<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Change</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2023-06-20</td>
<td>Final Report and Management Summary</td>
<td>MSc. H. Moesl, Dipl.-Ing. D. Gstir, R. Weinberger and D. Oberhollenzer</td>
</tr>
</tbody>
</table>
Contents

1 Executive Summary 4

2 Introduction 7
  2.1 Findings Overview .................................................. 8
  2.2 Scope ................................................................. 8
  2.3 Coverage ............................................................. 8
  2.4 Recommended Further Tests ...................................... 11

3 Rating Methodology for Security Vulnerabilities 12
  3.1 Common Weakness Enumeration ................................. 13

4 Results 14
  4.1 Findings .............................................................. 14
  4.2 Informational Notes ................................................ 21

5 About X41 D-Sec GmbH 23

A Appendix 25
  A.1 Fuzzing Harnesses .................................................. 25
Dashboard

Target
Customer: Open Source Technology Improvement Fund (OSTIF)
Name: libjpeg-turbo
Type: Source Code
Version: As deployed between 2023-04-17 and 2023-05-30

Engagement
Type: Source Code Audit
Engagement Effort: 25 person-days, 2023-04-17 to 2023-05-30
Total issues found: 2

Figure 1: Issue Overview (l: Severity, r: CWE Distribution)
1 Executive Summary

In May 2023, X41 D-Sec GmbH performed a Source Code Audit against libjpeg-turbo to identify vulnerabilities and weaknesses in the source code. The test was organized by the Open Source Technology Improvement Fund (OSTIF).\footnote{https://ostif.org}

A total of two vulnerabilities were discovered during the test by X41. None were rated as critical, none were classified as high severity, none as medium, and two as low. Additionally, one issue without a direct security impact was identified.

**Figure 1.1:** Issues and Severity

libjpeg-turbo is a JPEG image decoding software that uses SIMD instructions to accelerate baseline JPEG compression and decompression for various platforms, including x86 and ARM.

As it is common for compiled languages, a particular focus was placed on the identification of typical memory corruption vulnerabilities such as buffer overflows, information leakages, use-
after-frees. Moreover, the testing team dedicated substantial focus and effort into examining
the input validation associated with the various API calls, conducting variant analysis on pre-
vious bugs classified as security vulnerabilities, and generally reviewing memory management
practices.

In a source code audit, the testers receive all available information about the target. The test was
performed by four experienced security experts between 2023-04-17 and 2023-05-30.

As we conclude the audit process for the libjpeg-turbo library, it's important to highlight the
excellent condition this library is in. After rigorous examination by multiple auditors, all reviewing
the library independently, it is remarkable to note that very few areas of concern were identified.
This signifies the strong robustness of the code, its quality, and its adherence to best practices
with regard to secure programming principles.

This audit was conducted using different testing techniques and approaches, ensuring broad test-
ing coverage. In terms of dynamic testing, the team developed several fuzz testing harnesses.
Fuzz testing is, in general, essential for the overall security of the libjpeg-turbo project, especially
since it is implemented in C, which is often prone to memory corruption vulnerabilities. For the
purpose of this test, code coverage driven fuzz testing using AFL++ in combination with address
space sanitizers (such as ASAN) was performed. It is highly recommended to incorporate the de-
veloped harnesses into the libjpeg-turbo project and to maintain the high standards and ensure
that fuzzing remains an integral and fixed part of libjpeg-turbo, either using AFL++ or libFuzzer,
resulting in better testing coverage.

Furthermore, both manual and tool-driven static source code analyses was performed. The for-
mer allowed the team to have a thorough and detailed understanding of the code, where the
public facing API interface was under scrutiny concerning memory corruption flaws as well as
secure coding standards in general. The manual auditing process was assisted by the usage of
state-of-the-art tools CodeQL, which allows for advanced semantic code analysis including taint
tracking and data flow analysis, adding an extra layer of comprehension to this audit.

In total, X41 was able to identify three issues through fuzz testing:

1. tj3Decompress: Out-of-bounds read in 2:1 upsampling code
2. tj3Transform: Memory corruption
3. tj3Transform: Out-of-bounds write

Again, it must be reiterated that this assessment provided valuable insights into the security pos-
ture at the time of testing, but it is important to note that any code audit is unable to guarantee
that the software complex is free of additional bugs.

To sum up, the results of this audit speak volumes about the soundness of the libjpeg-turbo library.
It deserves high praise to find a library that stands up so well under such rigorous and diverse testing approaches. The combination of manual and automated testing methods, in conjunction with the use of tools like CodeQL, has ensured a level of scrutiny that leaves X41 confident in the resilience and reliability of libjpeg-turbo library. Nevertheless, due to the widespread usage of libjpeg-turbo, it is encouraged to perform recurring security audits, because new vulnerabilities may be introduced as more features are added and also changes within one part of the system may have unintentional security impact to other parts.
2 Introduction

The assessment comprised a security review of the *libjpeg-turbo* library, utilizing static source code analysis as well as dynamic testing using dedicated fuzz testing harnesses. The branch in scope for this inspection was the main branch with the commit id:

- 3a53627306233013dcec61a90f0e9ed302ea5156

The main objective of this security assessment was the identification of vulnerabilities within the *libjpeg-turbo* code base. It must be noted that no communication channel with the maintainer of the *libjpeg-turbo* was available. Therefore, the testing team was unable to define specific areas of interest upfront.

From a programming style and software design perspective the code and design is clean and very well written, with security in mind.
2.1 Findings Overview

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>SEVERITY</th>
<th>ID</th>
<th>REF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out of Bounds Write in tj3Transform() for Non-Resizeable, Pre-Allocated Buffers</td>
<td>LOW</td>
<td>LJPGT-PT-23-01</td>
<td>4.1.1</td>
</tr>
<tr>
<td>Out-of-bounds Read in 2:1 Upsampling Code</td>
<td>LOW</td>
<td>LJPGT-PT-23-02</td>
<td>4.1.2</td>
</tr>
<tr>
<td>Memory Corruption in tj3Transform()</td>
<td>NONE</td>
<td>LJPGT-PT-23-100</td>
<td>4.2.1</td>
</tr>
</tbody>
</table>

Table 2.1: Security-Relevant Findings

2.2 Scope

Due to the absence of specifically outlined focus areas, the evaluation centered around a broad inspection, primarily looking for common memory corruption vulnerabilities.

2.3 Coverage

A security assessment attempts to find the most important or sometimes as many of the existing problems as possible, though it is practically never possible to rule out the possibility of additional weaknesses being found in the future.

A manual approach for code review is used by X41. This process was combined with fuzzing given the nature of libjpeg-turbo being exposed to parsing of potentially untrustworthy data.

The time allocated to X41 for this code review was sufficient to yield a reasonable coverage of the given scope.

2.3.1 Fuzzing

While conducting a source audit of the libjpeg-turbo library, it was found that some fuzz testing harnesses already exist and are part of OSS-Fuzz. Nevertheless, as not all API functions had undergone fuzzing via OSS-Fuzz, X41 made the decision to focus on conducting further fuzzing.

---

1 Application Programming Interface
For the fuzz testing, AFL++ was used in two ways/modes:

- argv (command-line) fuzzing of the cjpeg and djpeg tools utilizing AFL++ persistent mode
- Fuzzing of selected interesting looking functions of libjpeg-turbo utilizing AFL++ persistent mode

Persistent mode fuzzing is a feature in the AFL++ fuzzer that keeps the target program running in the background and continuously feeds it with new test cases. This is in contrast to the default "one-shot" mode in which the fuzzer launches the target program with each new test case. By utilizing this approach, it was possible to achieve execution speed improvements of 10 to 20 times. Moreover, AFL++ has recently incorporated support for command-line interface (CLI) fuzzing in persistent mode through the AFL_INIT_ARGV_PERSISTENT macro, rendering it an ideal choice for the CLI fuzzing of the cjpeg and djpeg CLI binaries.

2.3.1.1 Fuzzing Hardware

The fuzzing process was carried out on a system equipped with an AMD Ryzen Threadripper Processor, which boasts 64 cores and 128GB of RAM.

2.3.1.2 CLI Fuzzing

Considering that libjpeg-turbo comprises various tools, such as cjpeg and djpeg, as a component of its code base, X41, decided to conduct command-line interface (CLI) fuzzing against these tools to detect any bugs associated with the parsing of argv parameters.

To conduct CLI fuzzing on each of the aforementioned tools, X41 generated a valid set of CLI parameters and utilized them as input test cases for the fuzzer. Additionally, X41 built the code base with address sanitization enabled (-fsanitize=address) to detect any memory management errors. To facilitate the fuzzer in quickly finding valid CLI parameters, X41 configured the AFL++ compiler to create a dictionary using the AFL_LLVM_DICT2FILE flag based on the compiled C code.

The fuzzer executed each of the aforementioned tools for a total of approximately 5 billion times. Despite the extensive number of executions, X41 was unable to identify any immediate crashes. Hence, it can be concluded that the code base for these tools and libjpeg-turbo is well tested.

---

2 https://github.com/AFLplusplus/AFLplusplus/
3 Command-line Interface
4 Advanced Micro Devices
5 Random Access Memory
and written with security in mind, at least under the conditions and parameters we used for the fuzzing process.

However, it is worth noting that the absence of immediate crashes does not necessarily imply that the code is free from bugs or vulnerabilities.

### 2.3.1.3 Fuzzing of Selected Functions

During this project, X41 created fuzzing harnesses for the following functions:

- functions doing compression:
  - `tj3Compress8()`
  - `tj3Compress12()`
  - `tj3Compress16()`

- functions doing JPEG decompression:
  - `tj3Decompress8()`
  - `tj3Decompress12()`
  - `tj3Decompress16()`

- functions doing JPEG transformations:
  - `tj3Transform()`

The folder `testimages` already contained input test cases provided by OSS-Fuzz, which were utilized as a starting point for generating test cases using the `radamsa` tool\(^7\). X41 compiled the source code base with address sanitization enabled (`-fsanitize=address`).

The fuzzing harnesses executed each of the aforementioned functions approximately 5 billion times, but no memory memory corruptions or crashes were detected during this process.

Fuzzing the function `tj3Decompress12()` resulted in an out-of-bounds read, which is documented in the finding 4.1.1. This crash has also been identified mid-project by an independent security researcher\(^8\).

Further, fuzzing of `tj3Transform()` revealed a memory corruption issue when executing JPEG transform operations. This is described in finding 4.2.1.

---

\(^6\) Joint Photographic Experts Group

\(^7\) [https://gitlab.com/akihe/radamsa](https://gitlab.com/akihe/radamsa)

\(^8\) [https://github.com/libjpeg-turbo/libjpeg-turbo/issues/690](https://github.com/libjpeg-turbo/libjpeg-turbo/issues/690)
2.4 Recommended Further Tests

X41 recommends to subject all newly developed code to regular source code audits. Given the complexity of the jpeg-turbo library, the code base would benefit from recurring security audits as changes within one part of the system may have unintentional security impact to other parts.
3 Rating Methodology for Security Vulnerabilities

Security vulnerabilities are given a purely technical rating by the testers as they are discovered during the test. Business factors and financial risks for Open Source Technology Improvement Fund (OSTIF) are beyond the scope of a code audit which focuses entirely on technical factors. Yet technical results from a code audit may be an integral part of a general risk assessment. A code audit is based on a limited time frame and only covers vulnerabilities and security issues which have been found in the given time, there is no claim for full coverage.

In total, five different ratings exist, which are as follows:

<table>
<thead>
<tr>
<th>Severity Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
<tr>
<td>Low</td>
</tr>
<tr>
<td>Medium</td>
</tr>
<tr>
<td>High</td>
</tr>
<tr>
<td>Critical</td>
</tr>
</tbody>
</table>

A low rating indicates that the vulnerability is either very hard for an attacker to exploit due to special circumstances, or that the impact of exploitation is limited, whereas findings with a medium rating are more likely to be exploited or have a higher impact. High and critical ratings are assigned when the testers deem the prerequisites realistic or trivial and the impact significant or very significant.

Findings with the rating ‘none’ are called informational findings and are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.
3.1 Common Weakness Enumeration

The CWE\textsuperscript{1} is a set of software weaknesses that allows the categorization of vulnerabilities and weaknesses in software. If applicable, X41 provides the CWE-ID for each vulnerability that is discovered during a test.

CWE is a very powerful method to categorize a vulnerability and to give general descriptions and solution advice on recurring vulnerability types. CWE is developed by MITRE\textsuperscript{2}. More information can be found on the CWE website at \url{https://cwe.mitre.org/}.

\textsuperscript{1}Common Weakness Enumeration
\textsuperscript{2}https://www.mitre.org
4 Results

This chapter describes the results of this test. The security-relevant findings are documented in Section 4.1. Additionally, findings without a direct security impact are documented in Section 4.2.

4.1 Findings

The following subsections describe findings with a direct security impact that were discovered during the test.
4.1.1 LJPGT-PT-23-01: Out of Bounds Write in tj3Transform() for Non-Resizeable, Pre-Allocated Buffers

Severity: LOW
CWE: 787 – Out-of-bounds Write
Affected Component: turbojpeg.c:tj3Transform()

4.1.1.1 Description

In tj3Transform(), if the buffer is pre-allocated by the user and the function is instructed to not resize it, the buffer size is overwritten with a maximum computed value using tj3JPEGBufSize():

```c
if (this->noRealloc) {
    alloc = FALSE; dstSizes[i] = tj3JPEGBufSize(w, h, this->subsamp);
}
```

Listing 4.1: Buffer Size Recomputed

If the image is, e.g., rotated by 90 or 270 degrees, width and height are to be swapped and tj3JPEGBufSize() returns different results for transposed images.

The existing fuzzing code adheres this as well:

```c
transforms[2].op = TJXOP_ROT90;
transforms[2].options = TJXOPT_TRIM | TJXOPT_COPYNONE | TJXOPT_ARITHMETIC;
dstBufs[2] = (unsigned char *)malloc(tj3JPEGBufSize(height, width, jpegSubsamp));
```

Listing 4.2: Fuzzing Code in transform.cc

As a result, the assumed buffer size differs from the real size, leading to out-of-bounds writes when generating the transformed image.
This also affects code that uses a pre-allocated buffer that is too small for any other reason, as the buffer size passed to the function is ignored and replaced with an assumed maximum:

```c
#include <turbojpeg.h>
#include <stdlib.h>
#include <stdint.h>
#include <assert.h>
#include <stdio.h>

void do_test(unsigned char *data, size_t size)
{
    tjhandle handle = NULL;
    unsigned char *dstBuf[1] = {NULL};
    size_t dstSizes[1] = {0};
    size_t maxBufSize = 0;
    int width = 0, height = 0, jpegSubsamp;
    tjtransform transform = {0};

    handle = tj3Init(TJINIT_TRANSFORM);
    assert(handle != NULL);

    assert(tj3DecompressHeader(handle, data, size) == 0);

    width = tj3Get(handle, TJPARAM_JPEGWIDTH);
    height = tj3Get(handle, TJPARAM_JPEGHEIGHT);
    jpegSubsamp = tj3Get(handle, TJPARAM_SUBSAMP);

    assert(width >= 1);
    assert(height >= 1);
    assert(jpegSubsamp < TJ_NUMSAMP);
    assert((uint64_t)width * height < 1048576);

    transform.op = TJXOP_ROT90;

    maxBufSize = tj3JPEGBufSize(height, width, jpegSubsamp);
    assert(maxBufSize > 0);

    dstBuf[0] = malloc(maxBufSize);
    assert(dstBuf[0] != NULL);

    //dstSizes[0] = maxBufSize;
    assert(tj3Set(handle, TJPARAM_NOREALLOC, 1) == 0);

    tj3Transform(handle, data, size, 1, dstBuf, dstSizes, &transform);

    free(dstBuf[0]);
    tj3Destroy(handle);
}

int main(int argc, char **argv)
```
Listing 4.3: Reproducer Code for the tj3Transform() Crash

The input file used by the reproducer code in listing 4.3 is contained in the appendix (see section A.1).

4.1.1.2 Solution Advice

X41 recommends to either take the transformation into account when computing the buffer size, or simply refrain from overwriting the buffer size argument, and require it to be non-zero if the TJFLAG_NOREALLOC flag is set.
4.1.2  LJPGT-PT-23-02: Out-of-bounds Read in 2:1 Upsampling Code

Severity:  LOW
CWE:  125 – Out-of-bounds Read
Affected Component:  jdmrgext.c:h2v2_merged_upsample_internal()

4.1.2.1 Description

An out-of-bounds read error in function h2v2_merged_upsample_internal() was discovered by the testing harness, which can be triggered by malformed JPEG files. The fuzz testing produced three slightly different samples of lossless JPEG files (see appendix) which reproduce this issue using the djpeg CLI utility. When combined with code that prints the retrieved data, this can potentially cause an information leakage. However, due to time limitations, a thorough examination of whether this is feasible in this particular case was not conducted.

One can pertinently note that this issue was independently discovered and reported on GitHub by the user Shin-Yan.¹

The listing 4.4 shows the stack trace produced by samples 1 and 2:

```
$ ./djpeg-static -fast ../libjpeg-turbo-decompress/case1.jpg
P6
1 1
4095
Premature end of JPEG file
AddressSanitizer:DEADLYSIGNAL
=================================================================
==22626==ERROR: AddressSanitizer: SEGV on unknown address 0x62efffffa194
    (pc 0x55ed867b010d bp 0x7fffaf299d90 sp 0x7fffaf299cf0 T0)
==22626==The signal is caused by a READ memory access.
#0 0x55ed867b010d in h2v2_merged_upsample_internal
    /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmrgext.c
#1 0x55ed867b010d in h2v2_merged_upsample
    /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmerge.c:383:5
#2 0x55ed867a677f in merged_2v_upsample
    /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmerge.c:260:5
#3 0x55ed867a677f in merged_2v_upsample
    /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmerge.c:260:5
#4 0x55ed867b010d in h2v2_merged_upsample_internal
    /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmainct.c:317:3
#5 0x55ed867b010d in jpeg12_read_scanlines
    /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdapistd.c:335:3
```

¹ https://github.com/libjpeg-turbo/libjpeg-turbo/issues/690
Source Code Audit on libjpeg-turbo

Listing 4.4: Asan Output Produced by Sample 1 (Sample 2 Produces the Same Stack Trace)
Listing 4.5 shows the stack trace produced by sample 3:

```
$ ./djpeg-static -fast ../../libjpeg-turbo-decompress/case3.jpg
P6
64 1
4095
Premature end of JPEG file
AddressSanitizer:DEADLYSIGNAL
=================================================================
==9800==ERROR: AddressSanitizer: SEGV on unknown address 0x62efffffa194
   (pc 0x5606f23ccdcd bp 0x7ffdd6fbec70 sp 0x7ffdd6fbeb0 T0)
   ==9800==The signal is caused by a READ memory access.
   #0 0x5606f23ccdcd in h2v2_merged_upsample_internal
      /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmrgext.c
   #1 0x5606f23ccdcd in h2v2_merged_upsample
      /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmerge.c:383:5
   #2 0x5606f23c877f in merged_2v_upsample
      /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmerge.c:260:5
   #3 0x5606f23a23d1 in process_data_simple_main
      /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmainct.c:317:3
   #4 0x5606f237d1a8 in jpeg12_read_scanlines
      /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdapistd.c:335:3
   #5 0x5606f2309b75 in main
      /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/djpeg.c:868:25
   #6 0x7fa7d7f9829c in __libc_start_main
      (/lib64/libc.so.6+0x3529c) (BuildId: c8417d767bacccfadb39b474e484d46947915cd8f)
   #7 0x5606f2246359 in _start
      /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S:120

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV /home/dags/work/rootsys/libjpeg-turbo_2023-05/libjpeg-turbo/jdmrgext.c
  in h2v2_merged_upsample_internal
   ==9800==ABORTING
```

Listing 4.5: Asan Output Produced by Sample 3

In all samples the faulty read occurs when accessing `cinfo->sample_range_limit` for the red color. The inputs leading to the issue and the AFL++ test harness can be found in the appendix (see section A.4).

4.1.2.2 Solution Advice

X41 recommends adding boundary checks to mitigate the risk of reading out-of-bounds.
4.2 Informational Notes

The following observations do not have a direct security impact, but are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

4.2.1 LJPGT-PT-23-100: Memory Corruption in tj3Transform()

Affected Component: turbojpeg.c:tj3Transform()

4.2.1.1 Note

This informational note was previously a finding. It was re-rated because the API was used incorrectly.

4.2.1.2 Description

In `tj3Transform()`, if the buffer is pre-allocated by the user and the function is instructed to not resize it, a memory corruption error is encountered.

This error occurs on any valid JPEG transform operation (except `TJXOP_NONE`) and results in the following GDB output:

```
  double free or corruption (out)
  Program received signal SIGABRT, Aborted.
  0x00007f617116ea7c in pthread_kill () from /lib/x86_64-linux-gnu/libc.so.6
  #0 0x00007f617116ea7c in pthread_kill () from /lib/x86_64-linux-gnu/libc.so.6
  #1 0x00007f617111a476 in raise () from /lib/x86_64-linux-gnu/libc.so.6
  #2 0x00007f61711007f3 in abort () from /lib/x86_64-linux-gnu/libc.so.6
  #3 0x00007f61711616f6 in ?? () from /lib/x86_64-linux-gnu/libc.so.6
  #4 0x00007f6171178d7c in ?? () from /lib/x86_64-linux-gnu/libc.so.6
  #5 0x00007f617117aef0 in ?? () from /lib/x86_64-linux-gnu/libc.so.6
  #6 0x00007f617117d4d3 in free () from /lib/x86_64-linux-gnu/libc.so.6
  #7 0x000055e705db99d1 in free_pool (cinfo=0x55e706f89a98, pool_id=<optimized out>) at /src/jmemmgr.c:1142
  at /src/jcomapi.c:44
  #8 0x000055e705d999fd in jpeg_abort (cinfo=0xe944) at /src/jcomapi.c:44
  #9 0x000055e705d7f7af6 in tj3Transform (handle=handle@entry=0x55e706f89890,
```

2 GNU Debugger
Listing 4.6: Crash Output

The crash was detected through the use of specially crafted fuzz testing harnesses, leveraging AFL++. However, due to the limited time scope of the evaluation, it wasn't possible to pinpoint the root cause of this crash.

The crashing input and the AFL++ test harness can be found in the appendix (see section A.2).

4.2.1.3 Solution Advice

X41 recommends resolving this memory corruption issue after conducting further analysis to identify the root cause of this issue.
5 About X41 D-Sec GmbH

X41 D-Sec GmbH is an expert provider for application security and penetration testing services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world-class security experts enables X41 D-Sec GmbH to perform premium security services.

X41 has the following references that show their experience in the field:

- Source code audit of the Git source code version control system\(^1\)
- Review of the Mozilla Firefox updater\(^2\)
- X41 Browser Security White Paper\(^3\)
- Review of Cryptographic Protocols (Wire)\(^4\)
- Identification of flaws in Fax Machines\(^5,6\)
- Smartcard Stack Fuzzing\(^7\)

The testers at X41 have extensive experience with penetration testing and red teaming exercises in complex environments. This includes enterprise environments with thousands of users and vendor infrastructures such as the Mozilla Firefox Updater (Balrog).

Fields of expertise in the area of application security encompass security-centered code reviews, binary reverse-engineering and vulnerability-discovery. Custom research and IT security consulting, as well as support services, are the core competencies of X41. The team has a strong technical background and performs security reviews of complex and high-profile applications such as Google Chrome and Microsoft Edge web browsers.

X41 D-Sec GmbH can be reached via https://x41-dsec.de or mailto:info@x41-dsec.de.

\(^1\) https://x41-dsec.de/security/research/news/2023/01/17/git-security-audit-ostif/
\(^3\) https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf
\(^4\) https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf
\(^5\) https://www.x41-dsec.de/lab/blog/fax/
\(^7\) https://www.x41-dsec.de/lab/blog/smartcards/
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD</td>
<td>Advanced Micro Devices</td>
<td>9</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
<td>8</td>
</tr>
<tr>
<td>CLI</td>
<td>Command-line Interface</td>
<td>9</td>
</tr>
<tr>
<td>CWE</td>
<td>Common Weakness Enumeration</td>
<td>13</td>
</tr>
<tr>
<td>GDB</td>
<td>GNU Debugger</td>
<td>21</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
<td>10</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
<td>9</td>
</tr>
</tbody>
</table>
Appendix

A.1 Fuzzing Harnesses

For additional coverage, this section provides all test harnesses used during the fuzzing campaign of the libjpeg-turbo library.

A.1.0.1 Fuzzing of tj3Transform() - Out of Bounds Write

```
00000000: ffd8 ffe0 0010 4a46 3232 3232 3232 3201 ........JF222222.
00000010: 0001 0000 ffd8 0043 0008 0606 0700 0004 ........C........
00000020: 0007 0709 0908 0a0c 140d 0c0b 0b0c 1912 ............
00000030: 130f ffd8 iaff ield ialc ic20 242e 2720 .............. $.
00000040: 2211 2933 3232 3331 3232 8032 3232 1f27 "...3231222.222.
00000050: 393d 383a 3c05 ffd8 0043 4501 0509 9=8.......C...
00000060: 090c 0b04 180d 0d18 3221 1c21 3232 6a25 ........2!22j%
00000070: 3232 3232 0c40 327f 1c28 3729 2c30 3134 2222222222222222.
00000080: 4434 3232 3232 3232 3203 0303 0303 f803 44222222222222.
00000090: 3232 3232 3232 3232 3232 3232 ffc9 ................
000000a0: 0001 0c00 0100 0d03 0112 0002 1101 0311 ................
000000b0: 01ff cc00 0a00 1010 0514 1011 05ff da00 ................
000000c0: 0c03 0100 0211 03eb 8a13 ffd8 ffe0 f100 ........................
000000d0: 4a46 3232 3232 3232 3201 0000 0000 ffd8 JF222222222222.
000000e0: 0004 0007 0709 0908 0a0c 140d 0c0b 0b0c ................
000000f0: 1912 130f ffd8 iaff ield ialc ic20 242e ...................
00000100: 2211 2933 3232 3331 3232 8032 3232 1f27 "...3231222.222.
00000110: 4434 3232 3232 3232 3203 0303 0303 f803 44222222222222.
00000120: 3232 3232 3232 3232 3232 3232 ffc9 ................
00000130: 0001 0c00 0100 0d03 0112 0002 1101 0311 ................
00000140: 01ff cc00 0a00 1010 0514 1011 05ff da00 ................
00000150: 0c03 0100 0211 03eb 8a13 ffd8 ffe0 f100 ........................
00000160: 4a46 3232 3232 3232 3201 0000 0000 ffd8 JF222222222222.
00000170: 0004 0007 0709 0908 0a0c 140d 0c0b 0b0c ................
00000180: 1912 130f ffd8 iaff ield ialc ic20 242e ...................
00000190: 2211 2933 3232 3331 3232 8032 3232 1f27 "...3231222.222.
000001a0: 4434 3232 3232 3232 3203 0303 0303 f803 44222222222222.
000001b0: 3232 3232 3232 3232 3232 3232 ffc9 ................
000001c0: 0001 0c00 0100 0d03 0112 0002 1101 0311 ................
000001d0: 01ff cc00 0a00 1010 0514 1011 05ff da00 ................
```

Listing A.1: Input File for tj3Transform() Out-of-Bounds Write Crash

A.1.0.2 Fuzzing of tj3Transform() - Memory Corruption

```c
#include <turbojpeg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <limits.h>
#include <unistd.h>
#include <sys/types.h>

/* this lets the source compile without afl-clang-fast/lto */
#ifndef __AFL_FUZZ_TESTCASE_LEN
    ssize_t fuzz_len;
    unsigned char fuzz_buf[1024000];
#define __AFL_FUZZ_TESTCASE_LEN fuzz_len
#define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
#define __AFL_FUZZ_INIT() void sync(void);
#define __AFL_LOOP(x) ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
#define __AFL_INIT() sync()
#endif

typedef struct TestTJParam {
    int index;
    char *name;
} TestTJParam;

static TestTJParam testParams[] = {
    {TJXOPT_PERFECT, "TJXOPT_PERFECT"},
    {TJXOPT_TRIM, "TJXOPT_TRIM"},
    {TJXOPT_CROP, "TJXOPT_CROP"},
    {TJXOPT_GRAY, "TJXOPT_GRAY"},
    {TJXOPT_NOOUTPUT, "TJXOPT_NOOUTPUT"},
    {TJXOPT_PROGRESSIVE, "TJXOPT_PROGRESSIVE"},
    {TJXOPT_COPYNONE, "TJXOPT_COPYNONE"},
    {TJXOPT_ARITHMETIC, "TJXOPT_ARITHMETIC"},
    {TJXOPT_OPTIMIZE, "TJXOPT_OPTIMIZE"}
};
static unsigned int testParamLen = sizeof(testParams)/sizeof(TestTJParam);
__AFL_FUZZ_INIT();
```
void do_test(unsigned char *data, size_t size) {
    tjhandle handle = NULL;
    unsigned char *dstBuf[1] = {NULL};
    size_t dstSizes[1] = {0};
    size_t maxBufSize = 0;
    int width = 0, height = 0, jpegSubsamp, i, t;
    tjtransform transform = {0};
    
    unsigned short transform_options = (data[0] << 8) | (data[1]);

    /* IMPORTANT NOTE: first 3 bytes of the test file contain transform options and operation! */
    int op = data[2] % TJ_NUMXOP;

    if ((handle = tj3Init(TJINIT_TRANSFORM)) == NULL) {
        goto bailout;
    }
    
    data += 3;
    size -= 3;

    if (tj3DecompressHeader(handle, data, size) < 0) {
        goto bailout;
    }

    width = tj3Get(handle, TJPARAM_JPEGWIDTH);
    height = tj3Get(handle, TJPARAM_JPEGHEIGHT);
    jpegSubsamp = tj3Get(handle, TJPARAM_SUBSAMP);

    /* Ignore 0-pixel images and images larger than 1 Megapixel. Casting width to (uint64_t) prevents integer overflow if width * height > INT_MAX. */
    if (width < 1 || height < 1 || (uint64_t)width * height > 1048576) {
        goto bailout;
    }

    tj3Set(handle, TJPARAM_SCANLIMIT, 500);

    if (((transform_options & TJXOPT_CROP) > 0)) {
        transform.r.w = (width + 1) / 2;
        transform.r.h = (height + 1) / 2;

        width = transform.r.w;
        height = transform.r.h;
    }

    transform.options = transform_options;
    transform.op = op;
maxBufSize = tj3JPEGBufSize(width, height, jpegSubsamp);
if (!maxBufSize)
    goto bailout;

if ((op & TJXOP_ROT90) > 0 || ((op & TJXOP_ROT270) > 0))
{
    dstBuf[0] = (unsigned char *)malloc(tj3JPEGBufSize(height, width, jpegSubsamp));
    if (!dstBuf[0]) {
        goto bailout;
    }
}
else
{
    dstBuf[0] = (unsigned char *)malloc(tj3JPEGBufSize(width, height, jpegSubsamp));
    if (!dstBuf[0]) {
        goto bailout;
    }
}

if (tj3Set(handle, TJPARAM_NOREALLOC, 1) < 0) {
    goto bailout;
}

if (tj3Transform(handle, data, size, 1, dstBuf, dstSizes, &transform) == 0)
{
    /* Touch all of the output pixels in order to catch uninitialized reads
     * when using MemorySanitizer. */
    int sum = 0;
    for (i = 0; i < dstSizes[0]; i++)
        sum += dstBuf[0][i];

    /* Prevent the code above from being optimized out. This test should
    * never be true, but the compiler doesn’t know that. */
    if (sum > 255 * maxBufSize) {
        goto bailout;
    }
}

bailout:
    free(dstBuf[0]);
    tj3Destroy(handle);
}

void make_testcase(const char *filepath)
{
    int options = TJXOPT_PROGRESSIVE | TJXOPT_COPYNONE;
    }
```c
unsigned char data[200 * 1024] = {0};
int nread = 0;

data[0] = (char)(options >> 8);
data[1] = (char)(options);
data[2] = TJXOP_NONE;

FILE *fp = fopen("testcase", "wb");
if (fp)
{
    fwrite(data, 1, 3, fp);

    FILE *fJpeg = fopen(filepath, "rb");
    if (fJpeg)
    {
        nread = fread(data + 3, 1, sizeof(data) - 3, fJpeg);
        fclose(fJpeg);

        fwrite(data + 3, 1, nread, fp);
        nread += 3;
    } else {
        fclose(fp);
        return;
    }

    fclose(fp);
}
do_test(data, nread);
```

```c
int main(int argc, char **argv)
{
    #ifdef MAKE_TESTCASE
    make_testcase(argv[1]);
    return 0;
    #endif

    #ifdef LOAD_TESTFILE
    FILE *fp = fopen(argv[1], "rb");
    if (fp)
    {
        unsigned char imgdata[200 * 1024];
        int nread = fread(imgdata, 1, sizeof(imgdata), fp);
        fclose(fp);

        if (nread < 4)
        ```
{   
    printf("not enough data for test!\n");
}

unsigned short transform_options = (imgdata[0] << 8) | (imgdata[1]);

int op = imgdata[2] % TJ_NUMXOP;

printf("Operation: %02x\n", op);
printf("TransformOptions: %02x\n", transform_options);

for (int i=0; i<testParamLen; i++)
{
    if (transform_options & testParams[i].index) {
        printf("Setting %s to 1\n", testParams[i].name);
    }
}

do_test(imgdata, nread);

return 0;
#elif TEST_WITH_PARAM
FILE *fp = fopen(argv[1], "rb");
if (fp)
{
    unsigned char data[200*1024] = {0};
    int nread = fread(data + 3, 1, sizeof(data) - 3, fp);
    fclose(fp);

    // test param: no transform options, OP: ROT180
    data[0] = 0;
    data[1] = 0;
    data[2] = TJXOP_ROT180;
    do_test(data, nread + 3);
}
return 0;
#else
ssize_t len;    /* how much input did we read? */
unsigned char *buf;    /* test case buffer pointer */
__AFL_INIT();
buf = __AFL_FUZZ_TESTCASE_BUF;    /* this must be assigned before __AFL_LOOP! */

while (__AFL_LOOP(UINT_MAX))
{
    len = __AFL_FUZZ_TESTCASE_LEN;

    //...
if (len < 4)
continue;

unsigned char *buf_new = buf;

do_test(buf_new, len);
}

return 0;

Listing A.2: tj3Transform() Memory Safety

A crashing test input (first 3 bytes contain transform_options and operation, in this case options=0 and op=6 (TXJOP_ROT180)) is shown below. The file can be loaded by setting #define LOAD_TESTFILE 1 in the test harness before compilation.
Source Code Audit on libjpeg-turbo

X41 D-Sec GmbH
Listing A.3: tj3Transform() Crash Input

A.1.0.3 Fuzzing of tj3Decompress()

```c
#include <turbojpeg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <limits.h>
#include <unistd.h>
#include <sys/types.h>

/* this lets the source compile without afl-clang-fast/lto */
#ifndef __AFL_FUZZ_TESTCASE_LEN

ssize_t fuzz_len;
unsigned char fuzz_buf[1024000];

#define __AFL_FUZZ_TESTCASE_LEN fuzz_len
#define __AFL_FUZZ_TESTCASE_BUF fuzz_buf
#define __AFL_FUZZ_INIT() void sync(void);
#define __AFL_LOOP(x) ((fuzz_len = read(0, fuzz_buf, sizeof(fuzz_buf))) > 0 ? 1 : 0)
#define __AFL_INIT() sync()
#endif

__AFL_FUZZ_INIT();

#define MINIMAL_TESTCASE_LEN 4

typedef struct TestTJParam
{
    int index;
    char *name;
} TestTJParam;

static TestTJParam testParams[] = {
    {TJPARAM_FASTUPSAMPLE, "TJPARAM_FASTUPSAMPLE"},
    {TJPARAM_LOSSLESST, "TJPARAM_LOSSLESST"},
    {TJPARAM_BOTTOMUP, "TJPARAM_BOTTOMUP"},
    {TJPARAM_FASTDCT, "TJPARAM_FASTDCT"},
    {TJPARAM_LOSSLESS, "TJPARAM_LOSSLESS"},
    {TJPARAM_OPTIMIZE, "TJPARAM_OPTIMIZE"},
    {TJPARAM_PROGRESSIVE, "TJPARAM_PROGRESSIVE"},
    {TJPARAM_ARITHMETIC, "TJPARAM_ARITHMETIC"},
    {TJPARAM_NOREALLOC, "TJPARAM_NOREALLOC"},
    {TJPARAM_RESTARTROWS, "TJPARAM_RESTARTROWS"};
```

D-Sec GmbH
PUBLIC
Page 40 of 51
static unsigned int testParamLen = sizeof(testParams)/sizeof(TestTJParam);

void do_test_with_params(const unsigned char *data, size_t size, enum TJPF tjpf, int fastupsample, int bottomup, int fastdct)
{
    tjhandle handle = NULL;
    void *dstBuf = NULL;
    int width = 0, height = 0, precision, sampleSize, pfi;

    char env[18] = "JSIMD_FORCENONE=1";

    /* The libjpeg-turbo SIMD extensions produce false positives with
       MemorySanitizer. */
    putenv(env);

    if ((handle = tj3Init(TJINIT_DECOMPRESS)) == NULL)
        goto bailout;

    if (tj3DecompressHeader(handle, data, size) < 0)
    {
        goto bailout;
    }

    width = tj3Get(handle, TJPARAM_JPEGWIDTH);
    height = tj3Get(handle, TJPARAM_JPEGHEIGHT);
    precision = tj3Get(handle, TJPARAM_PRECISION);
    sampleSize = (precision > 8 ? 2 : 1);

    /* Ignore 0-pixel images and images larger than 1 Megapixel, as Google's
       OSS-Fuzz target for libjpeg-turbo did. Casting width to (uint64_t)
       prevents integer overflow if width * height > INT_MAX. */
    if (width < 1 || height < 1 || (uint64_t)width * height > 1048576)
    {
        goto bailout;
    }

    tj3Set(handle, TJPARAM_SCA N LIMIT, 500);

    int w = width, h = height;
    int i;
    int64_t sum = 0;

    tj3Set(handle, TJPARAM_FASTUPSAMPLE, fastupsample);
    tj3Set(handle, TJPARAM_BOTTOMUP, bottomup);
    tj3Set(handle, TJPARAM_F ASTDCT, fastdct);

    if (!tj3Get(handle, TJPARAM_LO SSSL ESSE))
    {
        tjscalingfactor sf = {1, 2};
        tj3SetScalingFactor(handle, sf);
        w = TJSCALED(width, sf);
h = TJSCALED(height, sf);

if (w >= 97 && h >= 75)
{
    tjregion cr = (32, 16, 65, 59);
    tj3SetCroppingRegion(handle, cr);
}
else
    tj3SetCroppingRegion(handle, TJUNCROPPED);

if ((dstBuf = malloc(w * h * tjPixelSize[tjpf] * sampleSize)) == NULL)
    goto bailout;

if (precision == 8)
    {
    if (tj3Decompress8(handle, data, size, (unsigned char *)dstBuf, 0, tjpf) == 0)
    {
        /* Touch all of the output pixels in order to catch uninitialized reads
        when using MemorySanitizer. */
        for (i = 0; i < w * h * tjPixelSize[tjpf]; i++)
            sum += ((unsigned char *)dstBuf)[i];
    }
    else
        goto bailout;
    }
else if (precision == 12)
    {
    if (tj3Decompress12(handle, data, size, (short *)dstBuf, 0, tjpf) == 0)
    {
        /* Touch all of the output pixels in order to catch uninitialized reads
        when using MemorySanitizer. */
        for (i = 0; i < w * h * tjPixelSize[tjpf]; i++)
            sum += ((short *)dstBuf)[i];
    }
    else
        goto bailout;
    }
else
    {
    if (tj3Decompress16(handle, data, size, (unsigned short *)dstBuf, 0, tjpf) == 0)
    {
        /* Touch all of the output pixels in order to catch uninitialized reads
        when using MemorySanitizer. */
        for (i = 0; i < w * h * tjPixelSize[tjpf]; i++)
            sum += ((unsigned short *)dstBuf)[i];
    }
    else
        goto bailout;
    }
free(dstBuf);
    dstBuf = NULL;

bailout:
    free(dstBuf);
    tj3Destroy(handle);
    }

void do_test(unsigned char *data, size_t size)
{
    tjhandle handle = NULL;
    void *dstBuf = NULL;
    int width = 0, height = 0, precision, sampleSize, pfi;
    /* TJPF_RGB-TJPF_BGR share the same code paths, as do TJPF_RGBA-TJPF_ARGB and
       TJPF_RGBA-TJPF_ARGB. Thus, the pixel formats below should be the minimum
       necessary to achieve full coverage. */
    enum TJPF pixelFormats[4] =
        (TJPF_RGB, TJPF_BGRX, TJPF_GRAY, TJPF_CMYK);
    const char *pixelFormatStr[] =
        ("TJPF_RGB", "TJPF_BGRX", "TJPF_GRAY", "TJPF_CMYK");
    /*
       IMPORTANT NOTE: data[0] = pixel format, data[1],data[2] = TJPARAM_*
       */
    pfi = data[0] % 4;
    //printf("pfi: %s\n", pixelFormatStr[pfi]);
    unsigned short tjparams = 0;
    tjparams = data[1] << 8 | data[2];
    data += 3;
    size -= 3;
    if ((handle = tj3Init(TJINIT_DECOMPRESS)) == NULL)
        goto bailout;
    if (tj3DecompressHeader(handle, data, size) < 0)
        goto bailout;
    width = tj3Get(handle, TJPARAM_JPEGWIDTH);
    height = tj3Get(handle, TJPARAM_JPEGHEIGHT);
    precision = tj3Get(handle, TJPARAM_PRECISION);
    sampleSize = (precision > 8 ? 2 : 1);
    /* Ignore 0-pixel images and images larger than 1 Megapixel, as Google's
       OSS-Fuzz target for libjpeg-turbo did. Casting width to (uint64_t)
       prevents integer overflow if width * height > INT_MAX. */
    }
if (width < 1 || height < 1 || (uint64_t)width * height > 1048576) {
    goto bailout;
}

tj3Set(handle, TJPARAM_SCANLIMIT, 500);

int w = width, h = height;
int pf = pixelFormats[pfi], i;
int64_t sum = 0;

for (int i=0; i<testParamLen; i++)
{
    if (tjparams & (1 << i)) {
        printf("Setting %s to 1\n", testParams[i].name);
        tj3Set(handle, testParams[i].index, 1);
    }
}

if (!tj3Get(handle, TJPARAM_LOSSLESS)) {
    tjscalingfactor sf = {1, 2};
    tj3SetScalingFactor(handle, sf);
    w = TJSCALED(width, sf);
    h = TJSCALED(height, sf);
    if (w >= 97 && h >= 75) {
        tjregion cr = {32, 16, 65, 59};
        tj3SetCroppingRegion(handle, cr);
    } else
        tj3SetCroppingRegion(handle, TJUNCROPPED);
}

if ((dstBuf = malloc(w * h * tjPixelSize[pf] * sampleSize)) == NULL)
    goto bailout;

if (precision == 8)
{
    if (tj3Decompress8(handle, data, size, (unsigned char *)dstBuf, 0, pf) == 0)
    {
        /* Touch all of the output pixels in order to catch uninitialized reads
         * when using MemorySanitizer. */
        for (i = 0; i < w * h * tjPixelSize[pf]; i++)
            sum += (*((unsigned char *)dstBuf)[i];
    } else
        goto bailout;
}
else if (precision == 12)
Source Code Audit on libjpeg-turbo

```c
{ if (tj3Decompress12(handle, data, size, (short *)dstBuf, 0, pf) == 0)
    {
        /* Touch all of the output pixels in order to catch uninitialized reads
           when using MemorySanitizer. */
        for (i = 0; i < w * h * tjPixelSize[pf]; i++)
            sum += (short *)dstBuf[i];
    }
    else
        goto bailout;
}
else
{
    if (tj3Decompress16(handle, data, size, (unsigned short *)dstBuf, 0, pf) == 0)
    {
        /* Touch all of the output pixels in order to catch uninitialized reads
           when using MemorySanitizer. */
        for (i = 0; i < w * h * tjPixelSize[pf]; i++)
            sum += (unsigned short *)dstBuf[i];
    }
    else
        goto bailout;
}
}
free(dstBuf);
dstBuf = NULL;

bailout:
free(dstBuf);
tj3Destroy(handle);
}

void make_testimage(const char *path)
{
    tjhandle handle = NULL;
    void *dstBuf = NULL;
    int width = 0, height = 0, precision, sampleSize, pfi;
    /* TJPF_RGB-TJPF_BGR share the same code paths, as do TJPF_RGBX-TJPF_XRGB and
       TJPF_RGBA-TJPF_ARGB. Thus, the pixel formats below should be the minimum
       necessary to achieve full coverage. */
    enum TJPF pixelFormats[4] =
        {TJPF_RGB, TJPF_BGRX, TJPF_GRAY, TJPF_CMYK};

    /*
       tj3Set(handle, TJPARAM_BOTTOMUP, pfi == 0);
       tj3Set(handle, TJPARAM_FASTUPSAMPLE, pfi == 0);
       tj3Set(handle, TJPARAM_FASTDCT, pfi == 0);
    */
    unsigned short tjparams = (1 << 2) | (1 << 0) | (1 << 3);
    unsigned char data[3] = {0};
```
data[0] = TJPF_RGB;
data[1] = (char)tjparams >> 8;
data[2] = (char)tjparams;

FILE *fp = fopen("testcase", "wb");
if (fp)
{
    fwrite(data, 1, sizeof(data), fp);
}

FILE *fp2 = fopen(path, "rb");
if (fp2)
{
    unsigned char imgdata[200 * 1024];
    int nread = fread(imgdata, 1, sizeof(imgdata), fp2);
    fclose(fp2);
    int nwritten = fwrite(imgdata, 1, nread, fp);
    printf("Written bytes: %d\n", 5 + nwritten);
}
else
{
    printf("Error opening file!\n");
}
fclose(fp);

unsigned char imgdata[200 * 1024];
fp = fopen("testcase", "rb");
if (fp)
{
    int nread = fread(imgdata, 1, sizeof(imgdata), fp);
    fclose(fp);
    printf("All done. Testing it now...\n");
    do_test(imgdata, nread);
}

#endif MAKE_TESTIMAGE
#endif LOAD_TESTFILE
#endif TEST_WITH_PARAM

int main(int argc, char **argv)
{
    #ifdef MAKE_TESTIMAGE
    make_testimage(argv[1]);
    return 0;
    #elif LOAD_TESTFILE
    FILE *fp = fopen(argv[1], "rb");
    if (fp)
{  
    unsigned char imgdata[200 * 1024];
    int nread = fread(imgdata, 1, sizeof(imgdata), fp);
    fclose(fp);
    
    if (nread < 4) 
    { 
        printf("not enough data for test!\n"); 
    } 
    
    do_test(imgdata, nread);
    } 
    else 
    { 
        printf("Error opening file\n"); 
    } 
    
    return 0;

#elif TEST_WITH_PARAM 
    FILE *fp = fopen(argv[1], "rb");
    if (!fp) { 
        printf("Error opening file\n"); 
        return 1; 
    } 
    
    unsigned char data[200*1024] = {0};
    size_t size = fread(data, 1, sizeof(data), fp);
    fclose(fp);
    
    do_test_with_params(data, size, TJPF_RGB, 1, 0, 0);
    return 0;
#else 
    ssize_t len; /* how much input did we read? */
    unsigned char *buf; /* test case buffer pointer */

    __AFL_INIT();
    buf = __AFL_FUZZ_TESTCASE_BUF; // this must be assigned before __AFL_LOOP!

    while (__AFL_LOOP(UINT_MAX)) 
    { 
        len = __AFL_FUZZ_TESTCASE_LEN;
        if (len < MINIMAL_TESTCASE_LEN) 
            continue;

        unsigned char *buf_new = buf;
        do_test(buf_new, len);
    } 

    return 0;
}"
Listing A.4: tj3Decompress() Listing

Crashing test inputs (first 3 bytes contain pixel format, and decompress parameters) are provided below. For all of the 3 inputs, the following parameters are set:

- TJPARAM_FASTUPSAMPLE¯1
- TJPARAM_BOTTOMUP¯1
- TJPARAM_FASTDCT¯1

The test files can be loaded by setting #define LOAD_TESTFILE 1 in the test harness before compilation.

For the first test case, the pixel format is set to 0 (which equals TJPF_RGB):

```
00000000: 0000 0dff d8ff e000 104a 4649 3030 3030 ..........JFI000
20000010: 3030 3030 0100 00ff db00 4300 3030 3030 0000000000000000
40000020: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
60000030: 3030 3030 3030 3030 3030 3030 3030 3030 ffd8 0043 0000000000000000
80000040: 0109 0909 0300 3030 3030 3030 3030 3030 3030 3030 3030 3030
A0000050: 0101 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
C0000060: 0102 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
E0000070: 0103 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000080: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000090: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
000000a0: dfb0 c300 110c 0001 0001 0301 2200 0211 2............"
000000b0: 0103 1101 ffcc 00f4 0000 0000 0000 0000 0000 0000 0000
000000c0: 0101 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
000000d0: 0708 090a 00ff c400 b510 0002 0103 0302 0000000000000000
000000e0: 0403 0505 0404 0000 007d 2102 0300 0411 0000000000000000
000000f0: 0512 2131 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

X41 D-Sec GmbH
Source Code Audit on libjpeg-turbo

Listing A.5: TJPF_RGB Testase

For the second test case, the pixel format is set to the pixel format is set to 0 (which equals TJPF_BGRX).

```
| 00000000: 0100 0dff d8ff e000 104a 4649 3030 3030 ..........JFII000 
| 00000010: 3030 3030 0100 00ff db00 4300 3030 3030 000000...C.0000 
| 00000020: 0000 03e8 7176 7879 7a83 8485 86ff ghijstuvvyzs.... 
| 00000030: f7f8 f9fa ffda 000c 0301 0002 1103 0001 ......... 
| 00000040: 0000 03e8 7176 7879 7a83 8485 86ff ghijstuvvyzs.... 
```

X41 D-Sec GmbH
PUBLIC
Listing A.6: TJPF_BGRX Testcase

For the third test case, the pixel format is again set to TJPF_RGB:

```
00000000: 0000 00ff d8ff e000 104a 4649 4600 0101 ...........JFIF...
00000010: 0e20 0100 0100 00ff db00 4300 0806 4a02 ........C...
00000020: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000030: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000040: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000050: 3030 3030 3030 3030 2e33 342 ffdb 0043 00000000.342....C
00000060: 0109 0909 0c0b 0c18 3030 3030 3030 3030 ........00000000
00000070: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000080: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000090: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
000000a0: 32ff c300 110c 0001 0040 0301 2200 0211 2........@.."...
000000b0: 0103 1101 ffc4 001f 0000 0105 0101 0101 ................
000000c0: 0101 0000 0000 0000 000f 0102 0000 0411 ................
000000d0: 0708 090a 0bff c400 b510 0002 0103 0302 ................
000000e0: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
000000f0: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000100: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000110: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000120: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000130: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000140: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000150: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000160: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000170: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000180: 3030 3030 3030 3030 f7f8 f9fa ffd4 0000000000000000
00000190: 0000 0301 0101 0101 0101 0101 0000 0000 ................
000001a0: 0001 0102 0304 0506 0708 090a 00ff c400 ............}
000001b0: b510 0002 0103 0302 0000 0000 0000 0000 ................
000001c0: 0777 0001 0203 1104 3030 3030 3030 3030 ......w............
000001d0: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
000001e0: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
000001f0: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000200: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000210: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000220: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000230: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000240: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000250: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000260: f7f8 f9fa ffd4 0000000000000000
00000270: 0001 0203 1104 0521 3106 1241 5107 6171 ..........!1..AQ.aq
00000280: 13b 3821 0814 4291 ab1 c109 2333 5200 ..........:.B......#3R.
00000290: 0000 03e8 ........................
```

Listing A.7: TJPF_RGB Testcase