

# FLARE-On 13 Challenge 3: FlareOn13.doc

By Ange Albertini (@corkami)

## Overview

In `flareon13.doc`, the `.doc` extension is misleading: the exact same bytes are simultaneously a PDF, a UDF disc image, a ZIP archive, a VHD disk image, and a DOS `.COM` executable. Running `file` reports `data` because every format magic sits at a non-standard offset, and the base filesystem is UDF so there is no `CD001` ISO9660 marker. Solving all six embedded stages unlocks the final flag:

`Ju$7_4_117713_7r34$ur3_HUN7@flare-on.com` ("Just a little treasure HUNT").

| Property | Value                                                                         |
|----------|-------------------------------------------------------------------------------|
| MD5      | <code>de0dfbf31ac55f4a6de56792bb152636</code>                                 |
| SHA-1    | <code>8f35d250c0c080abbecbd96a2b417a905cd76360</code>                         |
| SHA-256  | <code>21632dd113a3b4d4678100fb4c37476a1aa77135b5e129e25fca92fbd6f4521f</code> |
| Size     | <code>1,855,887 bytes</code>                                                  |

## Revisions

- **v1:** One true flag and 6 fake flags. Deemed too easy during initial review because automated tooling and AI can zip through it.
- **v2 (this version):** All six stage flags are real, more obfuscated, and all required to decrypt the final flag.

## Walkthrough

The final flag is **not stored anywhere** in cleartext. Six stage tokens are hidden across six distinct file format channels; you must recover all six and pass them (in any order) into the final Crystal gate binary (coded in the Crystal programming language), which validates each token and combines them to decrypt the final flag.

Miss any one channel and you get nothing.

## Six Treasure Flags

- **1. Modified EICAR DOS .COM Stub:** The file starts with the literal `EICAR` string. However, the stub has been modified to execute (it is clearly larger than a standard EICAR test file) and update the flag via alphanumeric DOS machine code. Run it or debug it (it performs a simple XOR patch), and the true treasure flag `FLARE-STANDARD-ANTIVIRUS-TEST-FILE!` becomes visible.
- **2. Encrypted PDF with EBCDIC Text:** The PDF contains hidden text written in EBCDIC via a missing `ToUnicode` map, causing `pdftotext` to fail. The text is also drawn invisibly (render mode `Tr 3`,

PDF-encrypted). Decrypt the PDF and inspect the object content to recover the treasure flag: `mainframe_ebcdic_ghost`.

- **3. Off-Page JBIG2 Image in PDF:** The PDF image objects contain a JBIG2 stream displayed outside the visible page crop box. Dumping the embedded images with `pdfimages` extracts it cleanly, revealing the treasure flag: `jbig2_fax_geometry`.
- **4. Legacy Reduced ZIP Archive:** The embedded ZIP archive uses legacy ZipCrypto encryption (password: `infected`) and the vintage `Reduce` compression method, which modern standard unzip utilities reject. Extracting it with a legacy-compatible utility such as `oldunzip` yields the treasure flag: `reduce_not_deflate`.
- **5. UDF Descriptor & Universal Mach-O:** The UDF filesystem contains a universal Mach-O binary bundling two executables (trivial to carve): one coded in Odin (using Odin's `SoA` layout) containing the 6th sub-flag, and the other coded in Crystal serving as the final gate. Visible in cleartext within the UDF volume properties is the 5th treasure flag: `udf_tagged_descriptor`.
- **6. Odin Mach-O Executable:** Run the carved Odin Mach-O binary (or trace its lightweight crypto routine) to display the 6th treasure flag beneath a flare animation: `Ød1n_373rn4l_f14r3` (encoded as UTF-8, where `Ø` is `C3 98`). To execute a macOS Mach-O binary without macOS installed, consider GitHub Actions, Qiling, Unicorn, or Darling.

## Final Flag

Run the final Crystal gate binary with all six recovered treasure flags as command-line parameters. The gate decryption relies on standard MD5, RC4, and SHA-256 primitives—this final step was not designed to be the difficult part; the multi-format polyglot treasure hunt is the focus. Executing the gate outputs the final flag: `Ju$7_4_117713_7r34$ur3_HUN7@flare-on.com`.