

flipperdevices / flipperzero-firmware Public[Code](#) [Issues](#) 176 [Pull requests](#) 84 [Actions](#) [Security and quality](#) [Insights](#)[New issue](#)

Potential thread stack overflow in main #4332

[Open](#)[#4350](#)

k6dpvrmm8z-g glitch opened on Jan 21



Issue details

While some detailed build configuration, There are potential stack overflow in thread function named main

[flipperzero-firmware/targets/f7/stm32wb55xx_flash.ld](#)

Line 6 in c9ab2b6

```
6     _stack_size = 0x400; /* required amount of stack */
```

In this line, main allows 1024 stack size.

However, after checking the stack using our internally developed tool, it might have 1464 Bytes in with some detailed configurations.

Steps to produce

****1. In cc.scons, Add this line:**

In 38 line..

```
"-fstack-usage"
```



This will help you automatically calculating stack size of each function.

2. Build flipperzero-firmware normally, using this configurations:

```
./fbt
```

Now we can get stack usage file (*.su) for each source file, So we can manually check stack size of each function.

In case of main :

There are large call stack with this flow:

```

main (main) ⇒ 8 bytes
flipper_boot_update_exec (flipper_boot_update_exec) ⇒ 24 bytes
flipper_update_process_manifest (flipper_update_process_manifest) ⇒ 920 bytes
update_manifest_init_mem (update_manifest_init_mem) ⇒ 24 bytes
update_manifest_init_from_ff.isra (update_manifest_init_from_ff.isra.0) ⇒ 16 bytes
flipper_format_read_header (flipper_format_read_header) ⇒ 16 bytes
flipper_format_read_string (flipper_format_read_string) ⇒ 16 bytes
flipper_format_stream_read_value_line (flipper_format_stream_read_value_line) ⇒ 88 bytes
flipper_format_stream_seek_to_key (flipper_format_stream_seek_to_key) ⇒ 88 bytes
stream_eof (stream_eof) ⇒ 8 bytes
__furi_crash_implementation (__furi_crash_implementation) ⇒ 40 bytes
__furi_print_name (__furi_print_name) ⇒ 8 bytes
__furi_put_uint32_as_text (__furi_put_uint32_as_text) ⇒ 24 bytes
furi_log_puts (furi_log_puts) ⇒ 8 bytes
furi_log_tx (furi_log_tx) ⇒ 24 bytes
furi_mutex_acquire (furi_mutex_acquire) ⇒ 16 bytes
xQueueTakeMutexRecursive (xQueueTakeMutexRecursive) ⇒ 16 bytes
xQueueSemaphoreTake (xQueueSemaphoreTake) ⇒ 40 bytes
xTaskResumeAll (xTaskResumeAll) ⇒ 40 bytes
xTaskIncrementTick (xTaskIncrementTick) ⇒ 40 bytes

```

SUM ⇒ 1464 bytes

So, there are potentially occur stack overflow in main function.

Environment

Version

Commit [ad2a800](#)



bad-antics on Mar 5



Excellent analysis with the `-fstack-usage` approach. The call chain you've documented is thorough:

```

main → flipper_boot_update_exec → flipper_update_process_manifest → ... →
xTaskIncrementTick
Total: 1464 bytes vs 1024 byte stack limit

```



A few observations:

1. Worst-case vs typical execution:

The call chain through `__furi_crash_implementation` → `__furi_print_name` → `furi_log_*` is the crash handler path. Under normal execution (no crash), the peak would be lower. However, if a crash occurs during `flipper_update_process_manifest`, it would indeed overflow the 1024-byte stack.

2. The `flipper_update_process_manifest` path (920 bytes) is the main concern:

Even without the crash handler, `main` → `flipper_boot_update_exec` → `flipper_update_process_manifest` alone uses ~952 bytes, leaving only 72 bytes of headroom for any interrupt-triggered stack usage.

3. Suggested fix:

The linker script at `targets/f7/stm32wb55xx_flash.ld` should increase the main thread stack:

```
/* Change from: */
_stack_size = 1024;
/* To at minimum: */
_stack_size = 2048;
```



Or alternatively, `flipper_update_process_manifest` could be refactored to reduce its 920-byte stack frame — the `update_manifest_init_mem` structure is likely being allocated on the stack and could be heap-allocated instead.

4. STM32WB55 consideration:

On this MCU with 256KB SRAM, increasing the main stack to 2048 bytes is negligible. The extra 1KB cost is worth the safety margin, especially since the update manifest parsing is a critical boot path.

Great find with the automated stack analysis tool — this kind of static analysis should ideally be part of the CI pipeline.

herbenderbler added 3 commits that reference this issue [on Mar 12](#)

fix(boot): increase main thread stack to prevent overflow ([flipperdev](#), `de0fa1d`)

refactor(boot): heap-allocate large FatFS structures in update path (: `5908589`)

build: disable -fstack-usage in production builds `421c175`

herbenderbler linked a pull request that will close this issue [on Mar 12](#)

[ISSUE-4332: Fix boot-path stack overflow risk and reduce update stack footprint #4350](#)

k6dpvrmm8z-glitch 2 weeks ago

Author

Hi,
Thanks for addressing this issue!
I have reserved a CVE ID (CVE-2026-30363) for this vulnerability.
Since the issue has been fixed, I plan to proceed with public disclosure and CVE publication, using this issue and the associated commits as references.
Please let me know if you have any concerns.
Thanks!

[Sign up for free](#) to join this conversation on **GitHub**. Already have an account? [Sign in to comment](#)

Metadata

Assignees

No one assigned

Labels

No labels

Type

No type

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

 **ISSUE-4332: Fix boot-path stack overflow risk and reduce update stack footprint**

flipperdevices/flipperzero-firmware

Participants



