Fault Tolerant Overflow

Moderate liydu published GHSA-8jqf-wjhq-4w9f 22 days ago

Package

FileX (Azure RTOS)

Affected versions	Patched versions
< 6.2.0	6.2.0

Description

Impact

The Fault Tolerant feature of Azure RTOS FileX includes integer under and overflows which may be exploited to achieve buffer overflow and modify memory contents.

When a valid log file with correct ID and checksum is detected by the `_fx_fault_tolerant_enable` function an attempt to recover the previous failed write operation is taken by call of `_fx_fault_tolerant_apply_logs` . This function iterates through the log entries and performs required recovery operations. When properly crafted a log including entries of type `FX_FAULT_TOLERANT_DIR_LOG_TYPE` may be utilized to introduce unexpected behavior. Multiple values are retrieved from the log file thus controlled by a potential attacker, these include log total size (size variable), number of log entries (`remaining_logs`), entry type (`log_type`), length (`log_len`), `copy_size` etc. Certain combinations of values may be utilized to bypass validation by exploiting a integer overflow (and optionally underflow).

For example, the comparison of copy_offset and copy_size against `media_ptr -> fx_media_memory_size` may be bypassed by using a value of log_len smaller than `FX_FAULT_TOLERANT_DIR_LOG_ENTRY_SIZE` to cause an integer underflow and set copy_size to a unexpectedly large value. With correct selection of copy_offset the sum of copy_offset and copy_size will result in an integer overflow resulting in check bypass. Alternatively, one may manipulate copy_offset and a not-undeflown value of copy_size to again cause integer overflow of the sum. With a substantially large offset value the destination address of mempcy should also be overflown.

With the condition bypassed one may force a buffer overflow in the memcpy call with the possibility to manipulate both the destination address with copy_offset and amount of copied data with copy_size.

For example with log_len set to 15 copy_size will have the value of 4294967295 due to integer underflow,
with copy_offset set to 1 the result of sum of copy_size and copy_offset will be 0 and will result in bypass of the comparison against `media_ptr -> fx_media_memory_size` allowing a buffer overflow.

```
case FX_FAULT_TOLERANT_DIR_LOG_TYPE:

/* This is a DIR log. */
dir_log = (FX_FAULT_TOLERANT_DIR_LOG *)current_ptr;

log_sector = _fx_utility_64_unsigned_read((UCHAR *)&amp;dir_log -&gt; fx_fault_tolerant_dir_log_

/* Get the destination sector. */
status = _fx_utility_logical_sector_read(media_ptr,
log_sector,
media_ptr -&gt; fx_media_memory_buffer,
((ULONG) 1), FX_DATA_SECTOR);

if (status != FX_SUCCESS)
{

/* Return the error status. */
return(status);
}

/* Set copy information. */
copy_offset = _fx_utility_32_unsigned_read((UCHAR *)&amp;dir_log -&gt; fx_fault_tolerant_dir_log

copy_size = log_len - FX_FAULT_TOLERANT_DIR_LOG_ENTRY_SIZE;

if ((copy_offset + copy_size) &gt; media_ptr -&gt; fx_media_memory_size)
{
return(FX_FILE_CORRUPT);
}

/* Copy data into destination sector. */
memcpy(media_ptr -&gt; fx_media_memory_buffer + copy_offset, /* Use case of memcpy is verified.
current_ptr + FX_FAULT_TOLERANT_DIR_LOG_ENTRY_SIZE, copy_size);
```

Patches

We analyzed this bug and determined that we needed to fix it. This fix will be included in FileX release v6.2.0.

Workarounds

Here is the proposed fix to line 218 in `fx_fault_tolerant_apply_logs.c` . Change it to,

```
if (((ULONG64)copy_offset + (ULONG64)copy_size) > (ULONG64)(media_ptr -> fx_media_memory_size))
```

This fix will avoid overflow in arithmetic. And thus ensures the size of data to copy will not overflow.

For more information

If you have any questions or comments about this advisory:

Open an issue in azure-rtos/filex
Post question on Microsoft Q&A

Severity

Moderate

CVE ID

CVE-2022-39343

Weaknesses

No CWEs