PNG-Fuzzing with JQF

Paul Kalz, Marwin Linke, and Sebastian Schatz

Humboldt University of Berlin, Germany

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Background on the File Format PNG

1.1 Overview

TODO: Overview: Gebt einen kurzen Überblick über das ausgewählte Datenformat (Historie, Verwendungszweck,...)

History

Use Case

1.2 Input Specification

TODO: Input Specification: Beschreibt im Detail die Spezifikation des Dateiformats. Wie sind Dateien dieses Formats aufgebaut? Existiert eine formale Spezifikation? Wie ist eine Beispieldatei aufgebaut?

Specifications

Structure

1.3 Security

TODO: Security: Beschreibt mögliche Sicherheitslücken im Zusammenhang mit dem Datenformat. Geht dabei näher auf bereits existierende Fälle ein (case study), ggf. auch im Zusammenhang mit den von euch ausgewählten Tools (Bug-Tracker).

2 Implementation

2.1 Tools

JQF The coverage-guided testing platform for Java named JQF, developed by R. Padhye, C. Lemieux and K. Sen, is designed for *practitioners*, who want to find bugs in Java programs, as well as for *researchers*, who wish to implement new fuzzing algorithms [1]. For our project, we used JQF to implement a fuzzing generator based on the file format PNG, which can be directly used in JQF to run test drivers for matching libraries.

PNGJ As for the library to test, we chose PNGJ, which is a pure, open-source Java library for high-performance reading and writing of PNG images [2]. For our use case, mainly the reading capability of PNGJ was tested by the fuzzer.

Java Standard Library A large component of writing PNG files are the compression algorithms, CRC32 checksums and Adler32 checksums, which all can be found in `java.util.zip`, a Java standard package. The PNG fuzzer relies heavily on the correctness of those algorithms and therefore uses this library to ensure accurate outputs instead of implementing the algorithms itself.

2.2 Process

To successfully write a PNG fuzzer and test the library, a generator and driver class need to be implemented. The generator is a self-contained class, which is called by JQF and returns a data type. In our case, the generator returns the data type `PngData`, which contains a byte array that resembles a functional PNG file. Next, the fuzz driver receives the `PngData`, which in return is read by PNGJ. The fuzz driver itself can be designed flexibly to test multiple functionalities in a library. This process is automatically repeated many times; each time JQF randomizes the seed to guide the random outputs from the generator in a more beneficial direction.

3 Generator

3.1 Chunk Structure

This section will outline the structure of chunks, an important aspect of PNG files. The byte stream of PNG images can be broken down into chunks, where each one contains certain information and serves a concrete purpose. A chunk is marked by its uniform structure found in the header and trailer surrounding its content.

As shown in table 1, each chunk consists of 4 parts [3]. The *Chunk length* refers to a 4-byte unsigned integer giving the number of bytes in the content field of that chunk, it doesn't include the length of the chunk type or the CRC

Table 1. Byte strucure of a chunk. [3]

Chunk length	**Chunk type**	**Chunk data**	**CRC**
4 bytes	4 bytes	*Length* bytes	4 bytes

checksum at the end. The *Chunk type* is always represented by 4 characters each 1 byte long to uniquely identify each type of chunk. It uses capitalization to imply information about the chunk. The *Chunk content* is the main part of each chunk and contains various amounts of data. The *CRC* is a well-known checksum algorithm, which uses 32 bits and therefore is also called CRC32 here.

We encapsulated this common structure in the class called `ChunkBuilder`. After generating the contents of a chunk, we simply call:

```
ChunkBuilder.constructChunk(chunkType, chunkContent)
```

with the according type and content as byte arrays/streams and receive a complete chunk, which then can be concatenated with other chunks to form a PNG file.

3.2 PNG Structure

Next, the ordering of chunks will be explained here.

Parameters To decide which chunks should appear, each chunk refers to a parameter which is initialized in `initializeParameters`

Chunks Here will all implemented chunks be represented with a short categorization of use.

3.3 IHDR

A short explanation what the IHDR chunk does and why it is so important for fuzzing.

Image Information Also, there are parameters to store important information about the image itself, like the bit-depth, coloring, image size, compression and filter methods.

3.4 IDAT

A short explanation with focus on the compression method.

Image Data The uncompressed image data will be explained here with the focus on scanlines and filter bytes.

Filtering Filtering is explained, the reason and implementation.

3.5 Interlacing

What is interlacing and how is it implemented?

3.6 Optional Chunks

Short explanation how the rest of the generator looks like and the reason why we are not going into detail much further.

4 Fuzz Driver

TODO: Fuzz Driver: Beschreibt grob, wie ihr bei der Implementation der Test-Treibers vorgegangen seid und welche Funktionalitäten der Library mit eurem Treiber getestet werden.

4.1 Guidance

TODO: Guidance: Beschreibt eure Änderungen an der Suchstrategie von JQF. Geht dabei insbesondere auf die Motivation/Intuition eurer Ideen ein, d.h. weshalb ihr diese Änderungen für sinnvoll haltet.

5 Evaluation

TODO: Beschreibt die durchgeführten Experimente und deren Ergebnisse. Wie hoch war die erreichte Coverage? Konnten Bugs/Crashes gefunden werden? Wenn ja, welche?

5.1 Experiments

5.2 Results

6 Result Discussion

TODO: Versucht die Ergebnisse der Experimente zu interpretieren und zu erklären (discussion). Zieht Sie Folgerungen aus den Ergebnissen (conclusion). Beschreibt die nächsten Schritte, die durchgeführt werden müssten/könnten/sollten (future work).

6.1 Discussion

6.2 Conclusion

6.3 Future Work

References

1. Rohan Padhye, Caroline Lemieux, and Koushik Sen. 2019. JQF: Coverage-Guided Property-Based Testing in Java. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3293882.3339002
2. PNGJ GitHub-Page, https://github.com/leonbloy/pngj?tab=readme-ov-file
3. LibPng, PNG specification for file structure, http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html