  if ( ptr )
+    memset(ptr, 0, size);
+
+  return ptr;
+}
+
```

Klibc – 2002



index : klibc/klibc.git

klibc main development tree

about summary refs log tree commit diff stats

path: root/klibc/calloc.c

author	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
committer	H. Peter Anvin <hpa@zytor.com>	2002-08-06 00:25:09 +0000
commit	74b67d34871be80a0ed5ef636f5d3ec9d97c0b99 (patch)	
tree	91d66a855bca52ee84b4cc860d88061104767347 /klibc/calloc.c	
parent	1b20b39d14c1bf37f011453a23a8d8306036b096 (diff)	
download	klibc-74b67d34871be80a0ed5ef636f5d3ec9d97c0b99.tar.gz	

Add calloc() and realloc()

Diffstat (limited to 'klibc/calloc.c')

-rw-r--r-- klibc/calloc.c 20

1 files changed, 20 insertions, 0 deletions

```
diff --git a/klibc/calloc.c b/klibc/calloc.c
```

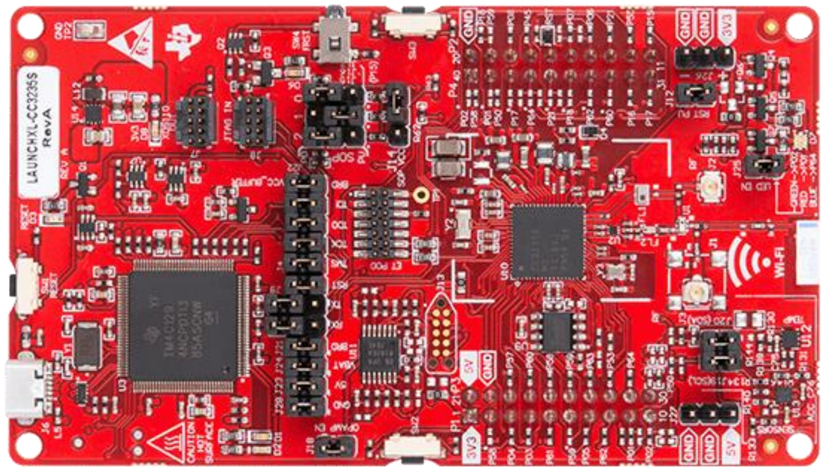
```
+/* FIXME: This should look for multiplication overflow */
```

```
+ * calloc.c
+ */
+
+#include <stdlib.h>
+
+/* FIXME: This should look for multiplication overflow */
+
+void *calloc(size_t nmemb, size_t size)
+{
+  void *ptr;
+
+  size *= nmemb;
+  ptr = malloc(size);
+  if ( ptr )
+    memset(ptr, 0, size);
+
+  return ptr;
+}
+
```

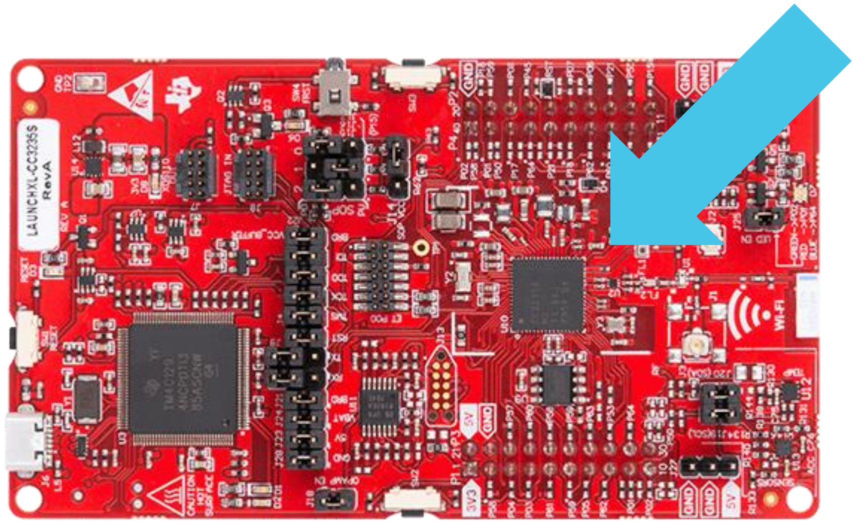
Technical Analysis

Texas Instruments “SimpleLink” SDK

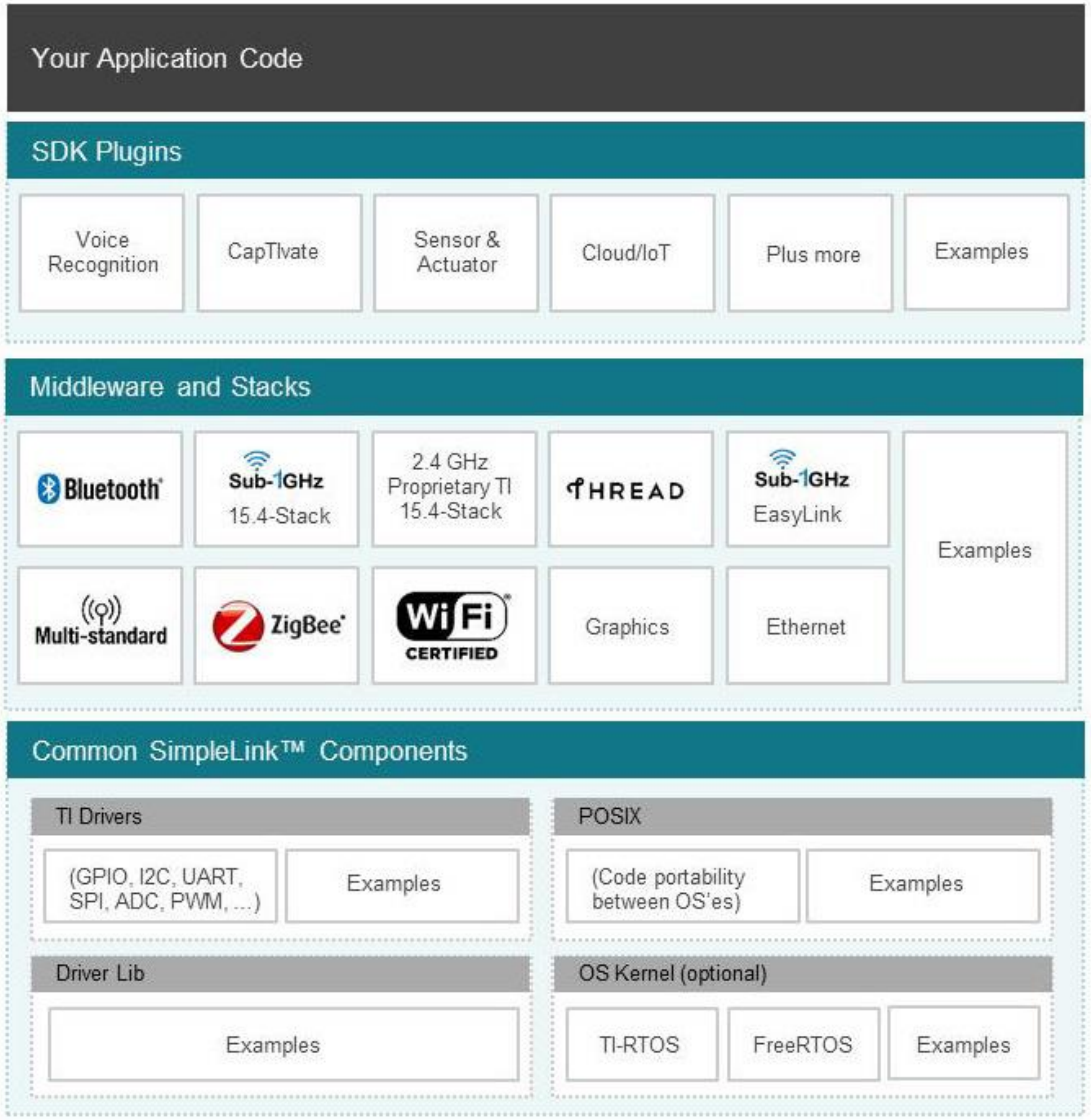
Texas Instruments “SimpleLink” SDK



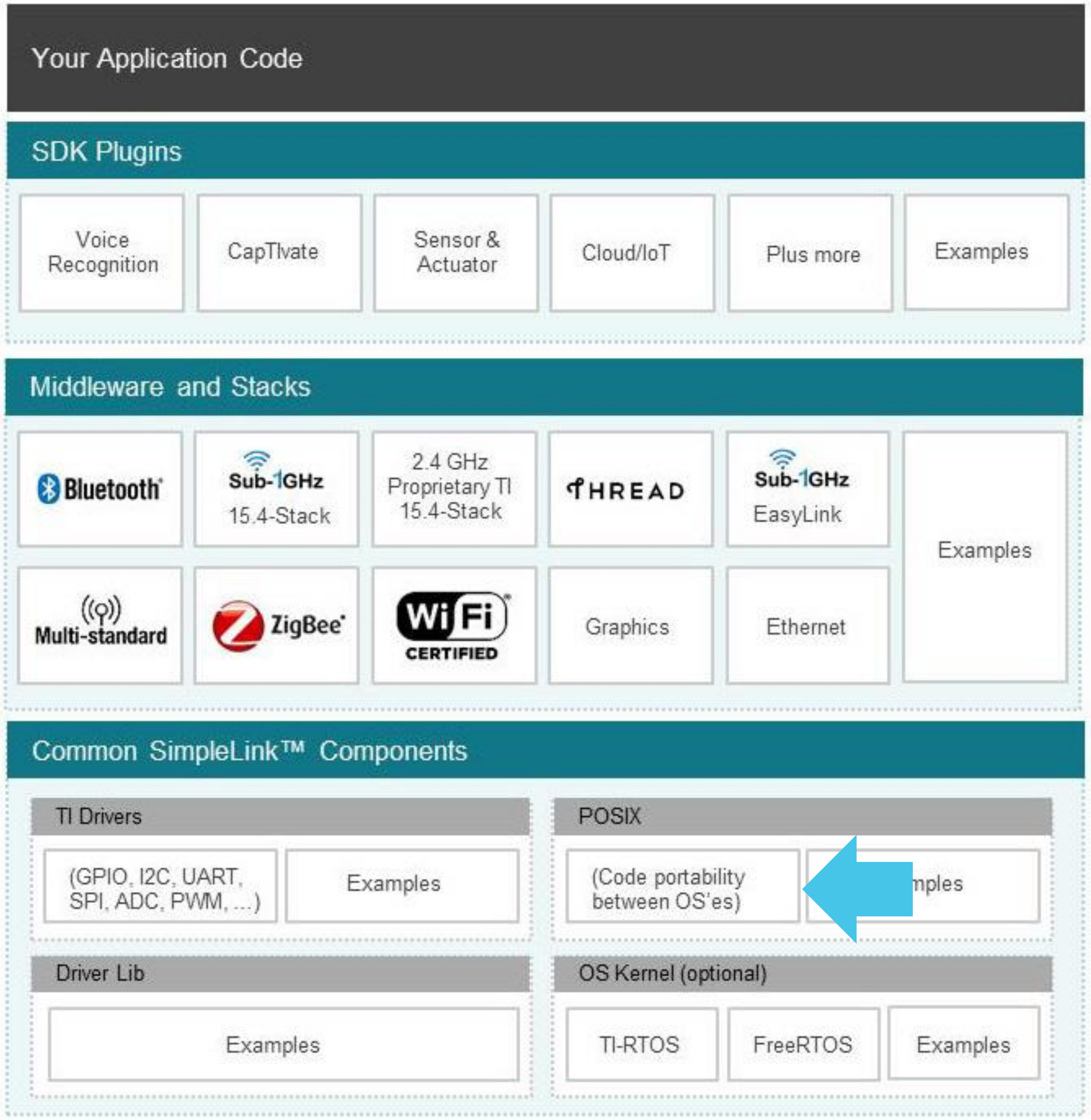
Texas Instruments "SimpleLink" SDK



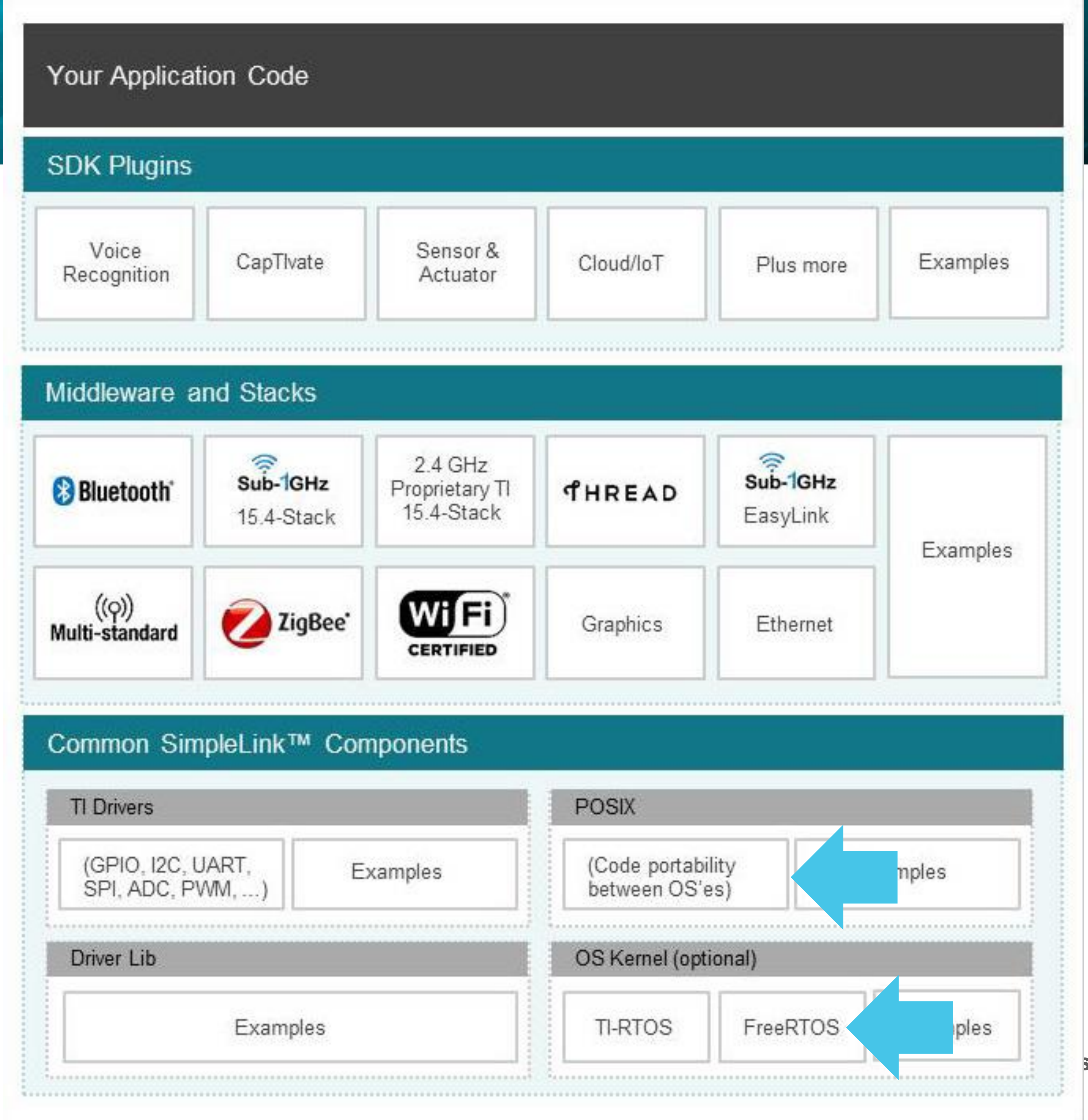
Texas Instruments "SimpleLink" SDK



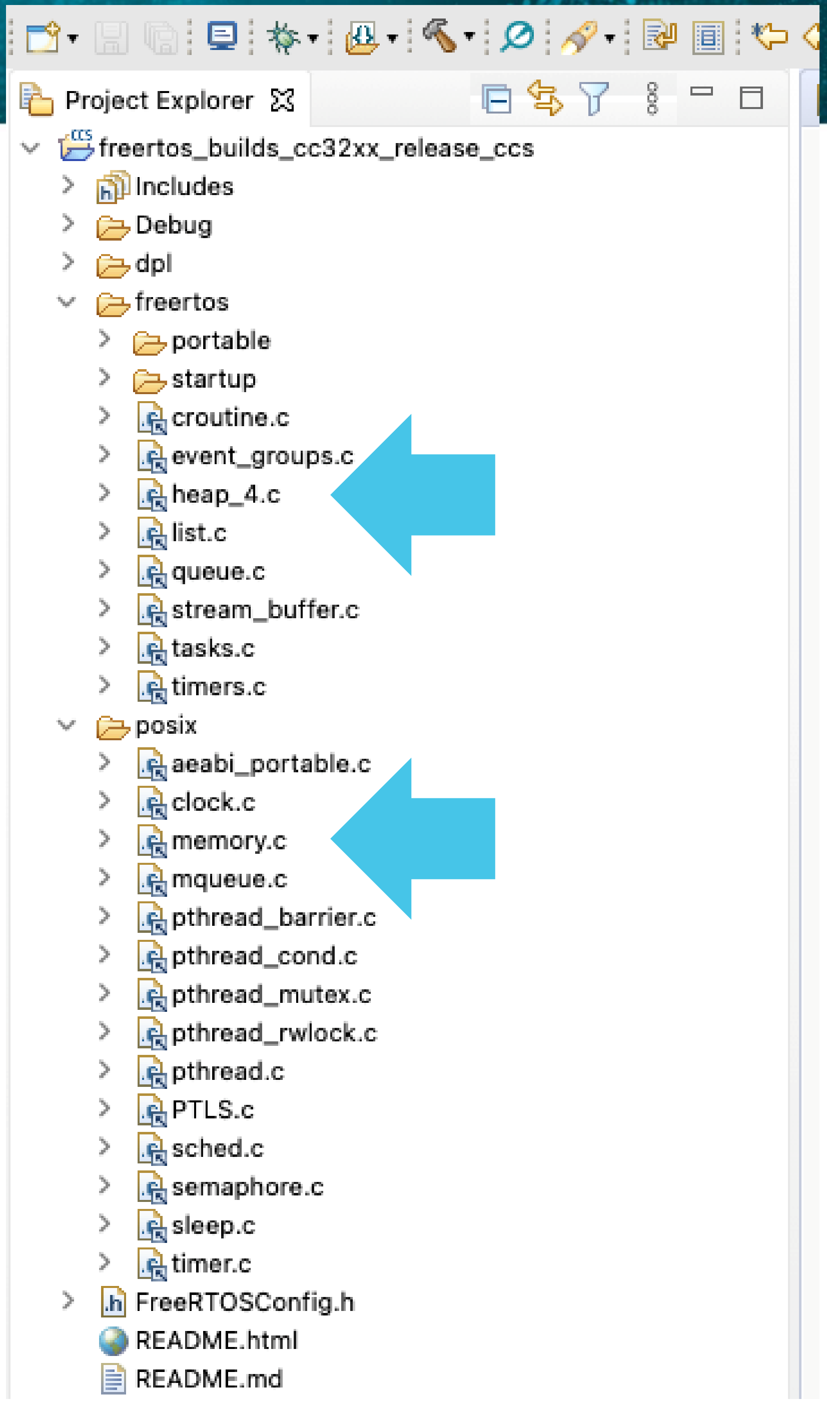
Texas Instruments "SimpleLink" SDK



Texas Instruments “SimpleLink” SDK

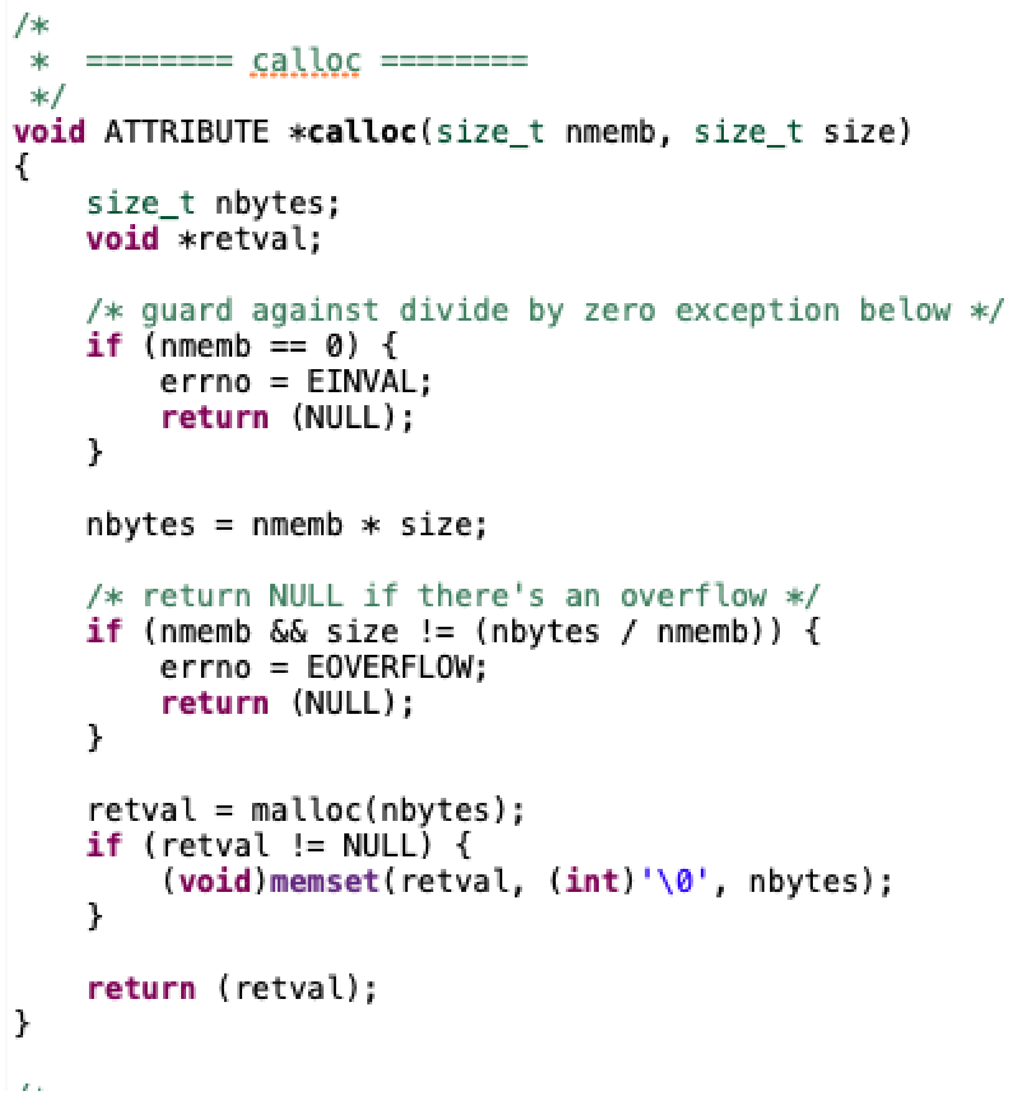


Texas Instruments "SimpleLink" SDK



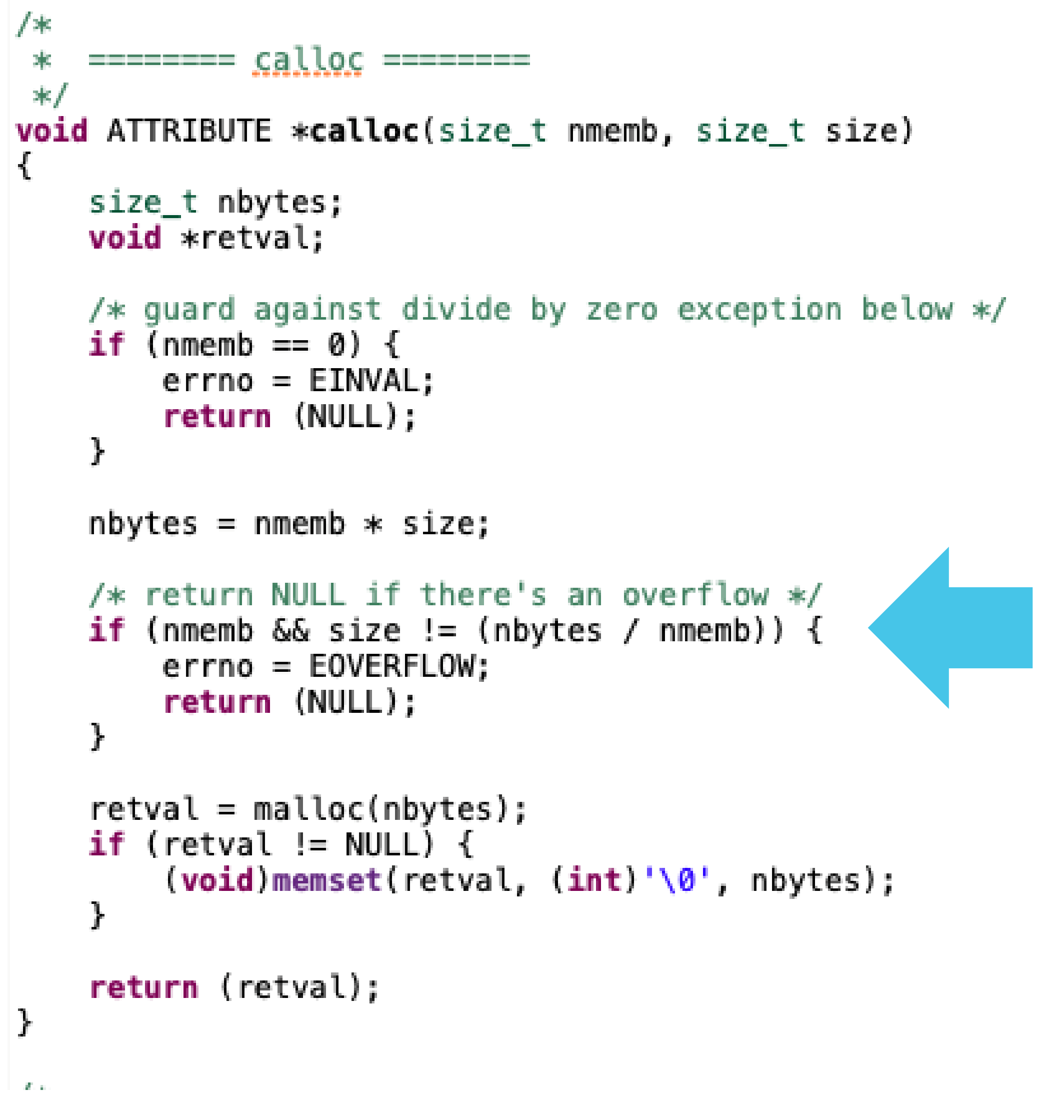
```
freertos_builds_cc32xx_release_ccs
  Includes
  Debug
  dpl
  freertos
    portable
    startup
    croutine.c
    event_groups.c
    heap_4.c
    list.c
    queue.c
    stream_buffer.c
    tasks.c
    timers.c
  posix
    aeabi_portable.c
    clock.c
    memory.c
    mqueue.c
    pthread_barrier.c
    pthread_cond.c
    pthread_mutex.c
    pthread_rwlock.c
    pthread.c
    PTLS.c
    sched.c
    semaphore.c
    sleep.c
    timer.c
  FreeRTOSConfig.h
  README.html
  README.md
```

Calloc is safe



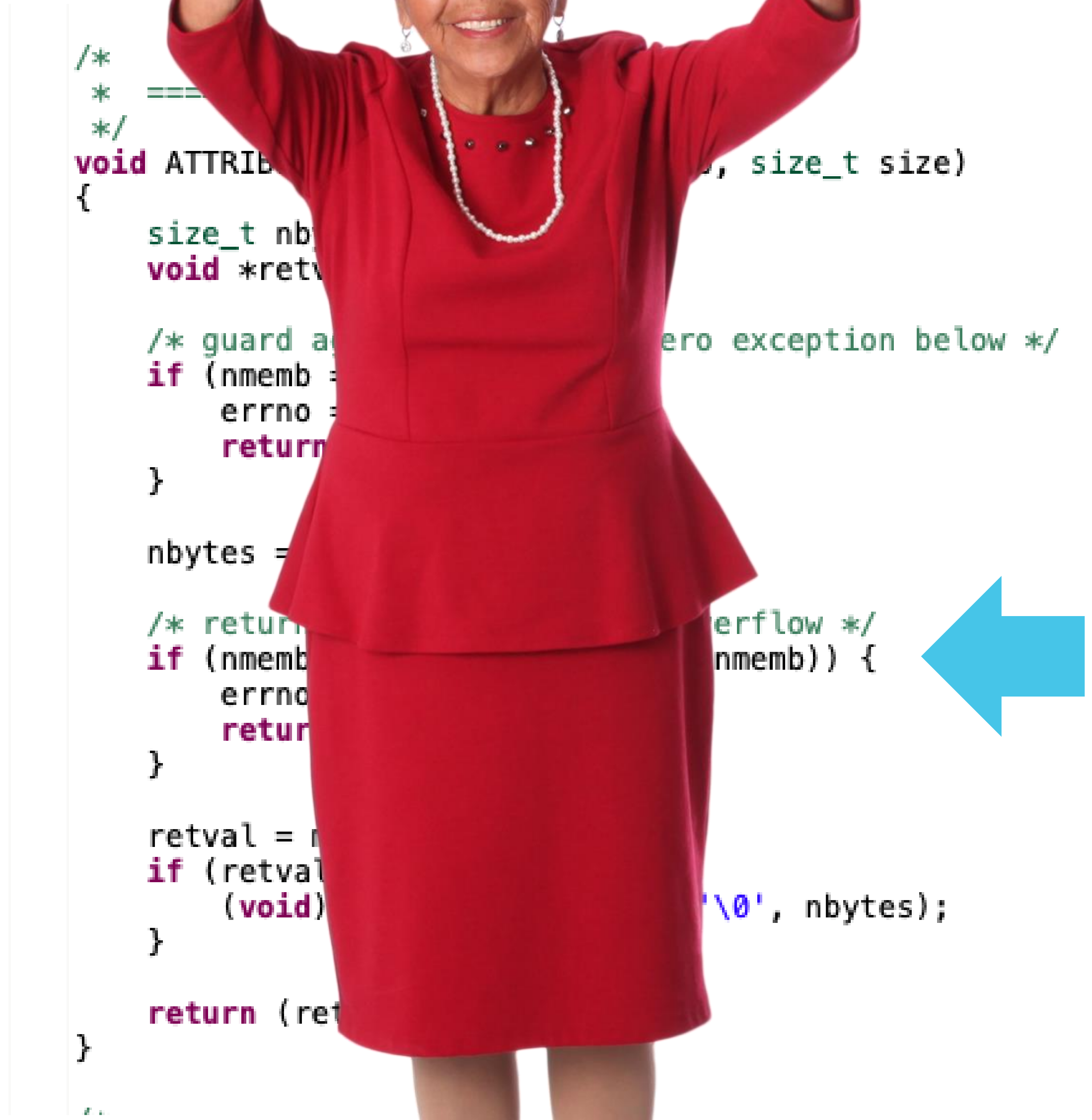
```
/*
 *  ======== calloc ========
 */
void ATTRIBUTE *calloc(size_t nmemb, size_t size)
{
    size_t nbytes;
    void *retval;

    /* guard against divide by zero exception below */
    if (nmemb == 0) {
        errno = EINVAL;
        return (NULL);
    }

    nbytes = nmemb * size;

    /* return NULL if there's an overflow */
    if (nmemb && size != (nbytes / nmemb)) {
        errno = EOVERFLOW;
        return (NULL);
    }

    retval = malloc(nbytes);
    if (retval != NULL) {
        (void)memset(retval, (int)'\0', nbytes);
    }

    return (retval);
}
```

Calloc is safe



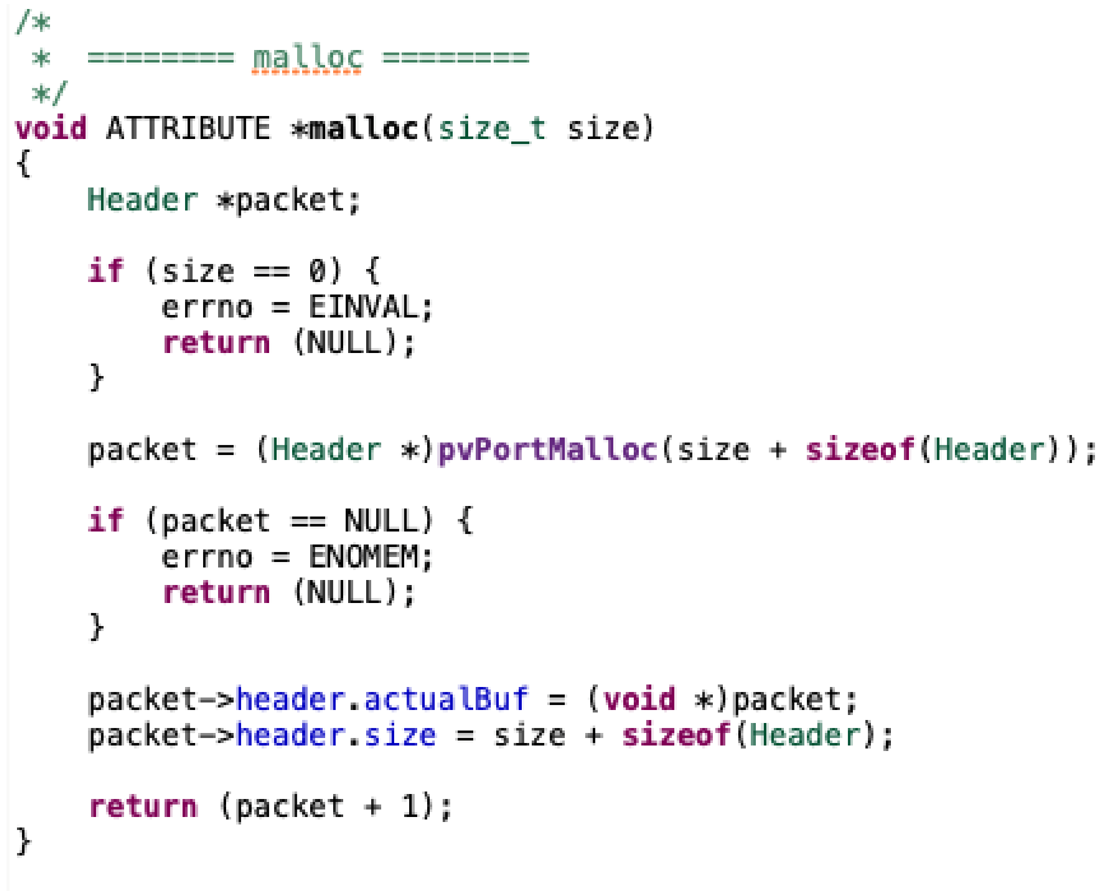
```
/*
 *   ======== calloc ========
 */
void ATTRIBUTE *calloc(size_t nmemb, size_t size)
{
    size_t nbytes;
    void *retval;

    /* guard against divide by zero exception below */
    if (nmemb == 0) {
        errno = EINVAL;
        return (NULL);
    }

    nbytes = nmemb * size;

    /* return NULL if there's an overflow */
    if (nmemb && size != (nbytes / nmemb)) {
        errno = EOVERFLOW;
        return (NULL);
    }

    retval = malloc(nbytes);
    if (retval != NULL) {
        (void)memset(retval, (int)'\0', nbytes);
    }

    return (retval);
}
```

Calloc is safe



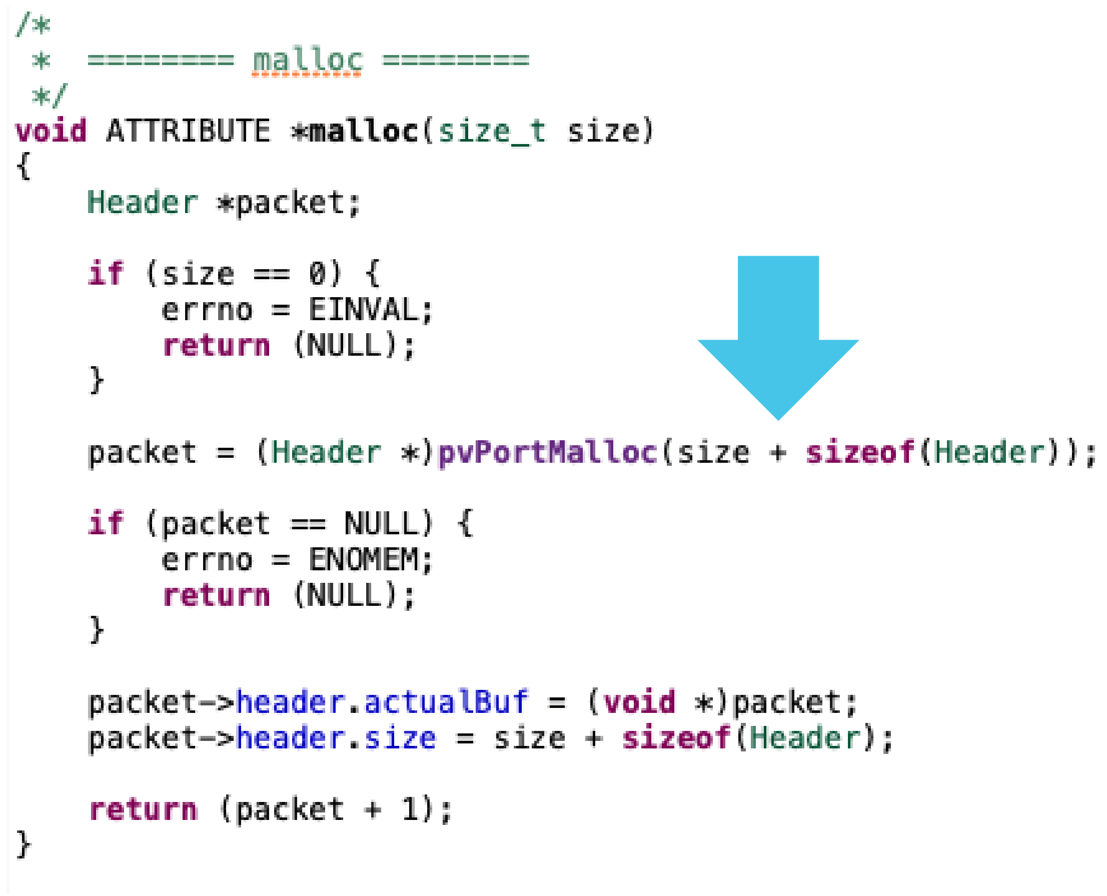
```
/*
 *  ===
 */
void ATTRIB                        , size_t size)
{
    size_t nb
    void *ret

    /* guard a                      ero exception below */
    if (nmemb =
        errno =
        return
    }

    nbytes =

    /* retur                       erflow */
    if (nmemb                      nmemb)) {
        errno
        retur
    }

    retval = r
    if (retval
        (void)                     '\0', nbytes);
    }

    return (ret
}
```

Malloc isn’t



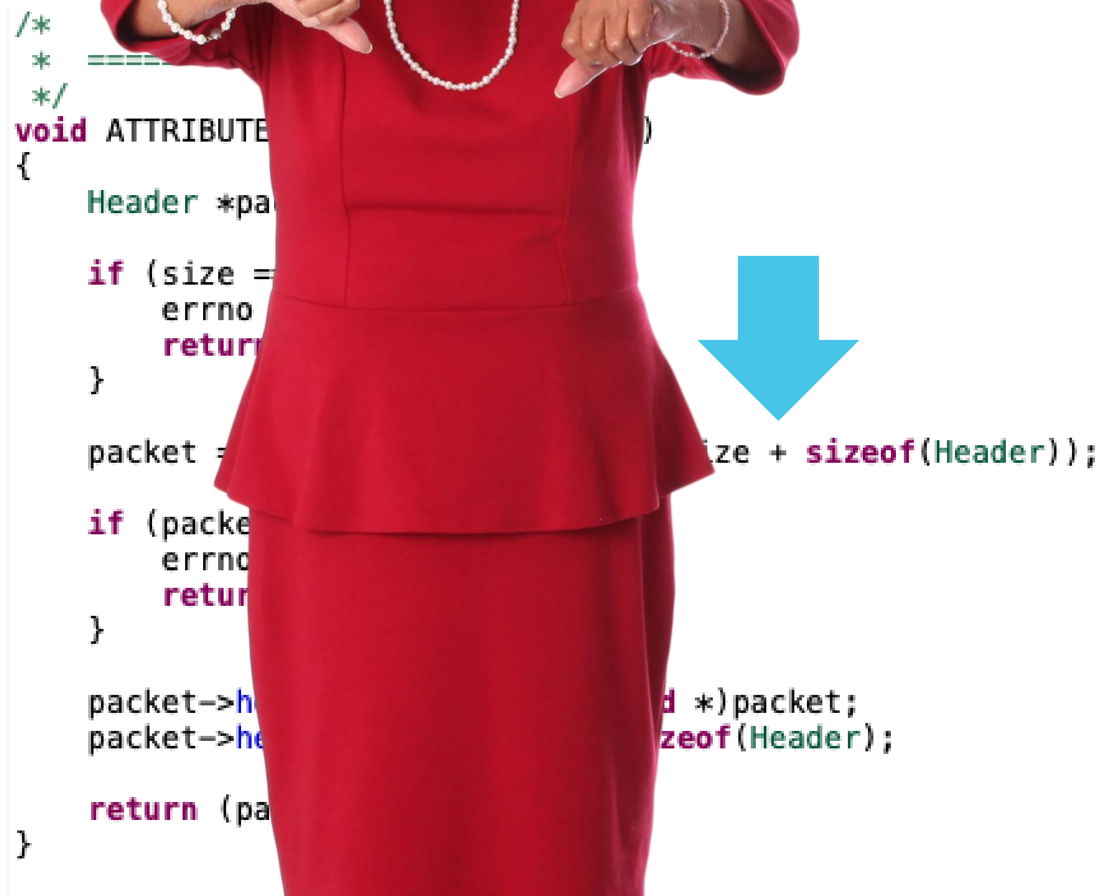
```
/*
 *  ======== malloc ========
 */
void ATTRIBUTE *malloc(size_t size)
{
    Header *packet;

    if (size == 0) {
        errno = EINVAL;
        return (NULL);
    }

    packet = (Header *)pvPortMalloc(size + sizeof(Header));

    if (packet == NULL) {
        errno = ENOMEM;
        return (NULL);
    }

    packet->header.actualBuf = (void *)packet;
    packet->header.size = size + sizeof(Header);

    return (packet + 1);
}
```

Malloc isn't



```c
/*
 *  ======== malloc ========
 */
void ATTRIBUTE *malloc(size_t size)
{
    Header *packet;

    if (size == 0) {
        errno = EINVAL;
        return (NULL);
    }

    packet = (Header *)pvPortMalloc(size + sizeof(Header));

    if (packet == NULL) {
        errno = ENOMEM;
        return (NULL);
    }

    packet->header.actualBuf = (void *)packet;
    packet->header.size = size + sizeof(Header);

    return (packet + 1);
}
```

Malloc isn’t



```
/*
 *  ====
 */
void ATTRIBUTE                              )
{
    Header *pa

    if (size =
        errno
        retur
    }

    packet =                               ize + sizeof(Header));

    if (packe
        errno
        retur
    }

    packet->h                            d *)packet;
    packet->he                           zeof(Header);

    return (pa
}
```

Exploitation

SimpleLink POC

Over-The-Air(OTA) Updates

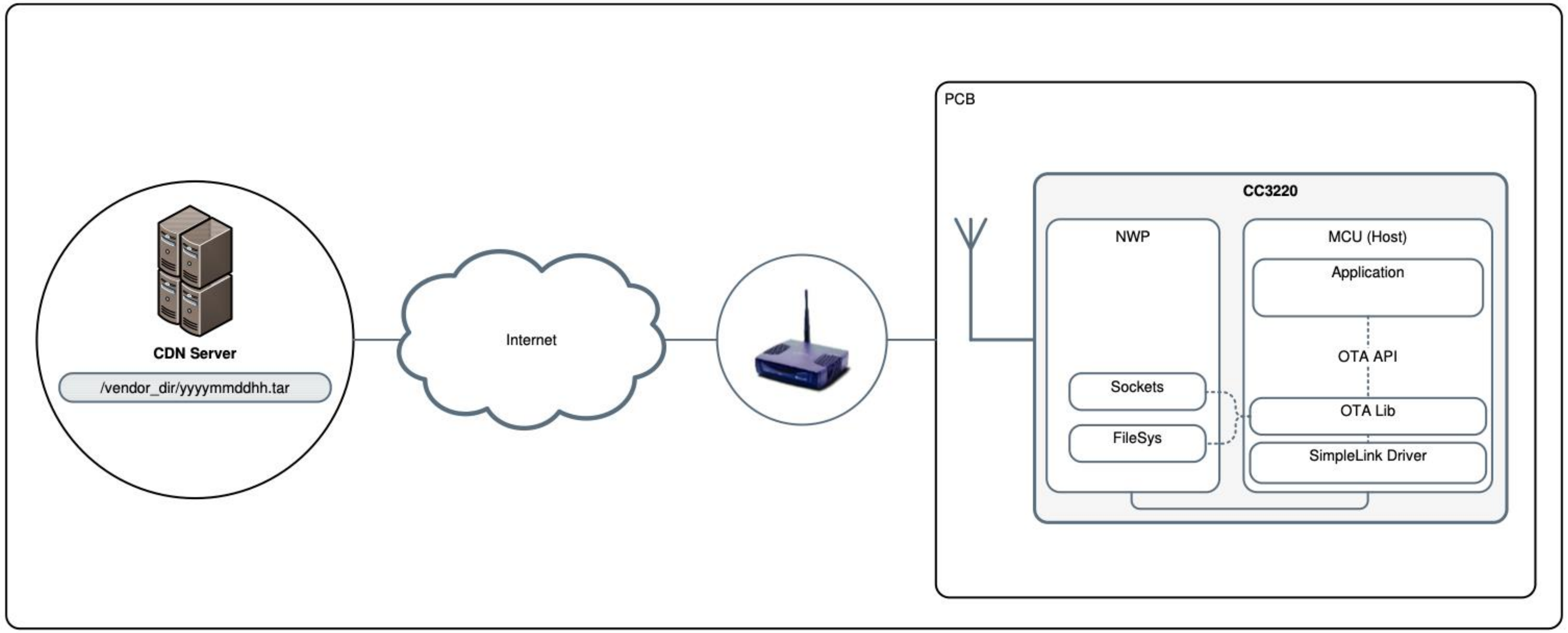
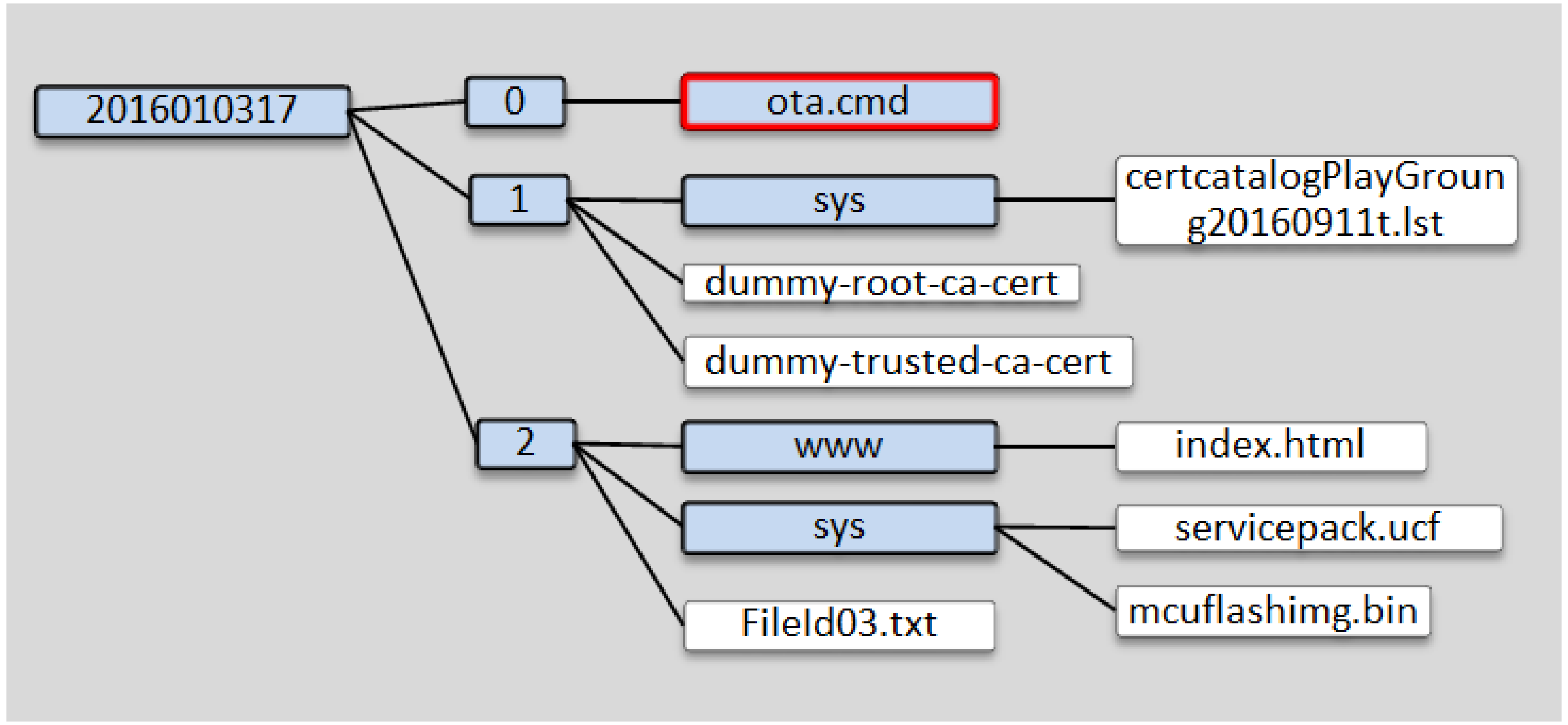
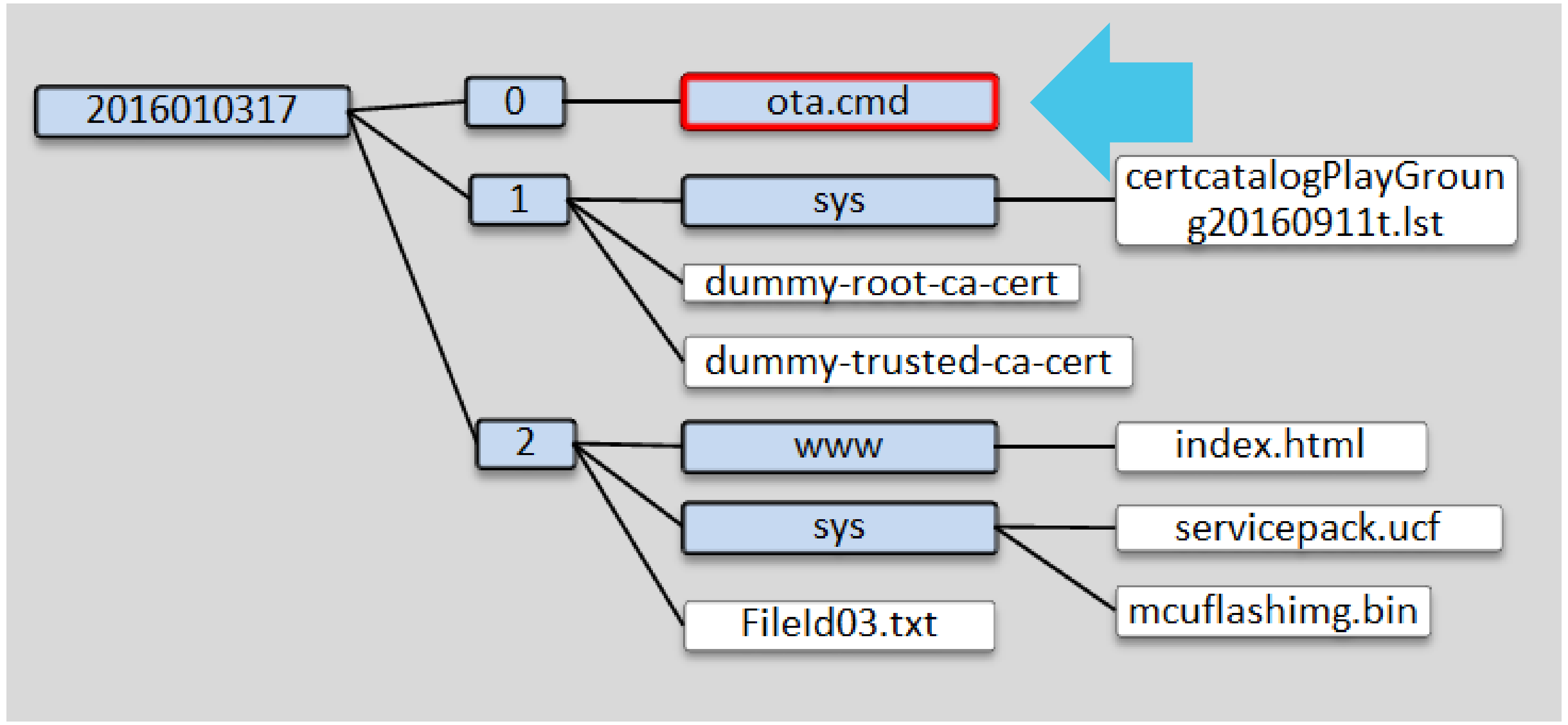


Figure 1-1. OTA System Diagram

Metadata File



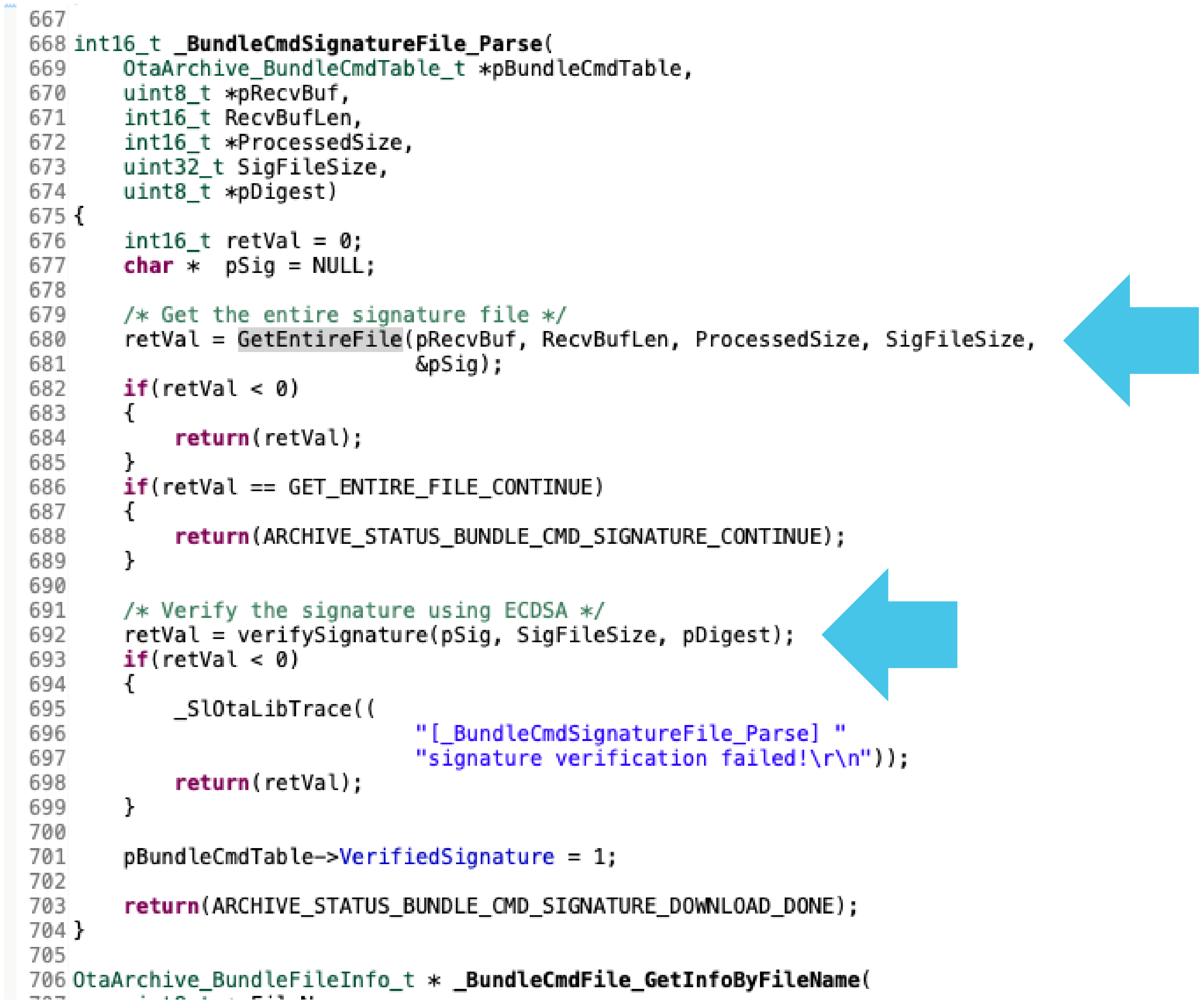
Metadata File



Metadata File

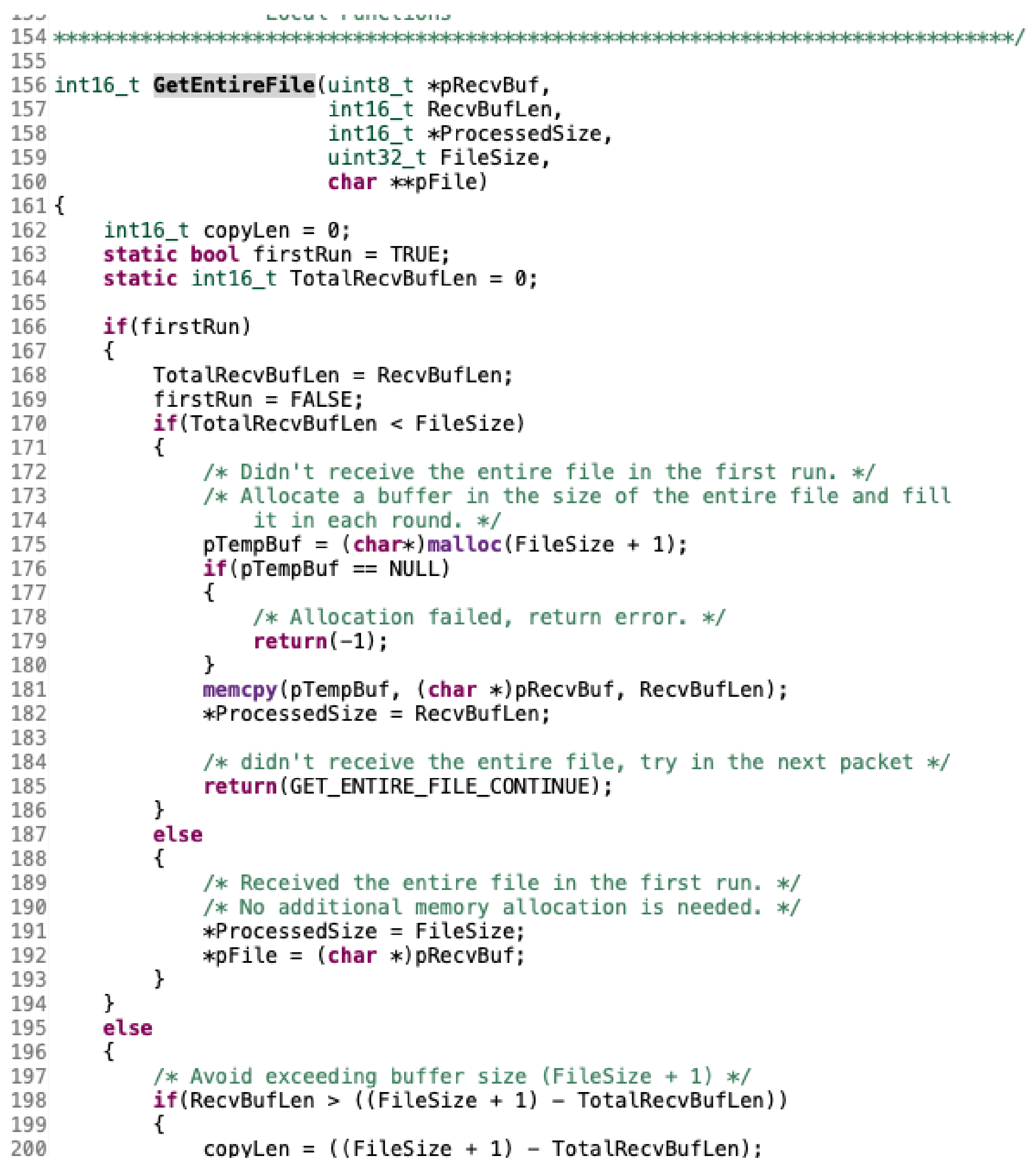
```
[
    {
        "filename": "/local/FileId03.txt",
        "signature_base64": "kc8XfFOfMfr4HBJiPxTRHyb99d2uOoICme0AYU94+...",
        "certificate": "dummy-trusted-ca-certcert",
        "secured": 1,
        "bundle": 0
    },
    {
        "filename": "/sys/servicepack.ucf"
        "signature_base64": "EEC6GZG1Oq6Agigmb2f9ny9rNK2Mg9hFClpgMhd4jCW/...",
        "certificate":"",
        "secured": 1,
        "bundle": 1
    },
    {
        "filename": "/sys/mcuflashimg.bin",
        "signature_base64": "dRTARlzLFKAog34ZUareCmo9j2lrHnvc+v3qqW9C/...",
        "certificate": "dummy-root-ca-certcert",
        "secured": 1,
        "bundle": 1
    }
]
```

Signature Verification



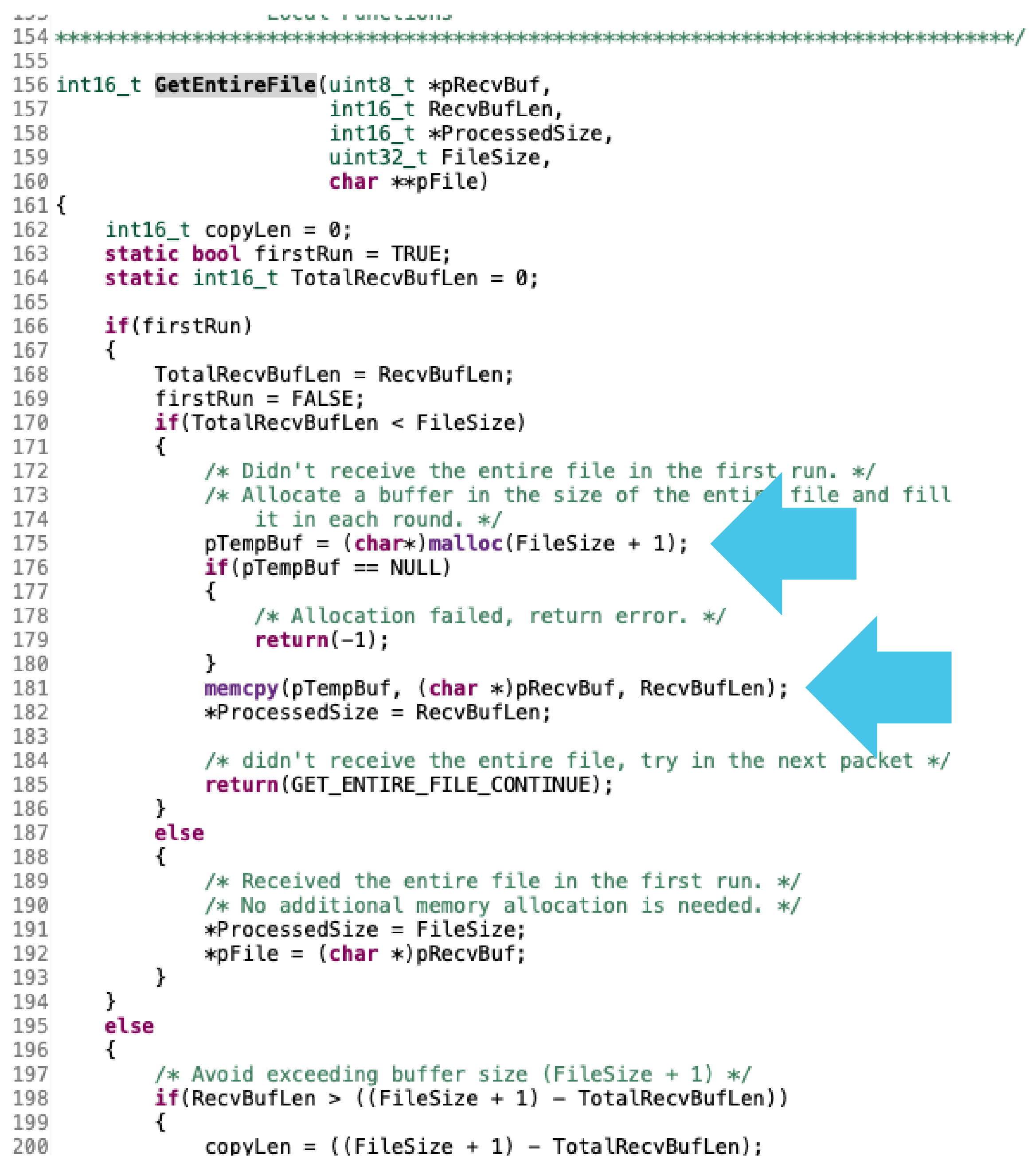
```c
int16_t _BundleCmdSignatureFile_Parse(
    OtaArchive_BundleCmdTable_t *pBundleCmdTable,
    uint8_t *pRecvBuf,
    int16_t RecvBufLen,
    int16_t *ProcessedSize,
    uint32_t SigFileSize,
    uint8_t *pDigest)
{
    int16_t retVal = 0;
    char *  pSig = NULL;

    /* Get the entire signature file */
    retVal = GetEntireFile(pRecvBuf, RecvBufLen, ProcessedSize, SigFileSize,
                           &pSig);
    if(retVal < 0)
    {
        return(retVal);
    }
    if(retVal == GET_ENTIRE_FILE_CONTINUE)
    {
        return(ARCHIVE_STATUS_BUNDLE_CMD_SIGNATURE_CONTINUE);
    }

    /* Verify the signature using ECDSA */
    retVal = verifySignature(pSig, SigFileSize, pDigest);
    if(retVal < 0)
    {
        _SlOtaLibTrace((
                       "[_BundleCmdSignatureFile_Parse] "
                       "signature verification failed!\r\n"));
        return(retVal);
    }

    pBundleCmdTable->VerifiedSignature = 1;

    return(ARCHIVE_STATUS_BUNDLE_CMD_SIGNATURE_DOWNLOAD_DONE);
}

OtaArchive_BundleFileInfo_t * _BundleCmdFile_GetInfoByFileName(
```

GetEntireFile



```
                 Local Functions
*****************************************************************************/

int16_t GetEntireFile(uint8_t *pRecvBuf,
                      int16_t RecvBufLen,
                      int16_t *ProcessedSize,
                      uint32_t FileSize,
                      char **pFile)
{
    int16_t copyLen = 0;
    static bool firstRun = TRUE;
    static int16_t TotalRecvBufLen = 0;

    if(firstRun)
    {
        TotalRecvBufLen = RecvBufLen;
        firstRun = FALSE;
        if(TotalRecvBufLen < FileSize)
        {
            /* Didn't receive the entire file in the first run. */
            /* Allocate a buffer in the size of the entire file and fill
                it in each round. */
            pTempBuf = (char*)malloc(FileSize + 1);
            if(pTempBuf == NULL)
            {
                /* Allocation failed, return error. */
                return(-1);
            }
            memcpy(pTempBuf, (char *)pRecvBuf, RecvBufLen);
            *ProcessedSize = RecvBufLen;

            /* didn't receive the entire file, try in the next packet */
            return(GET_ENTIRE_FILE_CONTINUE);
        }
        else
        {
            /* Received the entire file in the first run. */
            /* No additional memory allocation is needed. */
            *ProcessedSize = FileSize;
            *pFile = (char *)pRecvBuf;
        }
    }
    else
    {
        /* Avoid exceeding buffer size (FileSize + 1) */
        if(RecvBufLen > ((FileSize + 1) - TotalRecvBufLen))
        {
            copyLen = ((FileSize + 1) - TotalRecvBufLen);
```

GetEntireFile

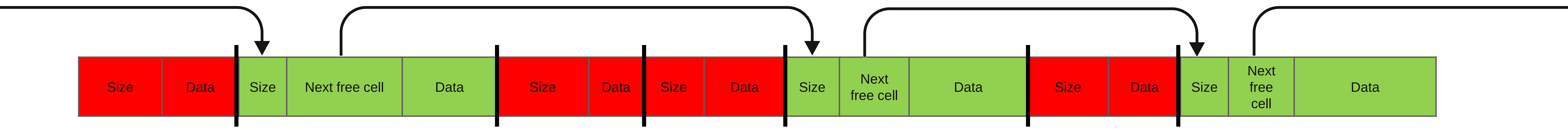


```
                 Local Functions
*****************************************************************************/

int16_t GetEntireFile(uint8_t *pRecvBuf,
                      int16_t RecvBufLen,
                      int16_t *ProcessedSize,
                      uint32_t FileSize,
                      char **pFile)
{
    int16_t copyLen = 0;
    static bool firstRun = TRUE;
    static int16_t TotalRecvBufLen = 0;

    if(firstRun)
    {
        TotalRecvBufLen = RecvBufLen;
        firstRun = FALSE;
        if(TotalRecvBufLen < FileSize)
        {
            /* Didn't receive the entire file in the first run. */
            /* Allocate a buffer in the size of the entire file and fill
                it in each round. */
            pTempBuf = (char*)malloc(FileSize + 1);
            if(pTempBuf == NULL)
            {
                /* Allocation failed, return error. */
                return(-1);
            }
            memcpy(pTempBuf, (char *)pRecvBuf, RecvBufLen);
            *ProcessedSize = RecvBufLen;

            /* didn't receive the entire file, try in the next packet */
            return(GET_ENTIRE_FILE_CONTINUE);
        }
        else
        {
            /* Received the entire file in the first run. */
            /* No additional memory allocation is needed. */
            *ProcessedSize = FileSize;
            *pFile = (char *)pRecvBuf;
        }
    }
    else
    {
        /* Avoid exceeding buffer size (FileSize + 1) */
        if(RecvBufLen > ((FileSize + 1) - TotalRecvBufLen))
        {
            copyLen = ((FileSize + 1) - TotalRecvBufLen);
```

Heap overflow to CODE EXECUTION

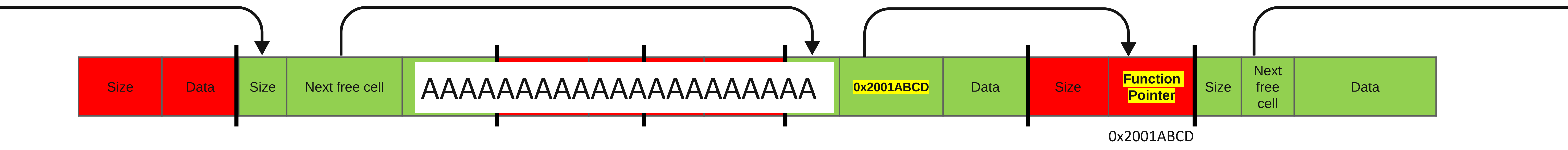
- **Heap Overflow**
- Find function pointer which we can override in memory.
- Override “next free” pointer of next block to desired address.
- Force another allocation with user-controlled data.
- Force call to overridden function pointer.

Heap overflow to CODE EXECUTION



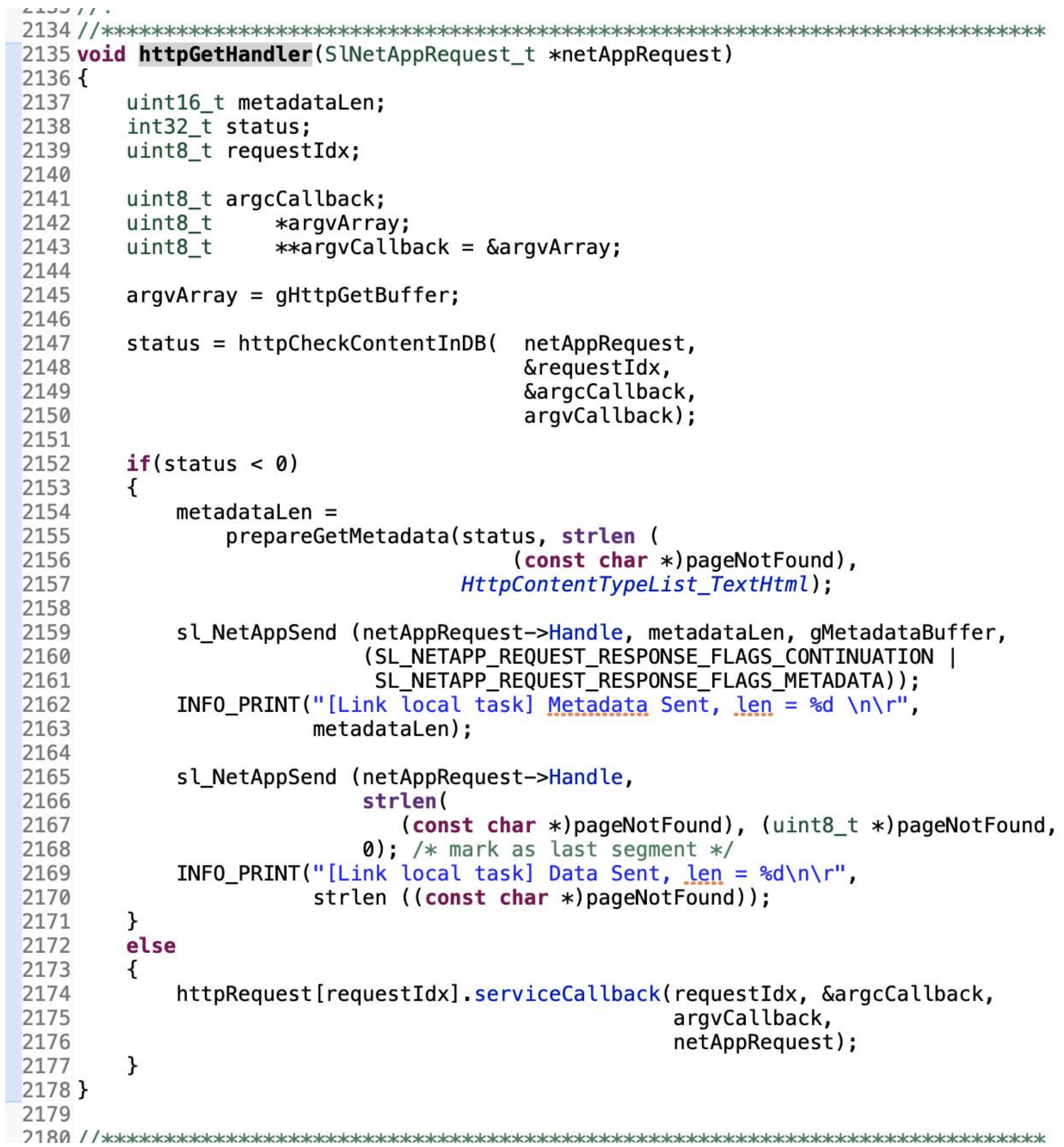
🟥 Allocated 🟩 Free

Heap overflow to CODE EXECUTION



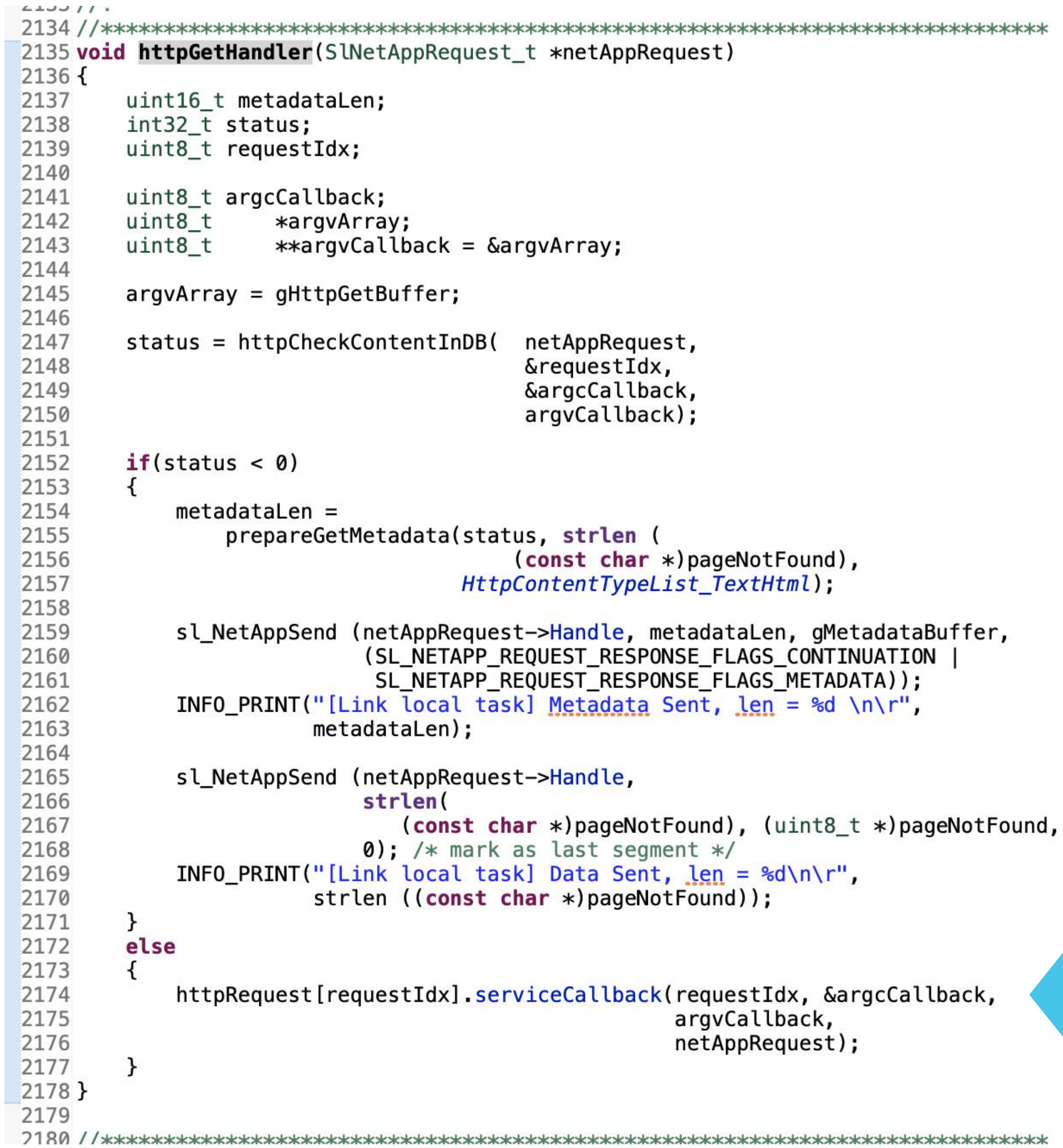
🟥 Allocated 🟩 Free

httpRequest



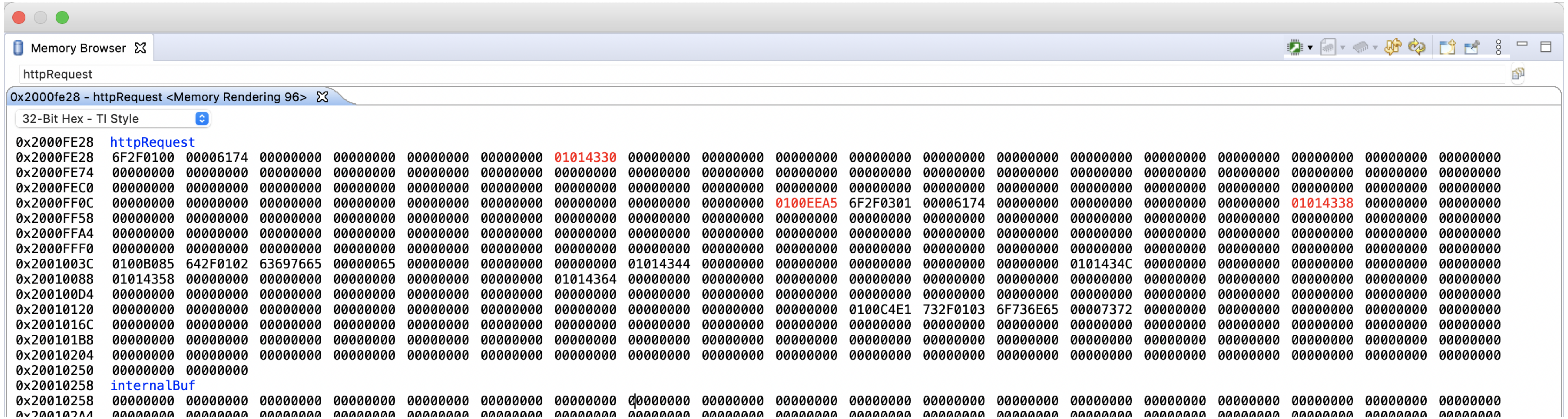
```
//***************************************************************************
void httpGetHandler(SlNetAppRequest_t *netAppRequest)
{
    uint16_t metadataLen;
    int32_t status;
    uint8_t requestIdx;

    uint8_t argcCallback;
    uint8_t     *argvArray;
    uint8_t     **argvCallback = &argvArray;

    argvArray = gHttpGetBuffer;

    status = httpCheckContentInDB(  netAppRequest,
                                    &requestIdx,
                                    &argcCallback,
                                    argvCallback);

    if(status < 0)
    {
        metadataLen =
            prepareGetMetadata(status, strlen (
                                   (const char *)pageNotFound),
                               HttpContentTypeList_TextHtml);

        sl_NetAppSend (netAppRequest->Handle, metadataLen, gMetadataBuffer,
                       (SL_NETAPP_REQUEST_RESPONSE_FLAGS_CONTINUATION |
                        SL_NETAPP_REQUEST_RESPONSE_FLAGS_METADATA));
        INFO_PRINT("[Link local task] Metadata Sent, len = %d \n\r",
                   metadataLen);

        sl_NetAppSend (netAppRequest->Handle,
                       strlen(
                           (const char *)pageNotFound), (uint8_t *)pageNotFound,
                       0); /* mark as last segment */
        INFO_PRINT("[Link local task] Data Sent, len = %d\n\r",
                   strlen ((const char *)pageNotFound));
    }
    else
    {
        httpRequest[requestIdx].serviceCallback(requestIdx, &argcCallback,
                                                argvCallback,
                                                netAppRequest);
    }
}

//***************************************************************************
```

httpRequest



```
//*****************************************************************************
void httpGetHandler(SlNetAppRequest_t *netAppRequest)
{
    uint16_t metadataLen;
    int32_t status;
    uint8_t requestIdx;

    uint8_t argcCallback;
    uint8_t      *argvArray;
    uint8_t      **argvCallback = &argvArray;

    argvArray = gHttpGetBuffer;

    status = httpCheckContentInDB(  netAppRequest,
                                    &requestIdx,
                                    &argcCallback,
                                    argvCallback);

    if(status < 0)
    {
        metadataLen =
            prepareGetMetadata(status, strlen (
                                   (const char *)pageNotFound),
                               HttpContentTypeList_TextHtml);

        sl_NetAppSend (netAppRequest->Handle, metadataLen, gMetadataBuffer,
                       (SL_NETAPP_REQUEST_RESPONSE_FLAGS_CONTINUATION |
                        SL_NETAPP_REQUEST_RESPONSE_FLAGS_METADATA));
        INFO_PRINT("[Link local task] Metadata Sent, len = %d \n\r",
                   metadataLen);

        sl_NetAppSend (netAppRequest->Handle,
                       strlen(
                          (const char *)pageNotFound), (uint8_t *)pageNotFound,
                       0); /* mark as last segment */
        INFO_PRINT("[Link local task] Data Sent, len = %d\n\r",
                   strlen ((const char *)pageNotFound));
    }
    else
    {
        httpRequest[requestIdx].serviceCallback(requestIdx, &argcCallback,
                                                argvCallback,
                                                netAppRequest);
    }
}

//*****************************************************************************
```

httpRequest


Memory Browser

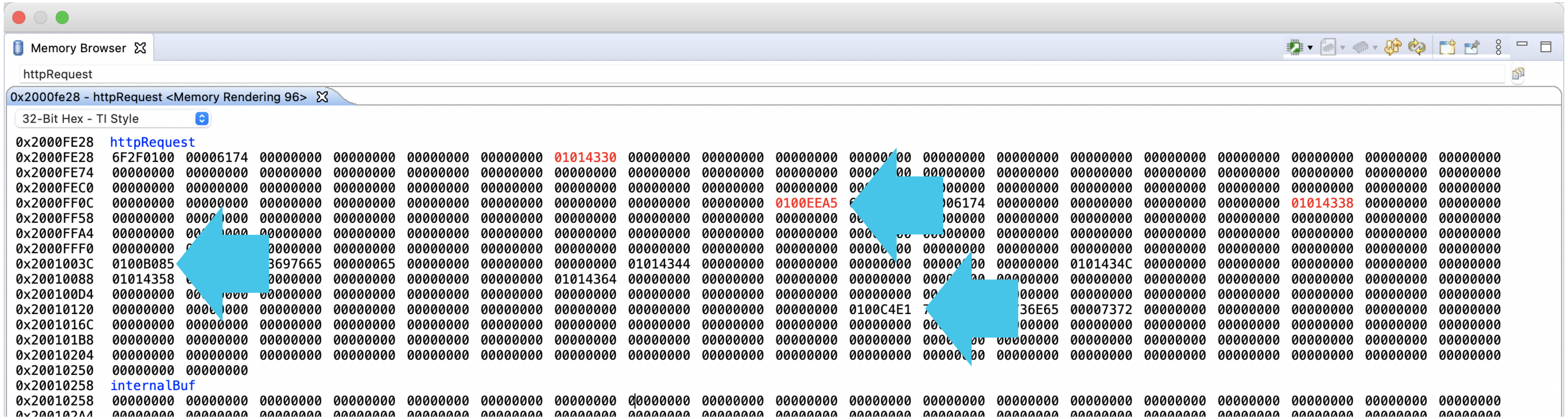
httpRequest

0x2000fe28 - httpRequest <Memory Rendering 96>

32-Bit Hex - TI Style

```
0x2000FE28   httpRequest
0x2000FE28   6F2F0100 00006174 00000000 00000000 00000000 00000000 01014330 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FE74   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FEC0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FF0C   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0100EEA5 6F2F0301 00006174 00000000 00000000 00000000 00000000 01014338 00000000 00000000
0x2000FF58   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FFA4   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FFF0   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2001003C   0100B085 642F0102 63697665 00000065 00000000 00000000 00000000 01014344 00000000 00000000 00000000 00000000 00000000 0101434C 00000000 00000000 00000000 00000000 00000000
0x20010088   01014358 00000000 00000000 00000000 00000000 00000000 01014364 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200100D4   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20010120   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0100C4E1 732F0103 6F736E65 00007372 00000000 00000000 00000000 00000000 00000000
0x2001016C   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200101B8   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20010204   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20010250   00000000 00000000
0x20010258   internalBuf
0x20010258   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200102A4   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

httpRequest



Memory Browser

httpRequest

0x2000fe28 - httpRequest <Memory Rendering 96>

32-Bit Hex - TI Style

```
0x2000FE28  httpRequest
0x2000FE28  6F2F0100 00006174 00000000 00000000 00000000 00000000 01014330 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FE74  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FEC0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 [illegible] [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FF0C  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0100EEA5 [illegible] [illegible] 00000000 00000000 00000000 00000000 01014338 00000000 00000000
0x2000FF58  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 [illegible] [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FFA4  00000000 [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 [illegible] [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2000FFF0  00000000 [illegible] [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x2001003C  0100B085 [illegible] [illegible] 00000065 00000000 00000000 00000000 01014344 00000000 00000000 00000000 00000000 00000000 0101434C 00000000 00000000 00000000 00000000 00000000
0x20010088  01014358 [illegible] 00000000 00000000 00000000 00000000 01014364 00000000 00000000 00000000 00000000 [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200100D4  00000000 [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 [illegible] [illegible] 00000000 00000000 00000000 00000000 00000000 00000000
0x20010120  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0100C4E1 [illegible] [illegible] 00007372 00000000 00000000 00000000 00000000 00000000
0x2001016C  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 [illegible] [illegible] 00000000 00000000 00000000 00000000 00000000 00000000
0x200101B8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20010204  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20010250  00000000 00000000
0x20010258  internalBuf
0x20010258  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200102A4  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Demo

Mitigation techniques

Mitigation techniques

Recommended function to check –

- malloc
- calloc
- realloc
- memalign
- valloc
- pvalloc
- aligned_alloc

Mitigation techniques

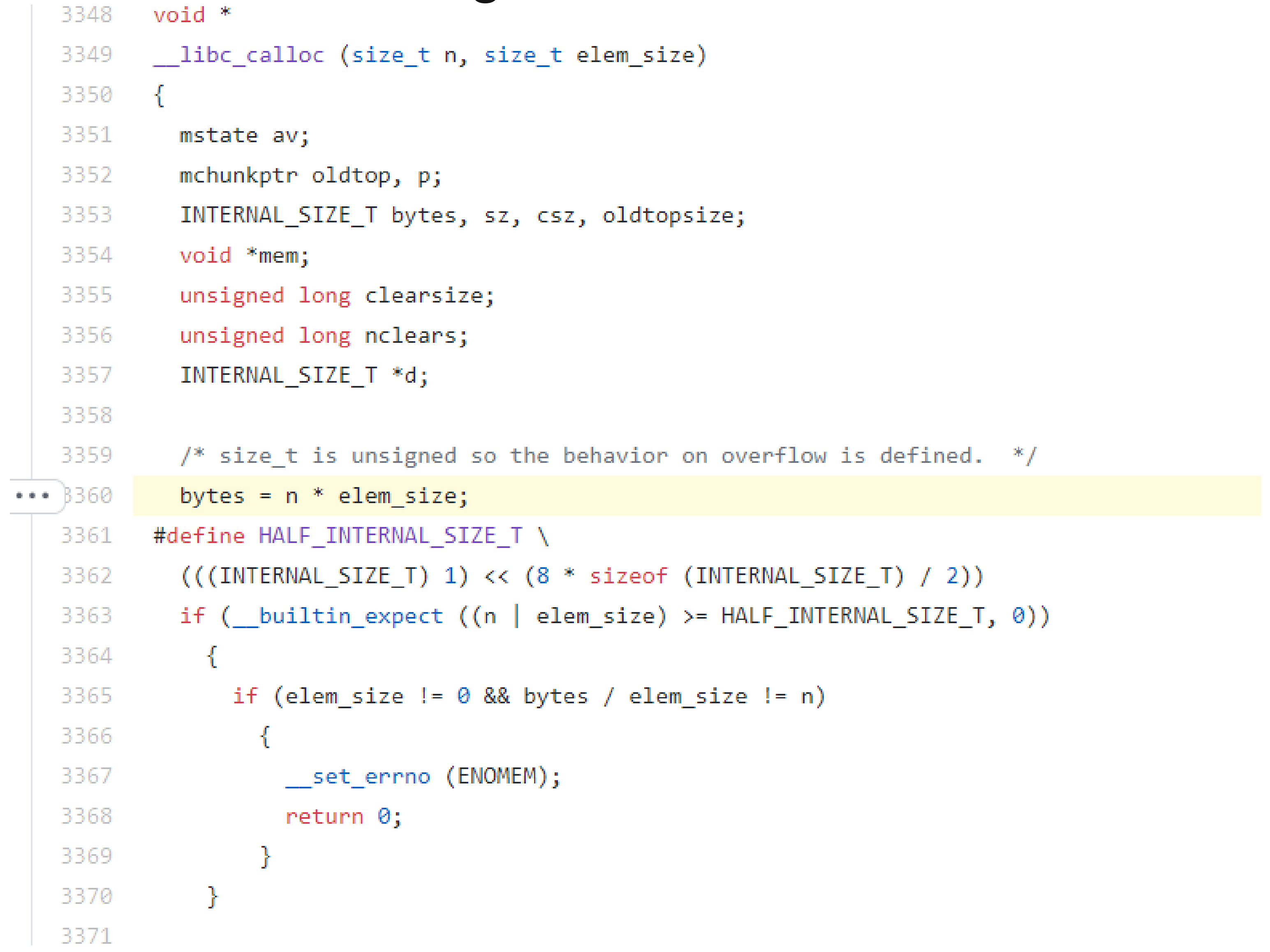
How should you do it?

- Start by checking-out the advisory.
- Reverse Engineer the binaries(always the best approach, in life).
- Source code review if it’s public.
- “Unit tests” - compile a small application the verify the environment.

It’s also recommended to check the macros that are being used in these functions

Mitigation techniques

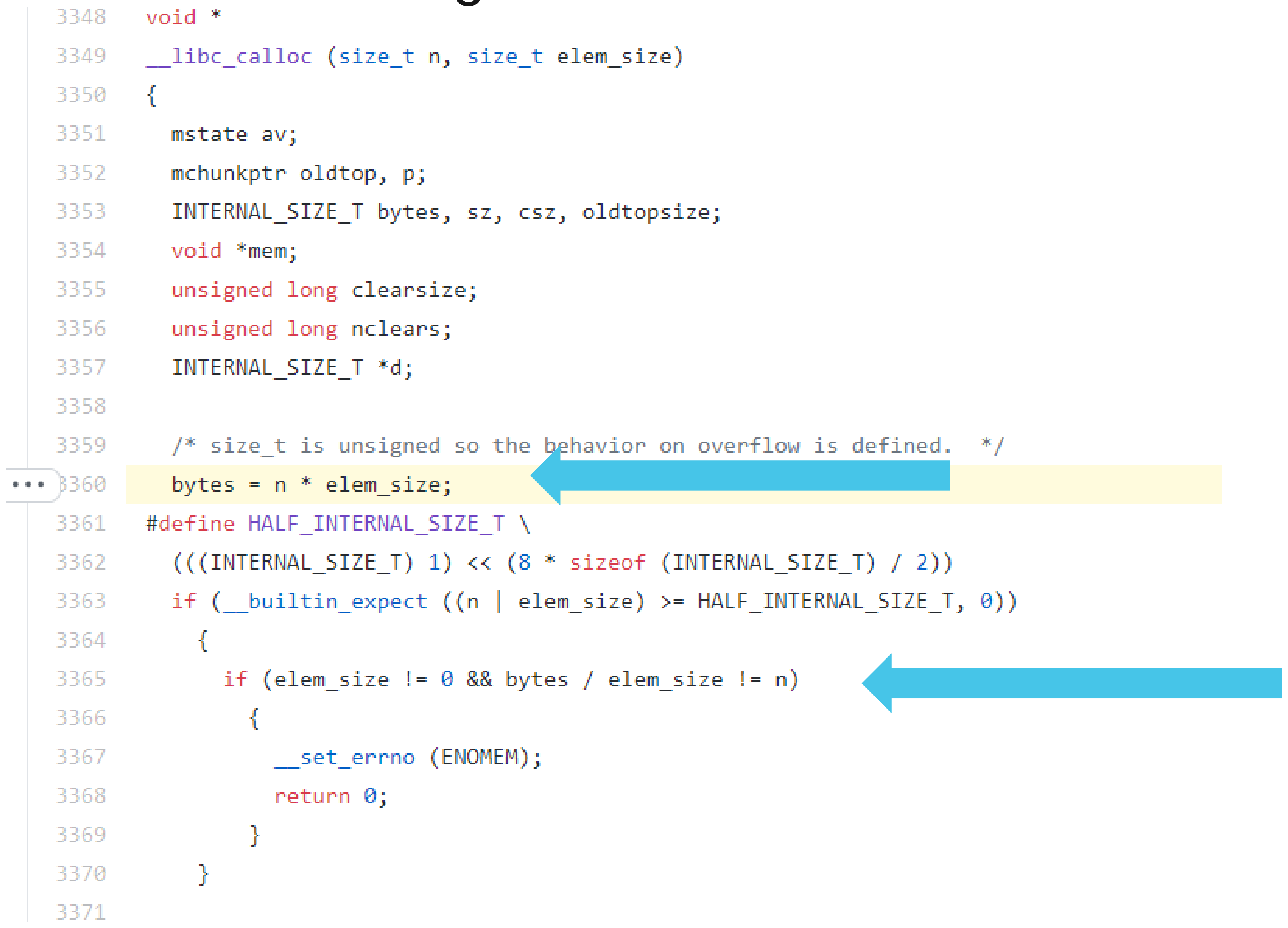
glibc



```
void *
__libc_calloc (size_t n, size_t elem_size)
{
  mstate av;
  mchunkptr oldtop, p;
  INTERNAL_SIZE_T bytes, sz, csz, oldtopsize;
  void *mem;
  unsigned long clearsize;
  unsigned long nclears;
  INTERNAL_SIZE_T *d;

  /* size_t is unsigned so the behavior on overflow is defined.  */
  bytes = n * elem_size;
#define HALF_INTERNAL_SIZE_T \
  (((INTERNAL_SIZE_T) 1) << (8 * sizeof (INTERNAL_SIZE_T) / 2))
  if (__builtin_expect ((n | elem_size) >= HALF_INTERNAL_SIZE_T, 0))
    {
      if (elem_size != 0 && bytes / elem_size != n)
        {
          __set_errno (ENOMEM);
          return 0;
        }
    }

```

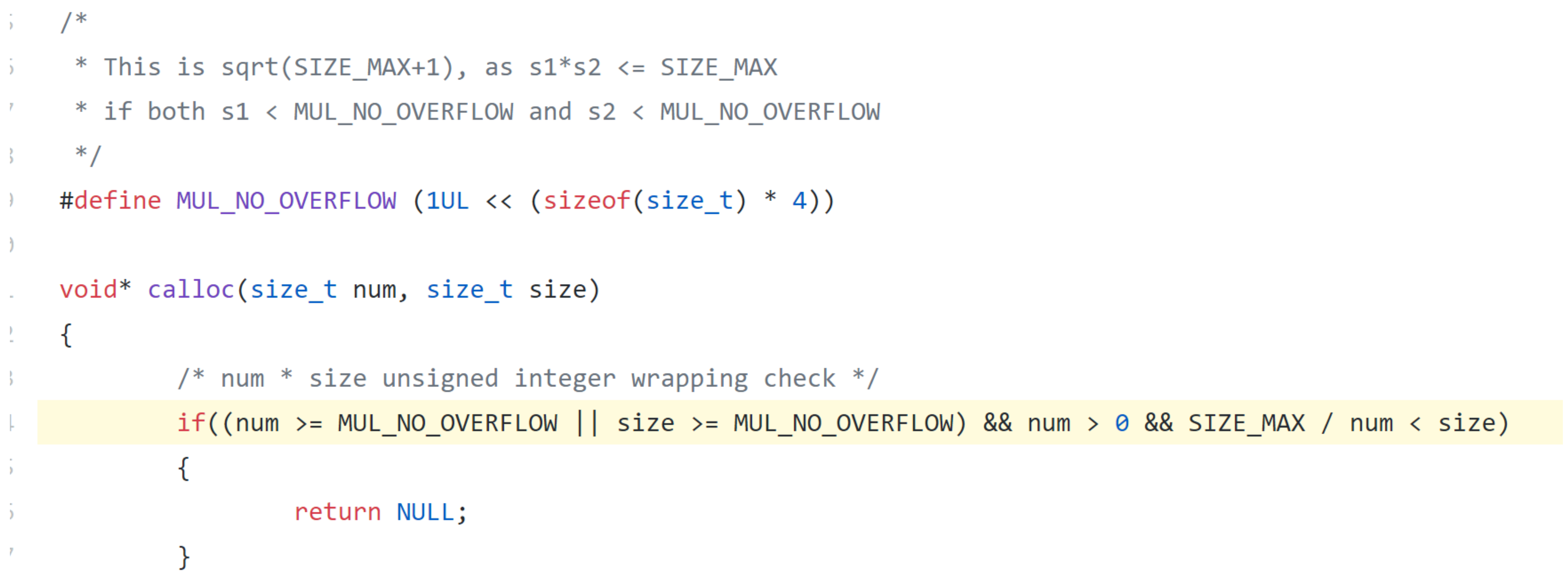
Mitigation techniques

glibc


```c
void *
__libc_calloc (size_t n, size_t elem_size)
{
  mstate av;
  mchunkptr oldtop, p;
  INTERNAL_SIZE_T bytes, sz, csz, oldtopsize;
  void *mem;
  unsigned long clearsize;
  unsigned long nclears;
  INTERNAL_SIZE_T *d;

  /* size_t is unsigned so the behavior on overflow is defined.  */
  bytes = n * elem_size;
#define HALF_INTERNAL_SIZE_T \
  (((INTERNAL_SIZE_T) 1) << (8 * sizeof (INTERNAL_SIZE_T) / 2))
  if (__builtin_expect ((n | elem_size) >= HALF_INTERNAL_SIZE_T, 0))
    {
      if (elem_size != 0 && bytes / elem_size != n)
        {
          __set_errno (ENOMEM);
          return 0;
        }
    }
```

Mitigation techniques

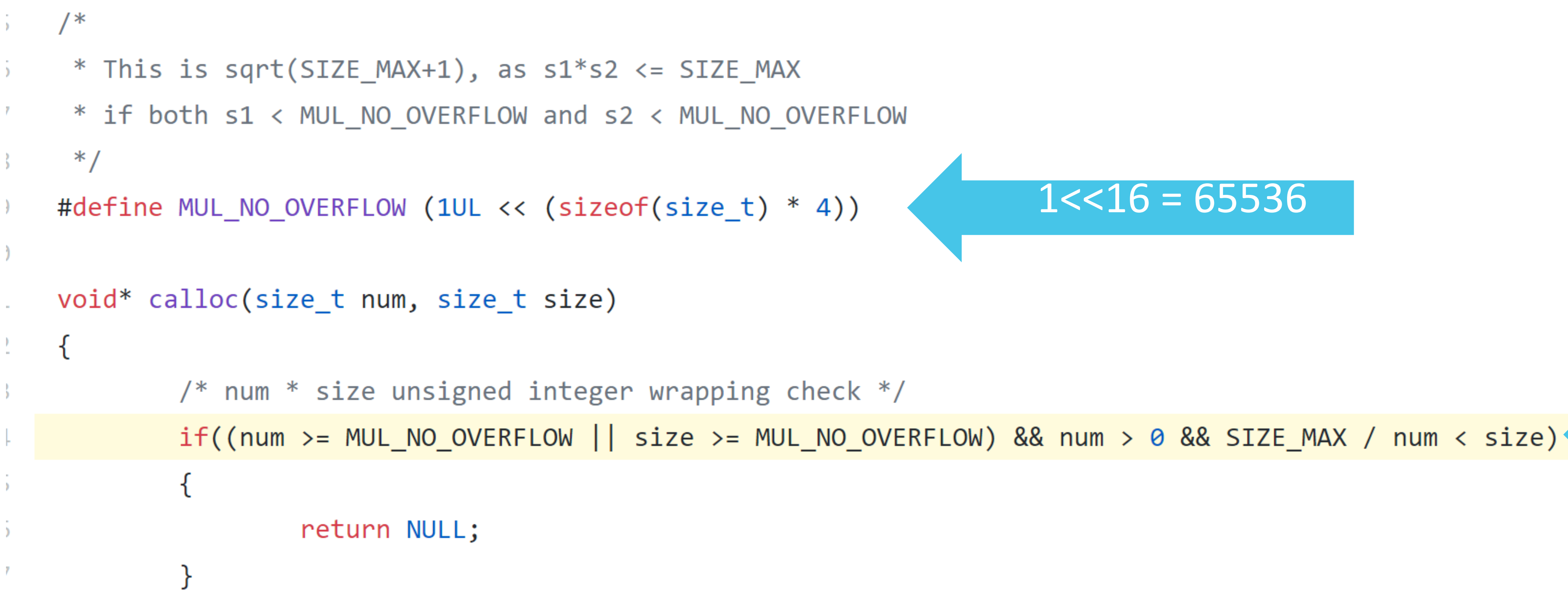
Embedded Artistry libc



```
/*
 * This is sqrt(SIZE_MAX+1), as s1*s2 <= SIZE_MAX
 * if both s1 < MUL_NO_OVERFLOW and s2 < MUL_NO_OVERFLOW
 */
#define MUL_NO_OVERFLOW (1UL << (sizeof(size_t) * 4))

void* calloc(size_t num, size_t size)
{
        /* num * size unsigned integer wrapping check */
        if((num >= MUL_NO_OVERFLOW || size >= MUL_NO_OVERFLOW) && num > 0 && SIZE_MAX / num < size)
        {
                return NULL;
        }
```

Mitigation techniques

Embedded Artistry libc



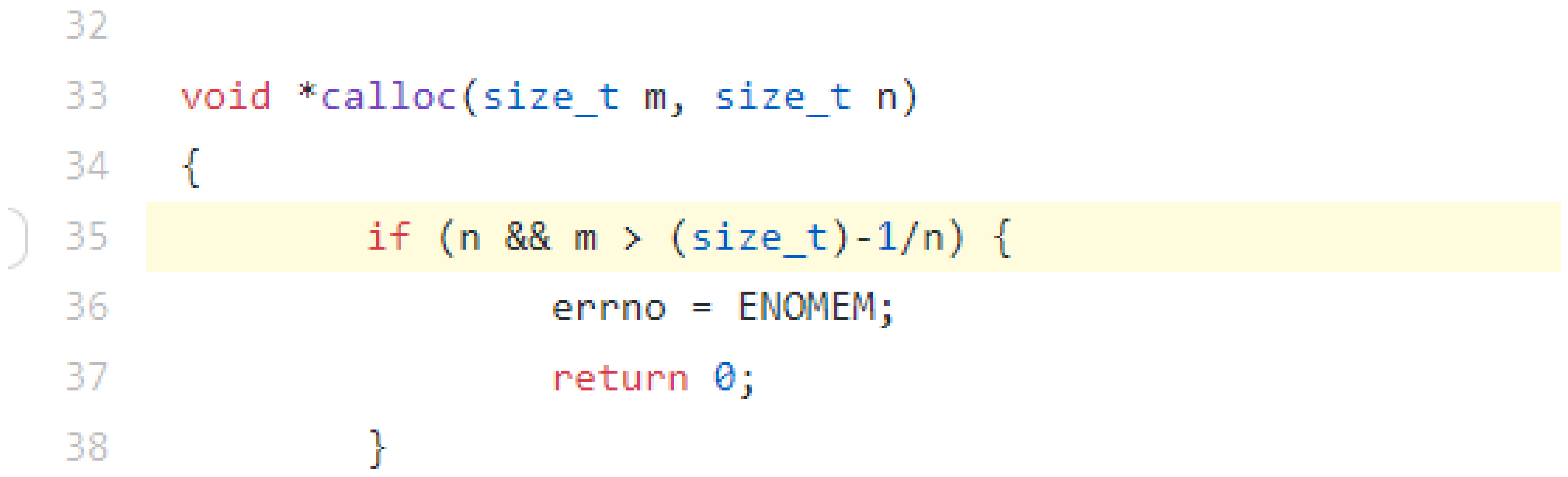
```
/*
 * This is sqrt(SIZE_MAX+1), as s1*s2 <= SIZE_MAX
 * if both s1 < MUL_NO_OVERFLOW and s2 < MUL_NO_OVERFLOW
 */
#define MUL_NO_OVERFLOW (1UL << (sizeof(size_t) * 4))

void* calloc(size_t num, size_t size)
{
        /* num * size unsigned integer wrapping check */
        if((num >= MUL_NO_OVERFLOW || size >= MUL_NO_OVERFLOW) && num > 0 && SIZE_MAX / num < size)
        {
                return NULL;
        }
```

1<<16 = 65536

SIZE_MAX = 0xffffffff

Mitigation techniques

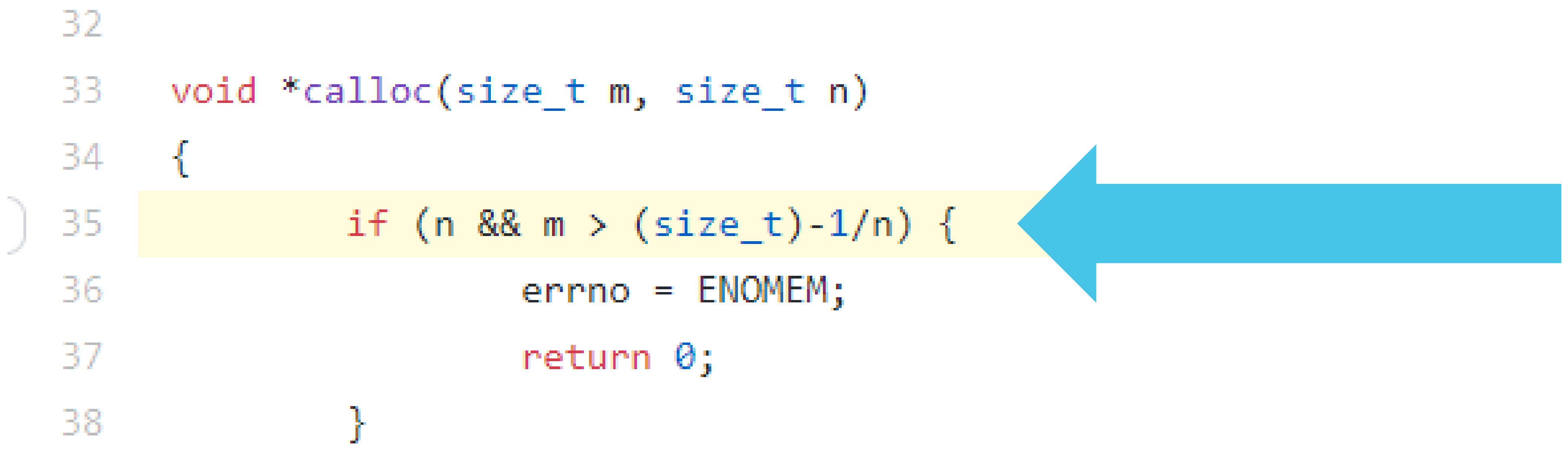
musl



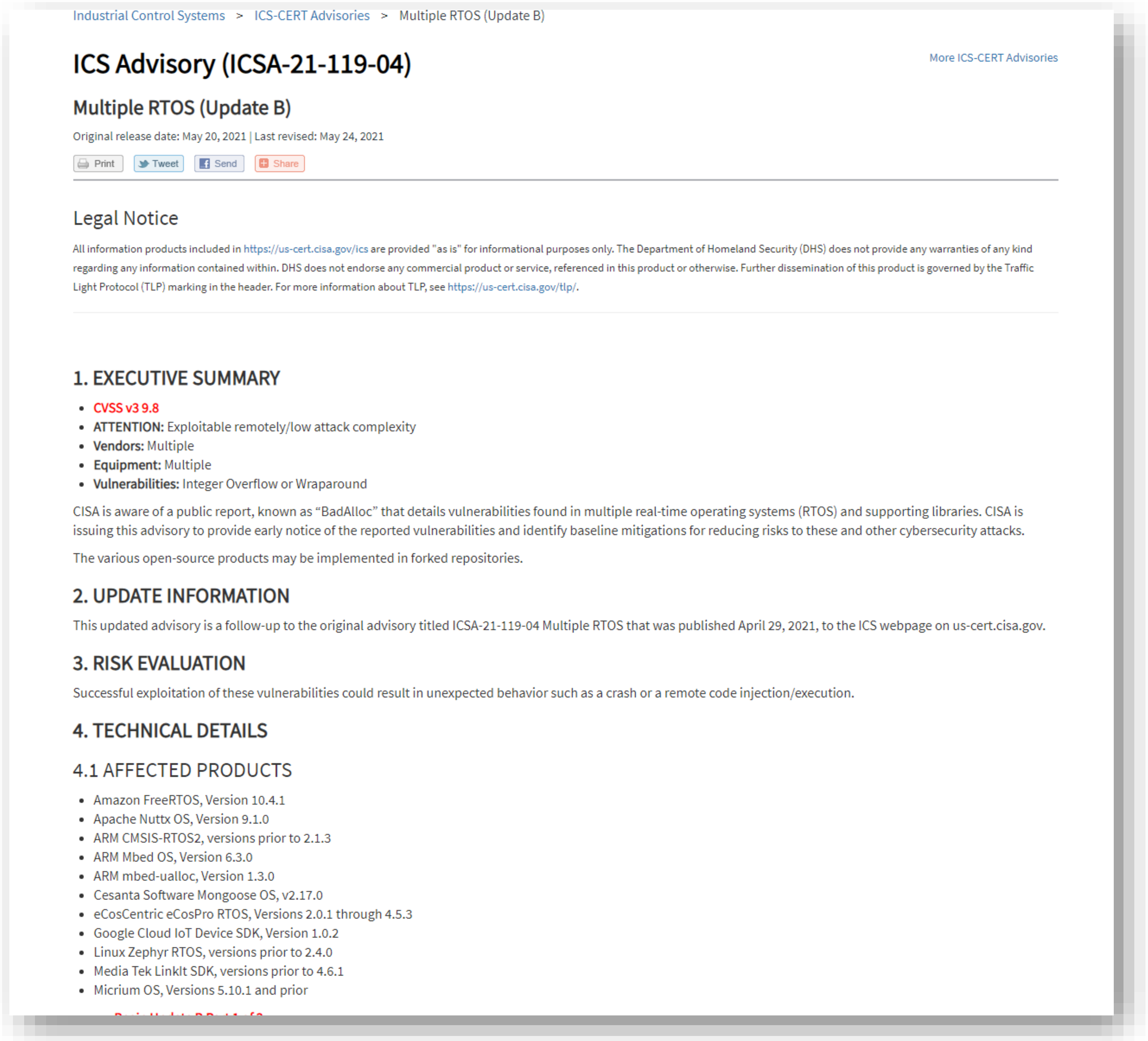
```
void *calloc(size_t m, size_t n)
{
        if (n && m > (size_t)-1/n) {
                errno = ENOMEM;
                return 0;
        }
```

Mitigation techniques

musl



```
void *calloc(size_t m, size_t n)
{
	if (n && m > (size_t)-1/n) {
		errno = ENOMEM;
		return 0;
	}
```



Industrial Control Systems > ICS-CERT Advisories > Multiple RTOS (Update B)

More ICS-CERT Advisories

ICS Advisory (ICSA-21-119-04)

Multiple RTOS (Update B)

Original release date: May 20, 2021 | Last revised: May 24, 2021

Legal Notice

All information products included in https://us-cert.cisa.gov/ics are provided "as is" for informational purposes only. The Department of Homeland Security (DHS) does not provide any warranties of any kind regarding any information contained within. DHS does not endorse any commercial product or service, referenced in this product or otherwise. Further dissemination of this product is governed by the Traffic Light Protocol (TLP) marking in the header. For more information about TLP, see https://us-cert.cisa.gov/tlp/.

1. EXECUTIVE SUMMARY

- CVSS v3 9.8
- **ATTENTION:** Exploitable remotely/low attack complexity
- **Vendors:** Multiple
- **Equipment:** Multiple
- **Vulnerabilities:** Integer Overflow or Wraparound

CISA is aware of a public report, known as “BadAlloc” that details vulnerabilities found in multiple real-time operating systems (RTOS) and supporting libraries. CISA is issuing this advisory to provide early notice of the reported vulnerabilities and identify baseline mitigations for reducing risks to these and other cybersecurity attacks.

The various open-source products may be implemented in forked repositories.

2. UPDATE INFORMATION

This updated advisory is a follow-up to the original advisory titled ICSA-21-119-04 Multiple RTOS that was published April 29, 2021, to the ICS webpage on us-cert.cisa.gov.

3. RISK EVALUATION

Successful exploitation of these vulnerabilities could result in unexpected behavior such as a crash or a remote code injection/execution.

4. TECHNICAL DETAILS

4.1 AFFECTED PRODUCTS

- Amazon FreeRTOS, Version 10.4.1
- Apache Nuttx OS, Version 9.1.0
- ARM CMSIS-RTOS2, versions prior to 2.1.3
- ARM Mbed OS, Version 6.3.0
- ARM mbed-ualloc, Version 1.3.0
- Cesanta Software Mongoose OS, v2.17.0
- eCosCentric eCosPro RTOS, Versions 2.0.1 through 4.5.3
- Google Cloud IoT Device SDK, Version 1.0.2
- Linux Zephyr RTOS, versions prior to 2.4.0
- Media Tek LinkIt SDK, versions prior to 4.6.1
- Micrium OS, Versions 5.10.1 and prior

Begin Update B Part 1 of 2

Q&A

https://msrc-blog.microsoft.com/2021/04/29/badalloc-memory-allocation-vulnerabilities-could-affect-wide-range-of-iot-and-ot-devices-in-industrial-medical-and-enterprise-networks/

Bing for ”ICSA-21-119-04”