LibreOffice oss-fuzz, crashtesting, coverity
Overview

- Oss-Fuzz
- Crashtesting
- Coverity
Oss-Fuzz
Overview

- Continuous Fuzzing of our import filters
- Thanks to Google we get to use their infrastructure and resources
Configuration

- Build remotely on google’s side
  - Calls our bin/oss-fuzz-build.sh
- 45 fuzzer targets in vcl/workben
- Each one is built with
  - libfuzzer + asan
  - libfuzzer + ubsan
  - afl + asan
  - honggfuzz + asan
  => 180 total
Configuration

- No dynamic libraries allowed
  - A serious pain for us
  - distro-configs/LibreOfficeOssFuzz.conf
  - Reuse --disable-dynamic-loading intended for iOS
  - Individual fuzzers are unfortunately v. large

- Run without config layer
  - Hardcoded suitable default for --enable-fuzzers
  - utl::ConfigManager::IsAvoidConfig()

- [https://dev-www.libreoffice.org/corpus/](https://dev-www.libreoffice.org/corpus/)
  - Contains our seed corpuses for 60 file formats
  - 15 are dtardons and co’s dlplib filters and are fuzzed separately
Oss-Fuzz Reports per Year

- Over 1100 issues over four years

- More than one a day in 2017 and 2018
- 113 this year to date, estimate 142 by end of year
  - This year's uptick due to a new route from sftfuzzer into old untested code
What a report looks like

**Crash State:** libreoffice_tiffuzer

**Crash Type:** Out-of-memory (exceeds 2048 MB)

**Issue:** 3566

**Created:** Thu, Oct 5, 2017, 6:56 PM

**Project:** libreoffice

**Fuzzer:** libFuzzer_libreoffice_tiffuzer

**Job Type:** libfuzzer_asan_libreoffice

**Platform:** linux

**Sanitizer:** address (ASAN)

**Security:** NO

**Reproducible:** YES

**Fixed:** NO

---

You can reproduce this crash painlessly with our reproduce tool. For Googlers, install the required libraries and run `prodaccess` & `google/data/ro/teams/clusterfuzz-tools/releases/clusterfuzz reproduce 6087102128193536`. For non-Googleers, see the installation section. Report any issues at `clusterfuzz-dev@chromium.org`.

---

**FIXED REVISION RANGE**

- NO (Last run with revision ad0e0cb25c7943663f6c946fd300b8f83e4ca59)

**REGRESSION REVISION RANGE**

- Libreoffice: 8abc7ba9784f7898576fbd7a48f0ff9e4445a68:4ac6a70524eb1383b8dab5e10bd3532434fe08a0

**CRASH STACKTRACE**

- ORIGINAL STACKTRACE ON **REVISION AD0E0CB25C7943663F6C946FD300BFF83E4CA59** (77 LINES)
### Sample ubsan bug

- **Buggy change, the unsigned short is promoted to int, undefined behavior in addition**

```c
- for ( sal_uLong i = 0; i < nColors; i++ )
+ for ( sal_uInt16 i = 0; i < nColors; i++ )
  {
    mpColorMap[ i ] = ( i << 16 ) + ( i << 8 ) + i;
  }
```

```
/srclibreoffice/filter/source/graphicsfilter/itga/itga.cxx:720:55: runtime error:
signed integer overflow: 2139160576 + 8356096 cannot be represented in type 'int'
 #0 0x2412e08 in (anonymous namespace)::TGARender::ImplReadPalette() /srclibreoffice/filter/source/graphicsfilter/itga/itga.cxx:720:55
 #1 0x240f06e in (anonymous namespace)::TGARender::ReadTGA(Graphic&) /srclibreoffice/filter/source/graphicsfilter/itga/itga.cxx:152:28
 #2 0x240ebf4 in itgGraphicImport /srclibreoffice/filter/source/graphicsfilter/itga/itga.cxx:788:23
```

- **Change back to a larger unsigned type**

```c
- for ( sal_uInt16 i = 0; i < nColors; i++ )
+ for ( sal_uInt32 i = 0; i < nColors; i++ )
  {
    mpColorMap[ i ] = ( i << 16 ) + ( i << 8 ) + i;
  }
```
Timeouts

- Sometimes timeout is genuine infinite loop
  - More often it’s just slow
- OssFuzz will report a maximum of one timeout per fuzzer
- Fix a timeout, another typically gets reported soon after
- Limit input size with a .options files
  ```
  [libfuzzer]
  max_len = 65536
  ```
- Some file formats have ~infinite decompression support
  - Tiny input can legitimately provide mega data to process
  - Examine FUZZ_MAX_INPUT_LEN (from .options) at runtime and limit to some factor of that
OOM

- Limit memory usage with
  
  - setenv("JPEGMEM", "768M", 1);
  - setenv("SC_MAX_MATRIX_ELEMENTS", "60000000", 1);
  - setenv("SC_NO_THREADED_CALCULATION", "1", 1);

- Pre-allocating buffers depending on potentially lying headers
  - Often a known relationship between remaining length of the file and the amount of data that it can produce
  - So short reads can be predicted before buffer allocation
    - GIF’s have a max compression of ~1:2560,
# Current Open Bugs

- **10 open bugs**
  - All Timeouts

<table>
<thead>
<tr>
<th>Star</th>
<th>Type</th>
<th>Status</th>
<th>Component</th>
<th>Created</th>
<th>Updated</th>
<th>Description</th>
<th>Owner</th>
<th>Category</th>
<th>Reproducible</th>
</tr>
</thead>
<tbody>
<tr>
<td>🌟 26099</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-10-02</td>
<td>----</td>
<td>libreoffice:htmlfuzzer: Timeout in htmlfuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 25667</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-09-14</td>
<td>----</td>
<td>libreoffice:ww2fuzzer: Timeout in ww2fuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 25520</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-09-09</td>
<td>----</td>
<td>libreoffice:xlsfuzzer: Timeout in xlsfuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 25005</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-08-18</td>
<td>----</td>
<td>libreoffice:hwpfuzzer: Timeout in hwpfuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 24932</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-08-15</td>
<td>----</td>
<td>libreoffice:scrtfuzzer: Timeout in scrtfuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 23602</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-06-20</td>
<td>----</td>
<td>libreoffice:ww8fuzzer: Timeout in ww8fuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 23523</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-06-17</td>
<td>----</td>
<td>libreoffice:ww6fuzzer: Timeout in ww6fuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 23492</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-06-16</td>
<td>----</td>
<td>libreoffice:fodpfuzzer: Timeout in fodpfuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 23503</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-06-16</td>
<td>----</td>
<td>libreoffice:lwpfuzzer: Timeout in lwpfuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
<tr>
<td>🌟 21753</td>
<td>Bug</td>
<td>New</td>
<td>libreoffice</td>
<td>2020-04-17</td>
<td>----</td>
<td>libreoffice:cgmfuzzer: Timeout in cmgfuzzer</td>
<td>ClusterFuzz</td>
<td>Reproducible</td>
<td></td>
</tr>
</tbody>
</table>
CrashTesting
Overview

- **Document Corpus**
  - Most scraped out of various bugzilla instances with `get-bugzilla-attachments-by-mimetype`
  - 116,200 files

- Import them all
  - With Markus Mohrhard’s `test-bugzilla-files`

- For many formats, then export to multiple formats
- Reimport exported output
- Report failed imports/exports
- Backtraces extracted from coredumps
New Setup

- New Hardware this year
  - Next day results
  - Vs ~3 days with old setup
- Thanks to Adfinis
12 Months of Importing

- Persistent <10 cluster of failures
12 Months of Exporting

- Large jumps as regressions detected and fixed
Coverity
Configuration

- Build locally with coverity’s tooling
- Outputs a big blob which we upload to their server which does the analysis
- https://scan.coverity.com/projects/libreoffice
- Project settings are open, no need to apply to be a “member” to see the findings
- Contemporary coverity scans both C++ and Java
- Coverity currently supports C++17, but not C++2a
  - patch configure.ac to not try c++2a for the coverity run
- We only scan LibreOffice, not dependencies
  - distro-configs/LibreOfficeCoverity.conf
  - So no ignored “external” category anymore
Example warning

- **uninit_member**
  - If a class initializes none of its members in its ctors there is no warning as its assumed to be intentional
  - If it initializes most of them, it warns about the uninitialized ones
  - A common mistake is with a class with multiple ctors, new member gets added and initialized in one ctor but not the other

```cpp
namespace sw
{
  AccessibilityIssue::AccessibilityIssue(sfx::AccessibilityIssueID eIssueID)
  : sfx::AccessibilityIssue(eIssueID)
  , m_eIssueObject(IssueObject::UNKNOWN)
  , m_pDoc(nullptr)
{

  2. **uninit_member**: Non-static class member m_pNode is not initialized in this constructor nor in any functions that it calls.

  4. **uninit_member**: Non-static class member m_nStart is not initialized in this constructor nor in any functions that it calls.

  ▶ CID 1458016 (#1 of 1): Uninitialized pointer field (UNINITCTOR)

  6. **uninit_member**: Non-static class member m_nEnd is not initialized in this constructor nor in any functions that it calls.
```
Pattern for waiving warnings

- An issue can be marked as a false positive or intentional in the web UI
  - But that only affects that coverity instance. Red Hat runs another one f.e.
  - If the code changes enough coverity will loose the ability to detect its the same code and reissue the warning

- INTENTIONAL pattern
  - // coverity[WARNING] - OPTIONAL_COMMENT
  - WARNING is the text before the : in the report

  ```
  // coverity[uninit_member] - members deliberately not initialized
  ScRawToken() {}
  ```

- FALSE POSITIVE pattern
  - // coverity[WARNING : FALSE] - OPTIONAL_COMMENT

  ```
  // coverity[copy_paste_error : FALSE] - posUB is correct
  if (posUB == mData.end())
  ```
Pattern to indicate program exit

- `// coverity[+kill]` indicates that the annotated function is intended to kill the program

```cpp
// coverity[+kill]
void Asserter::fail( std::string message,
                    const SourceLine &sourceLine )
```

- We use this in cppunit to indicate that Asserter::fail is intended to exit the program. In reality it throws a deliberately unhandled exception which would be warned about otherwise.

- Note that `–enable-assert-always-abort` is active for our coverity builds so failing asserts terminate program flow so coverity warnings about “impossible” situations are resolvable by adding appropriate asserts.
Tainted data

- Coverity detects common byteswapping techniques as indicating that data is probably untrusted tainted data.
- Very helpful for our general file format parsing, but not for our own legacy registry data format.
- `__coverity_tainted_data_sanitize__` can be used to sanitize the data.

```c
#if defined(__COVERITY__)
extern "C" void __coverity_tainted_data_sanitize__(void *);
#endif

sal_uInt16 MethodList::getMethodExcCount(sal_uInt16 index) const
{
    sal_uInt16 aCount = 0;

    if ((m_numOfEntries > 0) && (index <= m_numOfEntries))
    {
        try {
            aCount = readUINT16(m_pIndex[index] + calcMethodParamIndex);  
        } 
#if defined(__COVERITY__)  
        __coverity_tainted_data_sanitize__(&aCount);  
#endif
    }
```
Tainted data

- Validating untrusted data
  - A simple sanity test of tainted example

```c
sal_uInt32 nResourceLength(0);
m_rPSD.ReadUIInt32(nResourceLength);
if (nResourceLength > m_rPSD.remainingSize())
  return false;
```
Outstanding vs Fixed defects

Gap between requiring C++17 and coverity support
Coverity Stats 2020

Oct 11, 2020
Last Analyzed

6,139,868
Lines of Code Analyzed

0.00
Defect Density

Defect changes since previous build dated Oct 10, 2020

0
Newly detected

3
Eliminated

Defects by status for current build

26,222
Total defects

0
Outstanding

328
Dismissed

25,894
Fixed