EasyFlash Log 模块异常掉电后地址对齐缺陷报告

报告编号: EF-2024-001
发现日期: 2024年
严重程度: 严重 (Critical)
影响版本: EasyFlash v3.x 及之前版本
影响组件: `ef_utils.c` - 日志功能 (`EF_USING_LOG`)
目标平台: STM32F10x 系列 (已验证), 其他使用内部 Flash 的 MCU

---

1. 摘要 (Executive Summary)

在 EasyFlash 日志功能中，当 Flash 擦除过程中发生异常掉电，重新上电后日志系统可能无法正常工作。根因是日志区地址计算函数 `ef_find_sec_using_end_addr()` 在计算写入地址时未强制 4 字节对齐，导致后续 Flash 写入操作失败或系统异常。此问题在掉电场景下有 **75% 的触发概率**。

---

2. 问题描述 (Problem Description)

2.1 问题现象

当系统在 Flash 页擦除过程中意外掉电，重新上电后：

1. 日志系统初始化可能成功（无错误返回）
2. 后续日志写入操作失败
3. 可能触发硬件异常 (HardFault)
4. 严重情况下系统无法正常启动

2.2 触发条件

条件	说明
必要条件	使用 EasyFlash 日志功能 (`EF_USING_LOG`)
必要条件	Flash 擦除单位为 Page 模式 (非整块擦除)
触发条件	Flash 擦除过程中发生掉电
概率	已擦除字节数非 4 的整数倍时触发 (约 75%)

2.3 根本原因

STM32 Flash 擦除机制为逐字节执行，掉电时可能导致部分字节已擦除 (0xFF) 而部分字节未擦除。EasyFlash 通过统计连续 0xFF 字节数 (`continue_ff`) 来计算下次写入地址，但未对结果进行 4 字节对齐处理：

```
擦除进度示意：
原始数据：[D0][D1][D2][D3][D4][D5][D6][D7][D8][D9]...
掉电位置：                                ↑ 擦除中断
擦除结果：[FF][FF][FF][FF][FF][FF][FF][??][??][??]...
          |---- 已擦除 7 字节 ----|
                ↑ continue_ff = 7 (非 4 的倍数)

计算的写入地址 = base_addr + sector_size - continue_ff - 4
              = 非对齐地址
```

---

3. 技术分析 (Technical Analysis)

3.1 问题代码定位

文件: `easyflash/src/ef_utils.c`
函数: `ef_find_sec_using_end_addr()`

```c
/**
 * Find the current flash sector using end address by continuous 0xFF.
 */
uint32_t ef_find_sec_using_end_addr(uint32_t addr, size_t sec_size) {
#define READ_BUF_SIZE 32

    uint32_t start = addr, continue_ff = 0, read_buf_size = 0, i;
    uint8_t buf[READ_BUF_SIZE];

    EF_ASSERT(READ_BUF_SIZE % 4 == 0);

    /* counts continuous 0xFF which is end of sector */
    while (start < addr + sec_size) {
        if (start + READ_BUF_SIZE < addr + sec_size) {
            read_buf_size = READ_BUF_SIZE;
        } else {
            read_buf_size = addr + sec_size - start;
        }
        ef_port_read(start, (uint32_t *)buf, read_buf_size);
        for (i = 0; i < read_buf_size; i++) {
            if (buf[i] == 0xFF) {
                continue_ff++;          // 🟢 字节级计数，可能非 4 的倍数
            } else {
                continue_ff = 0;
            }
        }
        start += read_buf_size;
    }

    /* calculate current flash sector using end address */
    if (continue_ff == sec_size) {
        return addr >= 4 ? addr - 4 : 0;
    } else if (continue_ff >= 4) {
        /* 🟢 问题所在：未对齐处理 */
        return (addr + sec_size - continue_ff) * 4 / 4 - 4;
        /*
         *  当 continue_ff = 5 时：
         *  返回值 = addr + sec_size - 9 (非 4 字节对齐)
         */
    } else {
        return addr + sec_size - 4;
    }
}
```

3.2 问题传播路径

```
┌─────────────────────────────────────────────────────────────┐
│ 上电流程                                                    │
└─────────────────────────────────────────────────────────────┘
        │
        ▼
┌─────────────────────────────────────────────────────────────┐
│ ef_log_init()                                               │
│   └→ find_start_and_end_addr()                              │
│         └→ ef_get_sector_status()                           │
│               └→ ef_find_sec_using_end_addr()               │
│                     └→ 返回非对齐地址 → log_end_addr        │
│   (此时无错误返回)                                          │
└─────────────────────────────────────────────────────────────┘
        │
        ▼
┌─────────────────────────────────────────────────────────────┐
│ 用户日志写入操作                                            │
│   └→ elog_flash_write()                                     │
│         └→ elog_flash_flush()                               │
│               └→ ef_log_write()                             │
│                     └→ ef_port_write(non_aligned_addr, ...) │
│                           └→ FLASH_ProgramWord()            │
│                                 └→ 硬件拒绝非对齐写入       │
└─────────────────────────────────────────────────────────────┘
        │
        ▼
┌─────────────────────────────────────────────────────────────┐
│ 失败模式:                                                   │
│   1. 写入静默失败 (数据未写入)                              │
│   2. 写入成功但位置错误                                     │
│   3. 硬件错误异常 (HardFault)                               │
│   4. 系统死锁                                               │
└─────────────────────────────────────────────────────────────┘
```

3.3 STM32 Flash 写入约束

STM32 Flash 控制器对写入操作有严格约束：

```
// STM32 参考手册 RM0008, Section 3.2.3 Flash programming
// "A write operation to the Flash memory is performed by word (4 bytes)"
// "The write address must be aligned to a word boundary (addr[1:0] = 00)"
```

地址尾数	二进制	对齐状态	写入结果
0x...00	00	对齐	成功
0x...04	00	对齐	成功
0x...08	00	对齐	成功
0x...0C	00	对齐	成功
0x...01	01	**非对齐**	失败/异常
0x...02	10	**非对齐**	失败/异常
0x...03	11	**非对齐**	失败/异常
0x...05	01	**非对齐**	失败/异常

3.4 概率分析

假设掉电时刻均匀分布在擦除过程中：

```
擦除页大小: 2048 字节 (STM32F107 大容量系列)
所有可能的 continue_ff 值: 0 ~ 2048

continue_ff % 4 的分布:
- continue_ff % 4 == 0: 概率 25% (地址对齐 ✓)
- continue_ff % 4 != 0: 概率 75% (地址非对齐 ✗)

触发故障概率: 约 75%
```

---

4. 复现步骤 (Reproduction Steps)

4.1 环境准备

1. 硬件: STM32F107VC 开发板
2. 软件: EasyFlash v3.3.0 + EasyLogger v2.2.0
3. 配置: 启用 `EF_USING_LOG` 和 `ELOG_FLASH_USING_BUF_MODE`

4.2 复现方法

方法 A: 硬件模拟

```
1. 在代码中添加擦除延时点:

   // ef_port.c
   EfErrCode ef_port_erase(uint32_t addr, size_t size) {
       // ... 原有代码 ...
       for (i = 0; i < erase_pages; i++) {
           flash_status = FLASH_ErasePage(addr + (PAGE_SIZE * i));

           // 添加: 在擦除后立即触发手动掉电
           if (i == 0) {
               volatile int wait = 1;
               while(wait);  // 此时手动断电
           }
       }
   }

2. 运行程序，在断点处手动断电
3. 重新上电，调用日志写入函数
4. 观察写入结果
```

方法 B: 软件模拟 (推荐)

```
// 测试代码
void test_alignment_bug(void) {
    // 模拟部分擦除场景
    // 在测试扇区写入非 0xFF 数据

    uint32_t test_addr = 0x08015000;  // 测试扇区起始
    uint32_t test_data[512];          // 2048 字节

    // 1. 模拟部分擦除 (前 7 字节为 0xFF, 其余为随机数据)
    memset((uint8_t*)test_data, 0xFF, 7);
    memset((uint8_t*)test_data + 7, 0x55, 2048 - 7);

    // 2. 写入测试数据 (需要先擦除)
    // ... Flash 写入代码 ...

    // 3. 调用 ef_find_sec_using_end_addr
    uint32_t result = ef_find_sec_using_end_addr(test_addr, 2048);

    // 4. 检查结果对齐性
    printf("Calculated end address: 0x%08X\n", result);
    printf("Alignment check: %s\n",
           (result % 4 == 0) ? "PASS" : "FAIL (non-aligned!)");

    // 预期输出: FAIL (non-aligned!)
    // 因为 continue_ff = 7, result = test_addr + 2048 - 7 - 4 = 非对齐地址
}
```

4.3 预期结果

```
Calculated end address: 0x080157F5
Alignment check: FAIL (non-aligned!)
```

---

5. 影响范围 (Impact Assessment)

5.1 影响硬件平台

平台	Flash 架构	受影响程度
STM32F10x 系列	Page 擦除 (1KB/2KB)	高
STM32F20x 系列	Sector 擦除 (16KB+)	中
STM32F4xx 系列	Sector 擦除	中
其他 Cortex-M	依赖 Flash 控制器	视情况
外部 SPI Flash	Sector 擦除	中

5.2 影响场景

应用场景	风险等级	说明
工业控制	严重	异常掉电常见，日志丢失影响故障诊断
汽车电子	严重	安全相关，需符合 ASIL 标准
消费电子	中等	掉电较少见，但影响用户体验
医疗设备	严重	关键数据完整性要求
IoT 设备	中等	远程设备难以现场修复

5.3 潜在后果

1. **数据丢失**: 日志写入失败，关键信息丢失
2. **系统异常**: 非对齐访问触发 HardFault
3. **调试困难**: 问题仅在上电后发生，难以定位
4. **产品召回**: 批量产品可能面临返修风险

---

6. 修复建议 (Proposed Solution)

6.1 方案 A: 修正地址计算函数 (推荐)

修改文件: `easyflash/src/ef_utils.c`

```c
uint32_t ef_find_sec_using_end_addr(uint32_t addr, size_t sec_size) {
#define READ_BUF_SIZE 32

    uint32_t start = addr, continue_ff = 0, read_buf_size = 0, i;
    uint8_t buf[READ_BUF_SIZE];

    EF_ASSERT(READ_BUF_SIZE % 4 == 0);
    EF_ASSERT(sec_size % 4 == 0);  /* 扇区大小必须 4 字节对齐 */

    /* counts continuous 0xFF which is end of sector */
    while (start < addr + sec_size) {
        if (start + READ_BUF_SIZE < addr + sec_size) {
            read_buf_size = READ_BUF_SIZE;
        } else {
            read_buf_size = addr + sec_size - start;
        }
        ef_port_read(start, (uint32_t *)buf, read_buf_size);
        for (i = 0; i < read_buf_size; i++) {
            if (buf[i] == 0xFF) {
                continue_ff++;
            } else {
                continue_ff = 0;
            }
        }
        start += read_buf_size;
    }

    /* calculate current flash sector using end address */
    if (continue_ff == sec_size) {
        /* all sector is 0xFF, so the sector is empty */
        return addr >= 4 ? addr - 4 : 0;
    } else if (continue_ff >= 4) {
        /*
         * Calculate base address and ensure 4-byte alignment
         * This handles partial erase caused by power failure
         */
        uint32_t base_addr = addr + sec_size - continue_ff - 4;

        /* ☑ FIX: Force word alignment by truncating down */
        base_addr = (base_addr / 4) * 4;

        return base_addr;
    } else {
        /* less than 4 continuous 0xFF, sector is full */
        return addr + sec_size - 4;
    }
}
```

修改说明:

1. 添加 `sec_size` 对齐断言
2. 在返回地址计算后强制 4 字节对齐
3. 使用向下取整方式，避免增加已使用空间

6.2 方案 B:添加防御性检查

修改文件: `easyflash/port/ef_port.c`

```c
EfErrCode ef_port_write(uint32_t addr, const uint32_t *buf, size_t size) {
    EfErrCode result = EF_NO_ERR;
    size_t i;
    uint32_t read_data;

    EF_ASSERT(size % 4 == 0);

    /* ☑ FIX: Add address alignment check */
    if (addr % 4 != 0) {
        EF_DEBUG("Warning: Write address 0x%08X is not word-aligned!\n", addr);
        /* Align address down to word boundary */
        addr = (addr / 4) * 4;
        EF_DEBUG("Adjusted to: 0x%08X\n", addr);
    }

    FLASH_Unlock();
    FLASH_ClearFlag(FLASH_FLAG_BSY | FLASH_FLAG_EOP | FLASH_FLAG_PGERR | FLASH_FLAG_WRPRTERR);
    for (i = 0; i < size; i += 4, buf++, addr += 4) {
        FLASH_ProgramWord(addr, *buf);
        read_data = *(uint32_t *)addr;
        if (read_data != *buf) {
            result = EF_WRITE_ERR;
            break;
        }
    }
    FLASH_Lock();

    return result;
}
```

6.3 方案 C: 启动时完整性检查

添加文件或函数:

```c
/**
 * @brief Check and recover log area from power failure corruption
 *
 * @return EF_NO_ERR if log area is valid, EF_ERASE_ERR if recovery was needed
 */
EfErrCode ef_log_check_and_recover(void) {
    /* Check log_start_addr alignment */
    if (log_start_addr % 4 != 0) {
        EF_DEBUG("Log start address misaligned: 0x%08X\n", log_start_addr);
        EF_DEBUG("Attempting recovery by cleaning log area...\n");
        return ef_log_clean();
    }

    /* Check log_end_addr alignment */
    if (log_end_addr % 4 != 0) {
        EF_DEBUG("Log end address misaligned: 0x%08X\n", log_end_addr);
        EF_DEBUG("Attempting recovery by cleaning log area...\n");
        return ef_log_clean();
    }

    return EF_NO_ERR;
}

/* Call this after ef_log_init() */
EfErrCode ef_log_init(void) {
    /* ... existing init code ... */

    /* Power failure recovery check */
    result = ef_log_check_and_recover();

    return result;
}
```

6.4 方案对比

方案	修改量	兼容性	有效性	副作用
A	小	高	根本解决	可能丢弃 1-3 字节数据
B	小	高	防御性	地址调整可能导致数据错位
C	中	高	恢复性	清空日志区，丢失历史数据

建议: 采用 **方案 A** 作为主修复，**方案 C** 作为可选的启动检查。

---

7. 测试验证 (Verification)

7.1 单元测试

```c
/* test_ef_alignment.c */

void test_ef_find_sec_using_end_addr_alignment(void) {
    uint32_t test_addr = 0x08015000;
    uint32_t result;

    /* Test case 1: All sector empty */
    /* Mock: continue_ff = sec_size */
    result = ef_find_sec_using_end_addr(test_addr, 2048);
    TEST_ASSERT_EQUAL_HEX32(test_addr - 4, result);
    TEST_ASSERT_TRUE(result % 4 == 0);

    /* Test case 2: Partial erase - 7 bytes (non-aligned) */
    /* This simulates power failure during erase */
    /* Mock: continue_ff = 7 */
    result = ef_find_sec_using_end_addr(test_addr, 2048);
    TEST_ASSERT_TRUE(result % 4 == 0);
    /* Expected: (test_addr + 2048 - 7 - 4) aligned down */
    TEST_ASSERT_EQUAL_HEX32(test_addr + 2048 - 8 - 4, result);

    /* Test case 3: Partial erase - 1 byte */
    /* Mock: continue_ff = 1 */
    result = ef_find_sec_using_end_addr(test_addr, 2048);
    TEST_ASSERT_TRUE(result % 4 == 0);

    /* Test case 4: Full sector (no 0xFF at end) */
    /* Mock: continue_ff = 0 */
    result = ef_find_sec_using_end_addr(test_addr, 2048);
    TEST_ASSERT_EQUAL_HEX32(test_addr + 2048 - 4, result);
    TEST_ASSERT_TRUE(result % 4 == 0);
}

void test_alignment_all_values(void) {
    /* Test all possible remainder values */
    uint32_t test_addr = 0x08015000;

    for (int ff_count = 0; ff_count <= 100; ff_count++) {
        /* Mock continue_ff = ff_count */
        uint32_t result = ef_find_sec_using_end_addr_mocked(test_addr, 2048, ff_count);

        char msg[64];
        snprintf(msg, sizeof(msg), "Failed at continue_ff=%d, result=0x%08X",
                 ff_count, result);
        TEST_ASSERT_TRUE_MESSAGE(result % 4 == 0, msg);
    }
}
```

7.2 集成测试

```
测试流程：
1. 清空日志区
2. 写入若干日志
3. 模拟部分擦除 （手动修改 Flash 内容）
4. 重新初始化日志系统
5. 执行日志写入操作
6. 验证写入成功
7. 验证历史日志可读
```

7.3 压力测试

```
测试条件：
- 循环次数：10000
- 每次循环：
  1. 写入随机大小日志
  2. 随机模拟部分擦除
  3. 重新初始化
  4. 验证日志写入

通过标准：无硬件异常，无数据损坏，无内存泄漏
```

---

8. 附录 (Appendix)

8.1 相关源文件列表

文件	路径	说明
ef_utils.c	easyflash/src/	问题所在文件
ef_log.c	easyflash/src/	日志功能实现
ef_port.c	easyflash/port/	平台移植层
ef_cfg.h	easyflash/inc/	配置文件

8.2 相关宏定义

```c
/* ef_cfg.h */
#define EF_USING_LOG                    /* 启用日志功能 */
#define EF_ERASE_MIN_SIZE    PAGE_SIZE  /* 擦除单位：1024 或 2048 */
#define LOG_AREA_SIZE        (10 * EF_ERASE_MIN_SIZE)  /* 日志区大小 */

/* elog_flash_cfg.h */
#define ELOG_FLASH_USING_BUF_MODE       /* 启用缓冲模式 */
#define ELOG_FLASH_BUF_SIZE    1024     /* 缓冲区大小 */
```

8.3 STM32 Flash 时序参数 (参考)

参数	STM32F107	单位
页擦除时间	20 - 40	ms
全片擦除时间	20 - 40	ms
字编程时间	30 - 70	μs
预取缓冲区使能	可选	-

8.4 参考文档

1. STM32F10x 参考手册 (RM0008), Section 3: Flash memory
2. EasyFlash 文档: https://github.com/armink/EasyFlash
3. EasyLogger 文档: https://github.com/armink/EasyLogger
4. AN2606: STM32 microcontroller system memory boot mode

---

9. 联系信息 (Contact)

报告提交者: [您的姓名/组织]
联系方式: [邮箱地址]
日期: 2024年

GitHub Issues 链接:

- EasyFlash: https://github.com/armink/EasyFlash/issues

- EasyLogger: https://github.com/armink/EasyLogger/issues

本报告使用 Markdown 格式编写，可通过 Pandoc 或类似工具转换为 PDF。

```
pandoc EasyFlash_Alignment_Bug_Report.md -o EasyFlash_Alignment_Bug_Report.pdf \
    --pdf-engine=xelatex \
    -V mainfont="SimSun" \
    -V geometry:margin=1in
```