# RST IV&V Issues

## RST-IVV-173: ICDH Build 2: cFS Unchecked Return Values from Function OS_ConvertToArrayIndex Could Result in Use of Uninitialized Variables

| | |
|---|---|
| Attachments: | |
| IVV Severity: | 3 |
| Issue Category: | Code |
| Issue Type: | Incorrect Code |
| Count: | 4 |
| Description: | There are four instances [2, 3, 4, 5] where an uninitialized value (TaskID, TaskIndex, local_id, and array_idx respectively) could be used to access an array if an unknown object id is passed to function OS_ConvertToArrayIndex. None of these four instances check the return value of the OS_ConvertToArrayIndex before using the variable to index an array.<br><br>Four calls [2, 3, 4, 5] are made to the OS_ConvertToArrayIndex function [1] to assign a value to a variable, which will later be used to access an array index. In the event that OS_ConvertToArrayIndex succeeds, the pointer passed as the “ArrayIndex” parameter will be populated with the appropriate array index. However, in the event that the function call fails, no value is assigned to the “ArrayIndex” parameter, and an error code is returned [1]. Although the function does not initialize the variable to be used when the call fails, none of the four calls [2, 3, 4, 5] check the return value of the OS_ConvertToArrayIndex before using the variable to index an array. |
| Recommended Actions: | Add a check when using the OS_ConvertToArrayIndex function to make sure it succeeds prior to using the value that it assigns to ensure that value is valid for use as an array index. |
| Impact: | Improper use of the OS_ConvertToArrayIndex function could result in unexpected/invalid values being used as array indices when attempting to access elements of an array. The unpredictable nature of using an invalid/unknown value as an array index due to using an uninitialized variable is a defect resulting in the degradation of system dependability. |
| Workaround: | |
| References: | Instrument Command and Data Handling (ICDH) Flight Software, Build 2.0, dated 14 August 2020<br>Version cFS 6.7 |

1. File: osal\src\os\shared\osapi-idmap.c

```
959: int32 OS_ConvertToArrayIndex(uint32 object_id, uint32 *ArrayIndex)
960: {
961:     uint32 max_id;
962:     int32 return_code;
963:
964:     max_id = OS_GetMaxForObjectType(object_id >> OS_OBJECT_TYPE_SHIFT);
965:     if (max_id == 0)
966:     {
967:         return_code = OS_ERR_INCORRECT_OBJ_TYPE;
968:     }
969:     else
970:     {
971:         *ArrayIndex = (object_id & OS_OBJECT_INDEX_MASK) % max_id;
972:         return_code = OS_SUCCESS;
973:     }
974:
975:     return return_code;
```

2. File: cfe\fsw\cfe-core\src\es\cfe_es_api.c

```
503: bool CFE_ES_RunLoop(uint32 *RunStatus)
504: {
505:     bool    ReturnCode;
506:     int32  Status;
507:     uint32 AppID;
508:     uint32 TaskID;
509:
510:     CFE_ES_LockSharedData(__func__,__LINE__);
511:
512:     /*
513:     ** Get App ID
514:     */
515:     Status = CFE_ES_GetAppIDInternal(&AppID);
516:
517:     if ( Status == CFE_SUCCESS )
518:     {
519:
520:         /*
521:         ** Get the Task ID for the main task
522:         */
523:         OS_ConvertToArrayIndex(CFE_ES_Global.AppTable[AppID].TaskInfo.MainTaskId, &TaskID);
524:
525:         /*
526:         ** Increment the execution counter for the main task
527:         */
528:         CFE_ES_Global.TaskTable[TaskID].ExecutionCounter++;
```

3. File: cfe\fsw\cfe-core\src\es\cfe_es_start.c

```
740: void  CFE_ES_CreateObjects(void)
741: {
742:     int32     ReturnCode;
743:     uint32    TaskIndex;
744:     bool      AppSlotFound;
745:     uint16    i;
746:     uint16    j;
747:
```

```
748:    CFE_ES_WriteToSysLog("ES Startup: Starting Object Creation calls.\n");
---
840:            else
841:            {
842:               OS_ConvertToArrayIndex(CFE_ES_Global.AppTable[j].TaskInfo.MainTaskId, &TaskIndex);
843:
844:               /*
845:               ** Allocate and populate the CFE_ES_Global.TaskTable entry
846:               */
847:               if ( CFE_ES_Global.TaskTable[TaskIndex].RecordUsed == true )
848:               {
849:                  CFE_ES_WriteToSysLog("ES Startup: CFE_ES_Global.TaskTable record used error for App: %s, continuing.\n",
850:                                       CFE_ES_ObjectTable[i].ObjectName);
851:               }
852:               else
853:               {
854:                  CFE_ES_Global.TaskTable[TaskIndex].RecordUsed = true;
855:               }
```

4. File: osal\src\os\rtems\ostimer.c

```
112: static rtems_timer_service_routine OS_TimeBase_ISR(rtems_id rtems_timer_id, void *arg)
113: {
114:    OS_U32ValueWrapper_t user_data;
115:    uint32 local_id;
116:    OS_impl_timebase_internal_record_t *local;
117:
118:    user_data.opaque_arg = arg;
119:    OS_ConvertToArrayIndex(user_data.value, &local_id);
120:    local = &OS_impl_timebase_table[local_id];
```

5. File: osal\src\os\shared\osapi-dir.c

```
340: os_dirent_t *OS_readdir (os_dirp_t directory)
341: {
342:    OS_Dirp_Xltr_t  dirdescptr;
343:    uint32 array_idx;
344:    os_dirent_t *tempptr;
345:
346:    dirdescptr.dirp = directory;
347:    OS_ConvertToArrayIndex(dirdescptr.dir_id, &array_idx);
348:
349:    tempptr = &OS_dir_table[array_idx].dirent_object;
```

| | |
|---|---|
| Resolution Chronology: | |
| Capability: | Perform WFI Operations |

| **RST-IVV-175: ICDH Build 2: cFS OSAL OS_ConsoleAPI_Init Function Returns the Wrong Value** | |
|---|---|
| Attachments: | |
| IVV Severity: | 4 |
| Issue Category: | Code |
| Issue Type: | Incorrect Code |
| Count: | 1 |
| Description: | On line 113 of osapi-printf.c in the OS_ConsoleAPI_Init function, a variable “return_code” is declared and then updated several times prior to returning from the function, however, on line 148 the function statically returns OS_SUCCESS regardless of the value of “return_code” [1]. |
| Recommended Actions: | In function OS_ConsoleAPI_Init, update the return statement to return “return code” instead of statically returning “OS_SUCCESS”. |
| Impact: | The inability to determine when this function has succeeded properly may affect the ability to use the printf function for debugging. Given that this has no impact in flight and only affects development and testing, this is a defect impacting maintainability on current mission or reuse on future missions. |
| Workaround: | |
| References: | Instrument Command and Data Handling (ICDH) Flight Software, Build 2.0, dated 14 August 2020 |

1. File: osal\src\os\shared\osapi-printf.c

```
110: int32 OS_ConsoleAPI_Init(void)
111: {
112:    OS_console_internal_record_t *console;
113:    int32 return_code;
114:    uint32 local_id;
115:    OS_common_record_t *record;
116:
117:    memset(&OS_console_table, 0, sizeof(OS_console_table));
118:
119:
120:    /*
121:     * Configure a console device to be used for OS_printf() calls.
122:     */
123:    return_code = OS_ObjectIdAllocateNew(OS_OBJECT_TYPE_OS_CONSOLE, OS_PRINTF_CONSOLE_NAME, &local_id, &record);
124:    if (return_code == OS_SUCCESS)
125:    {
126:       console = &OS_console_table[local_id];
127:
128:       record->name_entry = console->device_name;
129:       strcpy(console->device_name, OS_PRINTF_CONSOLE_NAME);
130:
131:       /*
132:        * Initialize the ring buffer pointers
133:        */
134:       console->BufBase = OS_printf_buffer_mem;
135:       console->BufSize = sizeof(OS_printf_buffer_mem);
```

```
136:
137:        return_code = OS_ConsoleCreate_Impl(local_id);
138:
139:        /* Check result, finalize record, and unlock global table. */
140:        return_code = OS_ObjectIdFinalizeNew(return_code, record, &OS_SharedGlobalVars.PrintfConsoleId);
141:
142:        /*
143:         * Printf can be enabled by default now that the buffer is configured.
144:         */
145:        OS_SharedGlobalVars.PrintfEnabled = true;
146:    }
147:
148:    return OS_SUCCESS;
149: } /* end OS_ConsoleAPI_Init */
```

| | |
|---|---|
| Resolution Chronology: | |
| Capability: | Perform WFI Operations |

**RST-IVV-177: ICDH Build 2: cFS Tool "elf2cfetbl" Experiences a Buffer Overflow when the Section Header Description Exceeds 60 Characters**

| | |
|---|---|
| Attachments: | |
| IVV Severity: | 4 |
| Issue Category: | Code |
| Issue Type: | Incorrect Code |
| Count: | 1 |
| Description: | When reading the section header using the function GetSectionHeader() on line 1667 of file elf2cfetbl.c [Ref. 1], the array of "VerboseStr", which is of size 60, will experience a buffer overflow when "i", the iteration integer used in the while loop on the same line, exceeds the value of 60. This can occur because the maximum characters allowed for the section header is 128 as defined by the macro MAX_SECTION_HDR_NAME_LEN on line 44 in the same file. On line 1674, "VerboseStr" is expected to have length MAX_SECTION_HDR_NAME_LEN - 1, which is 127. So, if the "SrcFileDesc" points to a section header with longer than 60 characters, on line 1667, the iterator "i" will exceed 60.<br><br>Note: The elf2cfetbl utility is a ground tool that provides a method of converting an object file containing the desired contents of a cFE application's Table Image into a binary file that is compatible with the cFE Table Services for loading the image. |
| Recommended Actions: | Declare the "VerboseStr" in function GetSectionHeader() to be an array of characters for at least MAX_SECTION_HDR_NAME_LEN, 128. |
| Impact: | A buffer overflow can result in unpredictable/undesired behavior during the execution of the elf2cfetbl utility. With the current code, the user would need to change the section header description to be shorter than 60 characters and re-run the elf2cfetbl tool if the header were greater than 60 characters long. This creates inconvenience for operators or other project personnel. |
| Workaround: | |
| References: | Instrument Command and Data Handling (ICDH) Flight Software, Build 2.0, dated 14 August 2020<br><br>Ref. 1: File location: /toos/elf2cfetbl/elf2cfetbl.c<br>44 #define MAX_SECTION_HDR_NAME_LEN (128)<br>…<br>1667 while ((VerboseStr[i] = fgetc(SrcFileDesc)) != '\0')<br>1668 {<br>1669 i++;<br>1670 }<br>1671 if (Verbose) printf("%s\n", VerboseStr);<br>1672<br>1673 /* Save the name for later reference */<br>1674 strncpy(SectionNamePtrs[SectionIndex], VerboseStr, (MAX_SECTION_HDR_NAME_LEN-1));<br>1675 SectionNamePtrs[SectionIndex][(MAX_SECTION_HDR_NAME_LEN-1)] = '\0'; |
| Resolution Chronology: | |
| Capability: | Perform WFI Operations |

| RST-IVV-179: ICDH Build 2: cFS Tool Contains Potentially Null Pointers that are De-referenced in Function AllocateSymbols | |
|---|---|
| Attachments: | |
| IVV Severity: | 4 |
| Issue Category: | Code |
| Issue Type: | Incorrect Code |
| Count: | 2 |
| Description: | There are 2 instances where pointers are checked for Null and then later de-referenced by function AllocateSymbols in file elf2cfetbl.c [Ref. 1], regardless of whether they are valid:<br>1. Pointer 'SymbolNames', of type Elf_Sym, is checked for NULL on line 841, and is dereferenced on line 850.<br>2. Pointer 'SymbolPtrs', of type Elf_Sym, is checked for NULL on line 834, and is dereferenced on line 849.<br>If either pointer is Null, a local variable, Status, of type int32, is set to FAILED. However, the Status is not checked prior to de-referencing the pointers. Null pointers should not be de-referenced as doing so will result in application exit or other unpredictable or undesired behavior. |
| Recommended Actions: | Recommend only de-referencing pointers, SymbolPtrs and SymbolNames, when Status is set to SUCCESS. |
| Impact: | De-referencing a Null pointer, in this case, will create an inconvenience for operators, crew or other projects' personnel since the function, which is part of ground software, will crash and exit. |
| Workaround: | |
| References: | Ref. 1: Instrument Command and Data Handling (ICDH) Flight Software, Build 2.0, dated 14 August 2020<br>/tools/elf2cfetbl/elf2cfetbl.c<br><br>143 union Elf_Sym **SymbolPtrs = NULL;<br>144 char **SymbolNames;<br>…<br>820 int32 AllocateSymbols(void)<br>821 {<br>…<br>834 if (SymbolPtrs == NULL)<br>835 {<br>836 printf("Error! Insufficient memory for number of Symbols in '%s'!\n", SrcFilename);<br>837 Status = FAILED;<br>838 }<br>839<br>840 SymbolNames = malloc(sizeof(char *)*NumSymbols);<br>841 if (SymbolNames == NULL)<br>842 {<br>843 printf("Error! Insufficient memory for number of Symbols in '%s'!\n", SrcFilename);<br>844 Status = FAILED;<br>845 }<br>846<br>847 for (i=0; i<NumSymbols; i++)<br>848 {<br>849 SymbolPtrs[i] = NULL;<br>850 SymbolNames[i] = NULL;<br>851 } |
| Resolution Chronology: | |
| Capability: | Perform WFI Operations |