

LibreOffice oss-fuzz, crashtesting, coverity

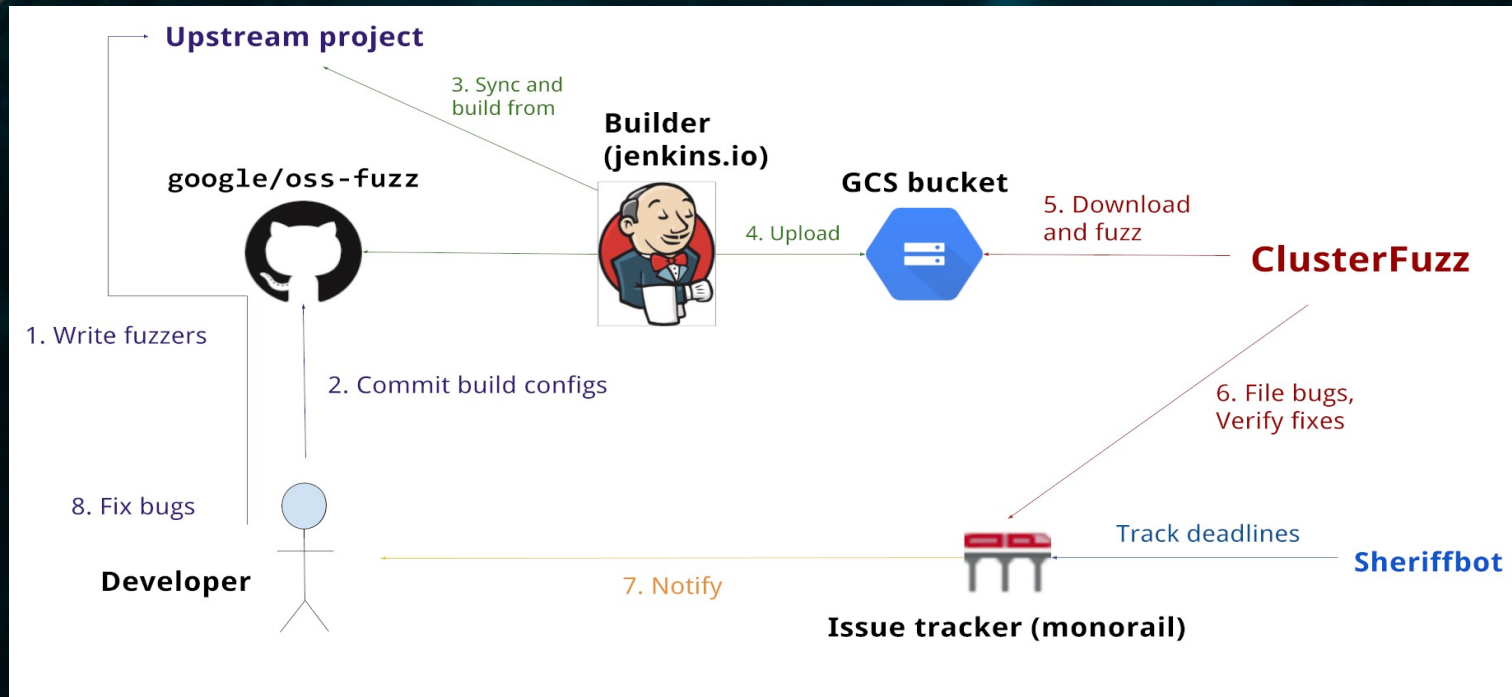
Overview

- 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

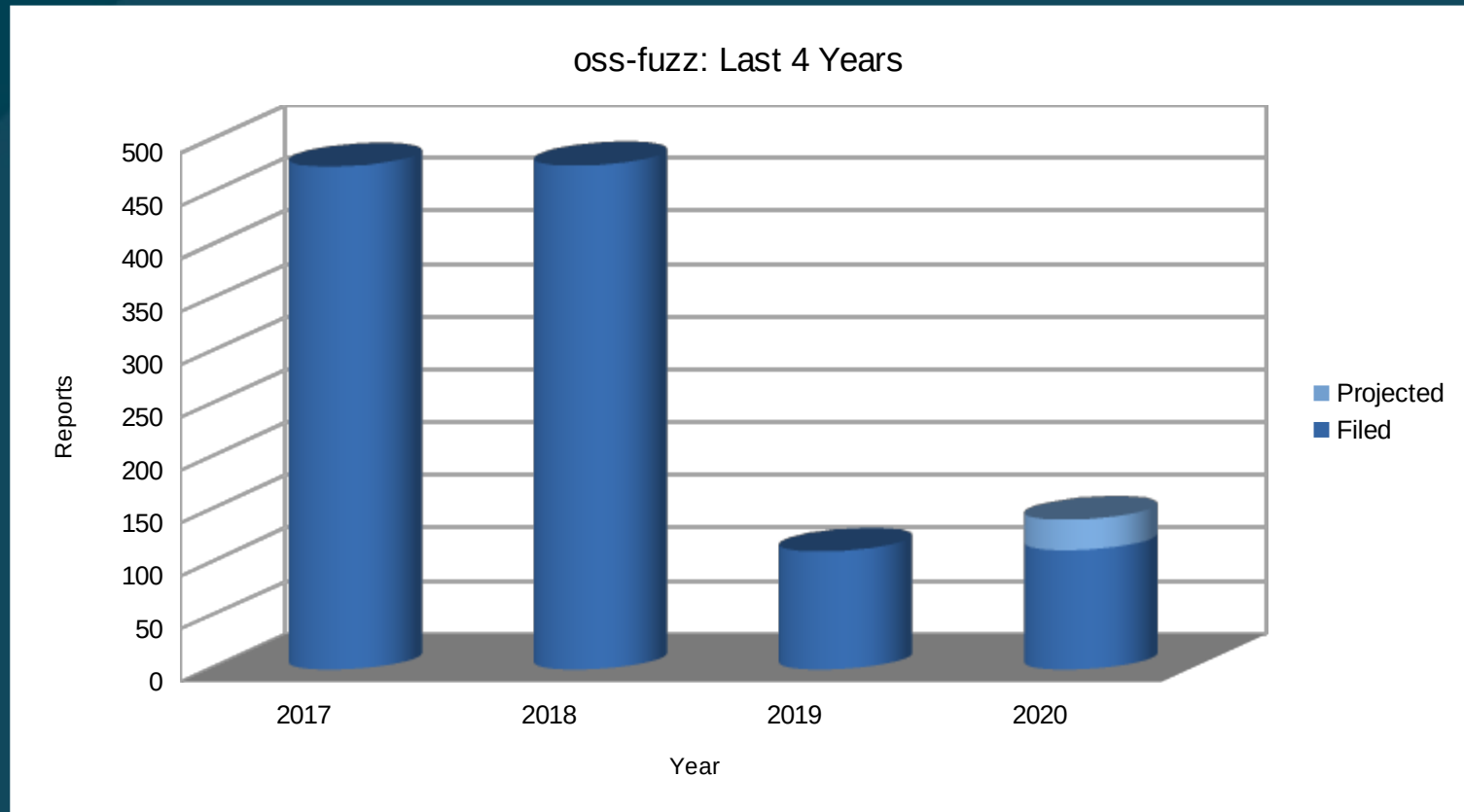
=> 180 total

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/>
 - 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 years uptick due to a new route from *sftfuzzer* into old untested code

What a report looks like

Crash State: libreoffice_tiffuzzer

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

Fuzzer: libFuzzer_libreoffice_tiffuzzer

Security: NO

Crash Address: ---

Job Type: libfuzzer_asan_libreoffice

Reproducible: YES

Issue: 3566

Platform: linux

Fixed: NO

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

Auto minimize

Project: libreoffice

Sanitizer: address (ASAN)

Minimized Testcase:  (210 B)

Unminimized Testcase:  (334 B)

Re-upload testcase: 

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-Googlers, see [the installation section](#). Report any issues at clusterfuzz-dev@chromium.org.


FIXED REVISION RANGE

NO (Last run with revision ad0e0cb25c7943663f6c946fd300bff83e4caa59)

REGRESSION REVISION RANGE

Libreoffice:
8abc7ba9784f7898576fbd7a48f0ff9e4445a68:4ac6a70524eb1383b8dab5e10bd3532434fe08a0

CRASH STACKTRACE

 ORIGINAL STACKTRACE ON **REVISION AD0E0CB25C7943663F6C946FD300BFF83E4CAA59** (77 LINES)

Sample ubsan bug

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

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

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

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

- Change back to a larger unsigned type

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

— 10 issues: -has:owner										
☆	26099	Bug	----	New	libreoffice	2020-10-02	----	libreoffice:htmlfuzzer: Timeout in htmlfuzzer	ClusterFuzz	Reproducible
☆	25667	Bug	----	New	libreoffice	2020-09-14	----	libreoffice:ww2fuzzer: Timeout in ww2fuzzer	ClusterFuzz	Reproducible
☆	25520	Bug	----	New	libreoffice	2020-09-09	----	libreoffice:xlsfuzzer: Timeout in xlsfuzzer	ClusterFuzz	Reproducible
☆	25005	Bug	----	New	libreoffice	2020-08-18	----	libreoffice:hwpfuzzer: Timeout in hwpfuzzer	ClusterFuzz	Reproducible
☆	24932	Bug	----	New	libreoffice	2020-08-15	----	libreoffice:scrtffuzzer: Timeout in scrtffuzzer	ClusterFuzz	Reproducible
☆	23602	Bug	----	New	libreoffice	2020-06-20	----	libreoffice:ww8fuzzer: Timeout in ww8fuzzer	ClusterFuzz	Reproducible
☆	23523	Bug	----	New	libreoffice	2020-06-17	----	libreoffice:ww6fuzzer: Timeout in ww6fuzzer	ClusterFuzz	Reproducible
☆	23492	Bug	----	New	libreoffice	2020-06-16	----	libreoffice:fodpfuzzer: Timeout in fodpfuzzer	ClusterFuzz	Reproducible
☆	23503	Bug	----	New	libreoffice	2020-06-16	----	libreoffice:lwpfuzzer: Timeout in lwpfuzzer	ClusterFuzz	Reproducible
☆	21753	Bug	----	New	libreoffice	2020-04-17	----	libreoffice:cgmfuzzer: Timeout in cgmfuzzer	ClusterFuzz	Reproducible

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

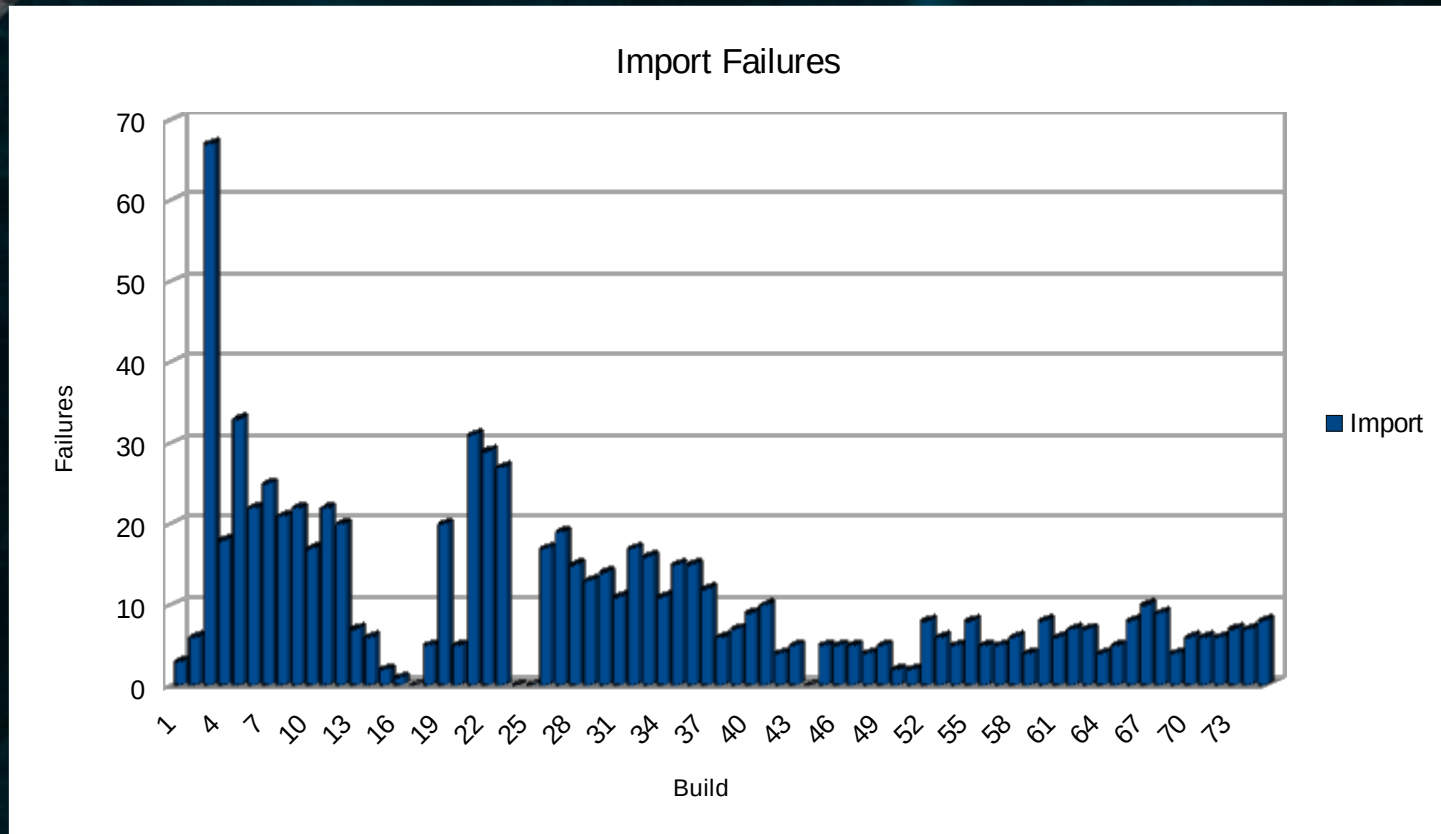
New Setup

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



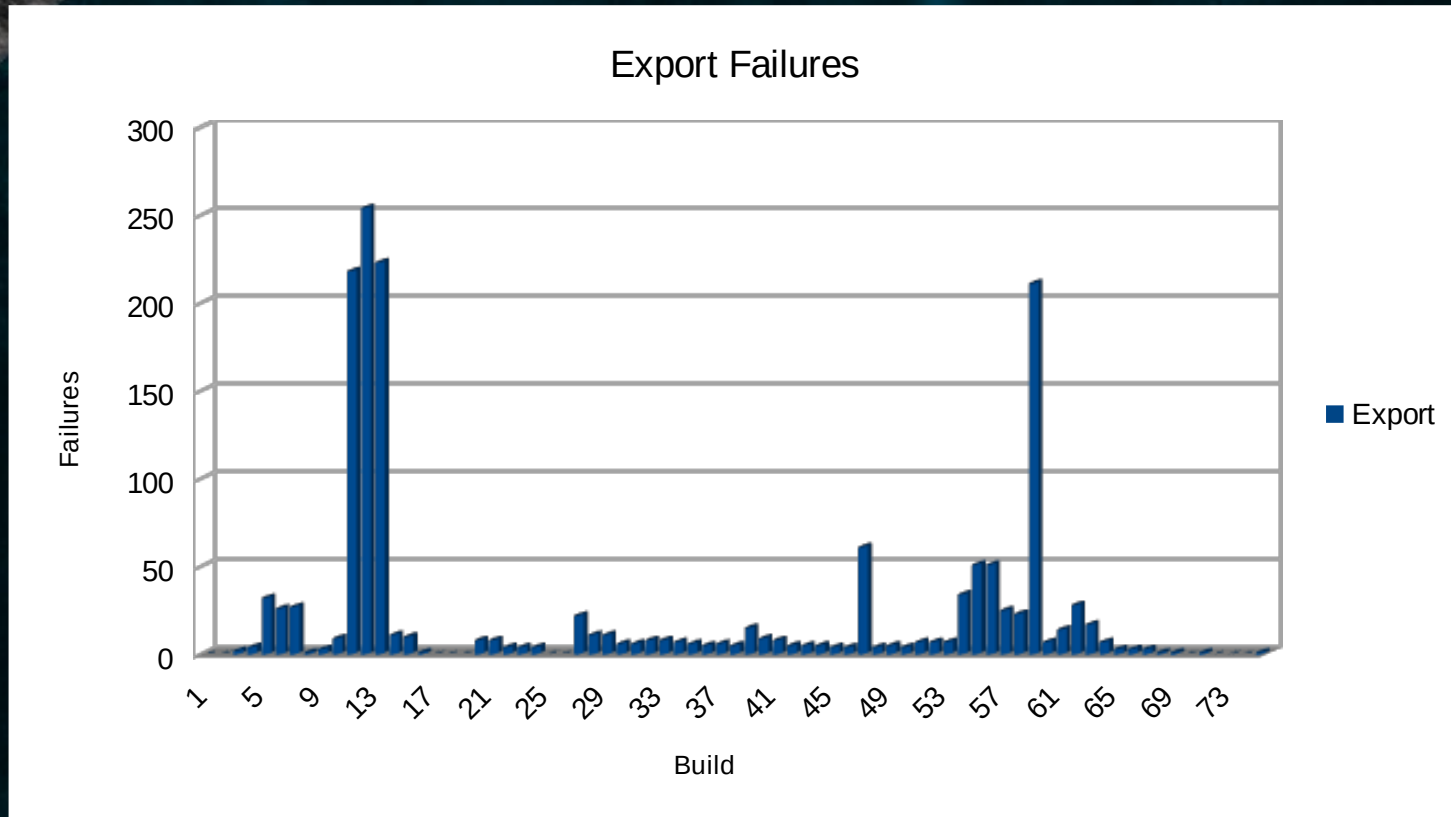
12 Months of Importing

- 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

```
15 namespace sw
16 {
17 AccessibilityIssue::AccessibilityIssue(sfx::AccessibilityIssueID eIssueID)
18     : sfx::AccessibilityIssue(eIssueID)
19     , m_eIssueObject( IssueObject::UNKNOWN)
20     , m_pDoc(nullptr)
21 {
22 }
```

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 (UNINIT_CTOR)

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

warning type

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

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

- We use this in cppunit to indicate that 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

```
#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
        }
    }
}

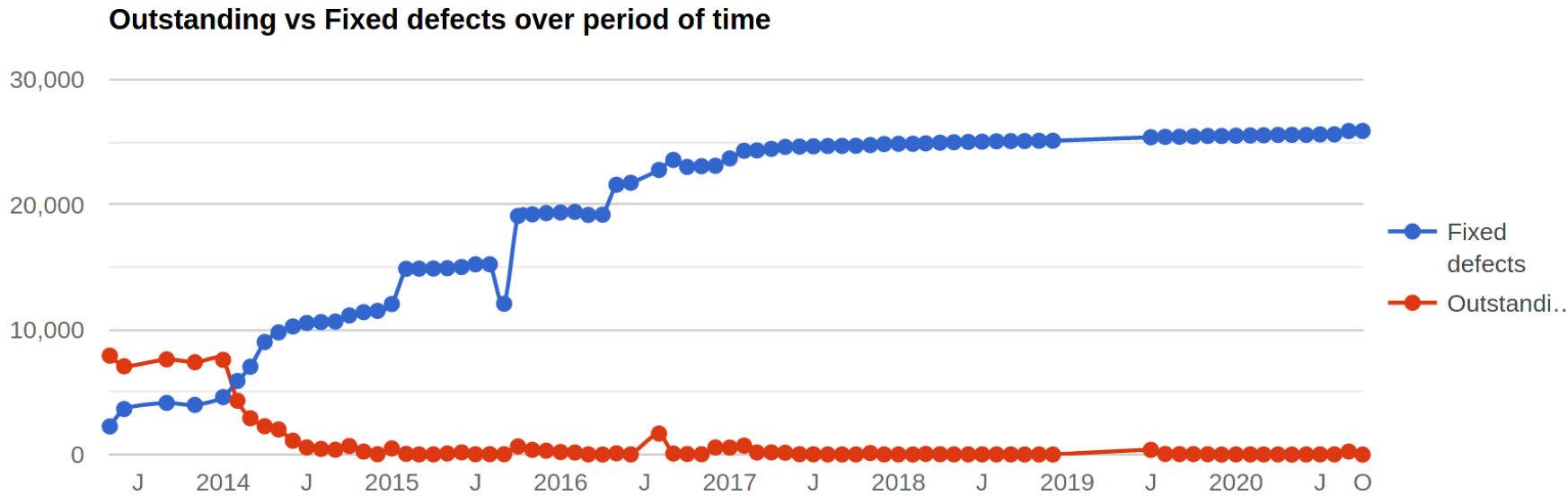
#endif
```


Tainted data

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

```
sal_uint32 nResourceLength(0);  
m_rPSD.ReadUInt32(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