B4$1C F0R3NS1CS

-by: Bl4cKc34sEr

Archive files

Most CTF challenges are contained in a ZIP, 7z, RAR, TAR or TGZ file, but only in a forensics challenge will the archive container file be a part of the challenge itself. Usually the goal here is to extract a file from a damaged archive, or find data embedded somewhere in an unused field (common forensics challenge). ZIP is the most common in the real world, and the most common in CTFs.

There are a lot of command-line tools that can be used for ZIP files that will be useful to know about.

- **unzip** -- will often output helpful information on why a zip will not decompress.
- ***zipdetails -v*** – This will provide in-depth information on the values present in the various fields of the format.
- **Zipinfo--** lists information about the zip file's contents, without extracting it.
- `zip -F input.zip --out output.zip` and `zip -FF input.zip --out output.zip` attempt to repair a corrupted zip file.
- **Fcrackzip--** brute-force the zip password using wordlists (e.g rockyou.txt).

Image file format analysis

CTFs are supposed to be fun, and image files are good for containing the flags, so of course image files often appear in CTF challenges. Image file formats are ways that make it interesting analysis puzzles involving metadata fields, checksums, steganography, or visual data encoding schemes.

The easy initial analysis step is to check an image file's metadata fields with ***exiftool***. If an image file has been abused for a CTF, its EXIF might identify the original image dimensions, camera type, embedded thumbnail image, comments and copyright strings, GPS location coordinates, etc. There might be a gold mine of metadata, or there might be almost nothing in there.

The ***strings*** command can be used to extract all the strings in image files and hope you get your flag there. The strings command may give you the flag or may give you hint sometimes, So, always use the strings command on image files.

Packet Capture (PCAP) file analysis

Network traffic is stored and captured in a PCAP file (Packet capture), with a program like **tcpdump** or **Wireshark**. A popular CTF challenge is to provide a PCAP file representing some network traffic and challenge the player to recover/reconstitute a transferred file or transmitted secret. Complicating matters, the packets of interest are usually in an ocean of unrelated traffic, so analysis and filtering the data is also a job for the player.
For initial analysis, take a high-level view of the packets with Wireshark's statistics or conversations view, or its **capinfos** command. Wireshark, and its command-line version ***tshark***, both support the concept of using "filters," which, if you master the syntax, can quickly reduce the scope of your analysis. There is also an online service called ***PacketTotal*** where you can submit PCAP files up to 50MB, and graphically display some timelines of connections, and SSL metadata on the secure connections. And also it will highlight the file transfers and show you any ***"suspicious"*** activity. If you already know what you're searching for, you can do `grep`-style searching through packets using ***ngrep.***
Just as "file carving" refers to the identification and extraction of files embedded within files, "packet carving" is a term sometimes used to describe the extraction of files from a packet capture. There are expensive commercial tools for recovering files from captured packets, but one open-source alternative is the Xplico framework. Wireshark also has an "Export Objects" feature to extract data from the capture ***(e.g.,*** File -> Export Objects -> HTTP -> Save all). Beyond that, you can try ***tcpxtract, Network Miner, Foremost, or Snort.***

If trying to repair a damaged PCAP file, there is an online service for repairing PCAP files called **PCAPfix.**

A note about PCAP vs PCAPNG: there are two versions of the PCAP file format; PCAPNG is newer and not supported by all tools. You may need to convert a file from PCAPNG to PCAP using Wireshark or another compatible tool, in order to work with it in some other tools.

PDF file analysis

PDF is an extremely complicated document file format, with enough tricks and hiding places. This also makes it popular for CTF forensics challenges.

The PDF format is partially plain-text, like HTML, but with many binary "objects" in the contents. The binary objects can be compressed or even encrypted data, and

include content in scripting languages like JavaScript. To display the structure of a PDF, you can either browse it with a text editor, or open it with a PDF-aware file-format editor like Origami.

qpdf is one tool that can be useful for exploring a PDF and transforming or extracting information from it. Another is a framework in Ruby called ***Origami*.**

When exploring PDF content for hidden data, some of the hiding places to check include:

- non-visible layers
- Adobe's metadata format "XMP"
- the "incremental generation" feature of PDF wherein a previous version is retained but not visible to the user
- white text on a white background
- text behind images
- an image behind an overlapping image
- non-displayed comments

There are also several Python packages for working with the PDF file format, like ***PeepDF***, that enable you to write your own parsing scripts.

Video and Audio file analysis

Like image file formats, audio and video file trickery is a common theme in CTF forensics challenges not because hacking or data hiding ever happens this way in the real world, but just because audio and video is fun. As with image file formats, steganography might be used to embed a secret message in the content data, and again you should know to check the file metadata areas for clues. Your first step should be to take a look with the ***mediainfo*** tool (or **exiftool**) and identify the content type and look at its metadata.
Audacity is the premiere open-source audio file and waveform-viewing tool, and CTF challenge authors love to encode text into audio waveforms, which you can see using the spectrogram view (although a specialized tool called **Sonic Visualiser** is better for this task in particular). Audacity can also enable you to slow down, reverse, and do other manipulations that might reveal a hidden message if you suspect there is one (if you can hear garbled audio, interference, or static). ***Sox*** is another useful command-line tool for converting and manipulating audio files.

It's also common to check least-significant-bits (LSB) for a secret message. Most audio and video media formats use discrete (fixed-size) "chunks" so that they can be streamed; the LSBs of those chunks are a common place to smuggle some data without visibly affecting the file.

Other times, a message might be encoded into the audio as ***DTMF tones*** or Morse code. For these, try working with ***multimon-ng*** to decode them.

Video file formats are really container formats, that contain separate streams of both audio and video that are multiplexed together for playback. For analysing and manipulating video file formats, **ffmpeg** is recommended. **ffmpeg -i** gives initial analysis of the file content. It can also de-multiplex or playback the content streams. The power of ffmpeg is exposed to Python using ***ffmpy***.