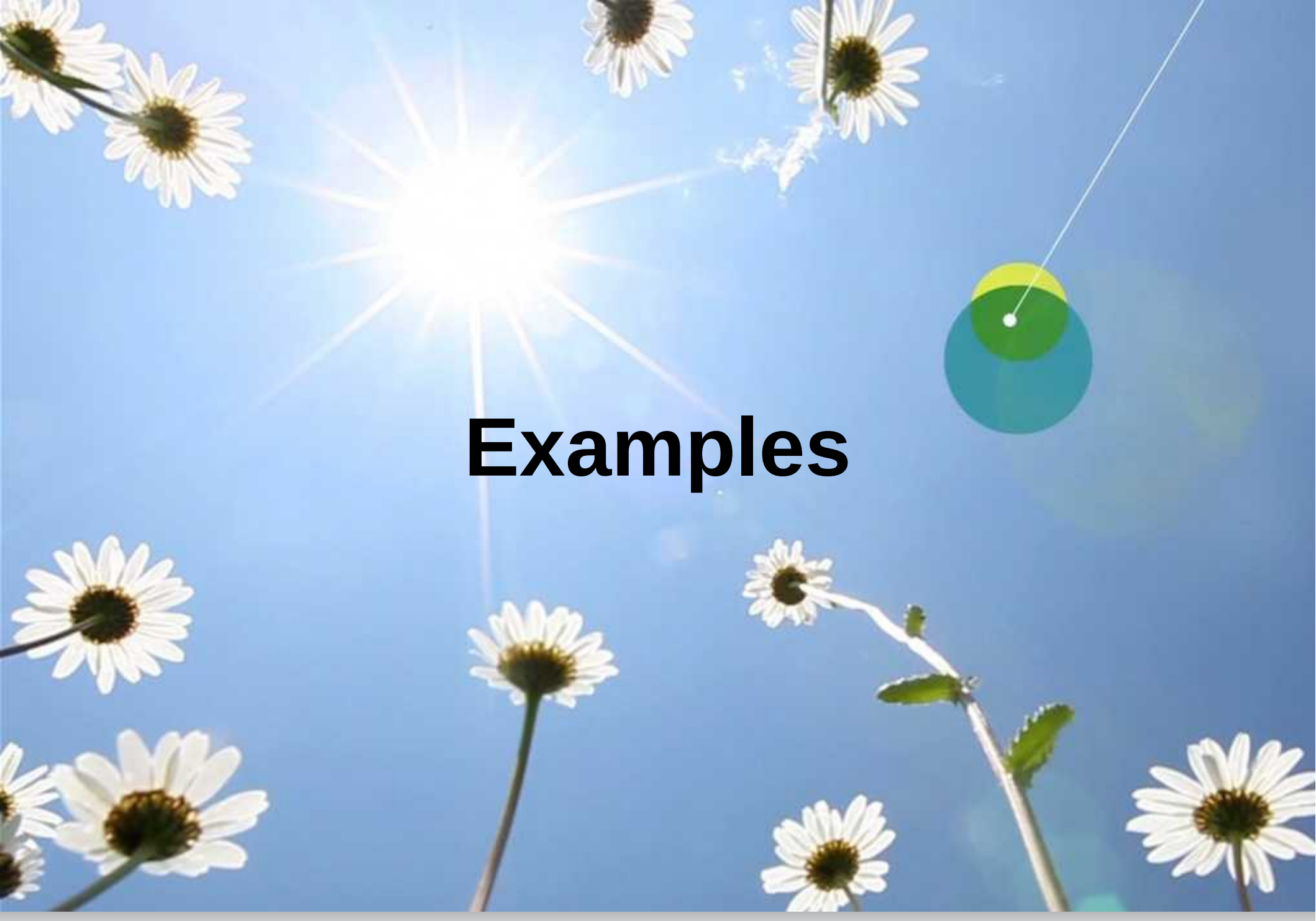
Crash Testing and Coverity

The Numbers

Caolán McNamara,
Red Hat
2015-09-25

- Coverity
  - Examples
  - Defect Density
  - Trends
- Crash Testing
  - Process
  - Trends


Examples

CID#707771 UNINIT_CTOR

```
2985        class SQLCommandPropertyUI : public ISQLCommandPropertyUI
2986        {
2987        protected:
2988            SQLCommandPropertyUI( const Reference< XPropertySet >& _rxObject )
2989                :m_xObject( _rxObject )
2990            {
    1. Condition "!this->m_xObject.is()", taking false branch
2991                if ( !m_xObject.is() )
2992                    throw NullPointerException();
    CID 707771 (#1 of 1): Uninitialized scalar field (UNINIT_CTOR)
    3. uninit_member: Non-static class member "m_refCount" is not initialized in this constructor nor in any functions that it calls.
2993            }
2994
2995            virtual oslInterlockedCount SAL_CALL acquire()
2996            {
2997                return osl_atomic_increment( &m_refCount );
2998            }
2999
3000            virtual oslInterlockedCount SAL_CALL release()
3001            {
3002                if ( 0 == osl_atomic_decrement( &m_refCount ) )
3003                {
3004                    delete this;
3005                    return 0;
3006                }
3007                return m_refCount;
3008            }
3009
3010        protected:
3011            Reference< XPropertySet >   m_xObject;
3012
3013        private:
    2. member_decl: Class member declaration for "m_refCount".
3014            oslInterlockedCount         m_refCount;
3015        };
```

CID#1209362 DEADCODE

```
55 bool ImplGetInvalidAsciiMultiByte(sal_uInt32 nFlags,
56                                   char * pBuf,
57                                   sal_Size nMaxLen)
58 {
59     if (nMaxLen == 0)
60         return false;
61     switch (nFlags & RTL_UNICODETOTEXT_FLAGS_UNDEFINED_MASK)
62     {
   CID 1209362 (#3 of 3): Logically dead code (DEADCODE) [select issue]
63     case RTL_UNICODETOTEXT_FLAGS_INVALID_0:
64         *pBuf = 0x00;
65         break;
66
   CID 1209362 (#2 of 3): Logically dead code (DEADCODE) [select issue]
67     case RTL_UNICODETOTEXT_FLAGS_INVALID_QUESTIONMARK:
68     default: /* RTL_UNICODETOTEXT_FLAGS_INVALID_DEFAULT */
69         *pBuf = 0x3F;
70         break;
71
   dead_error_condition: The switch value nFlags & 0xfU cannot be 80U.
   CID 1209362 (#1 of 3): Logically dead code (DEADCODE)
   dead_error_begin: Execution cannot reach this statement case 80U: .
72     case RTL_UNICODETOTEXT_FLAGS_INVALID_UNDERLINE:
73         *pBuf = 0x5F;
74         break;
75     }
76     return true;
77 }
```

Copy and Paste from previous ImplGetUndefinedAsciiMultiByte without corresponding change of UNDEFINED_MASK to INVALID_MASK

CID#983942 UNCAUGHT_EXCEPT

```
1037    // runtime adapter for lcl_UnoWrapFrame
        CID 983942 (#1 of 1): Uncaught exception (UNCAUGHT_EXCEPT)
        exn_spec_violation: An exception of type com::sun::star::uno::RuntimeException is thrown but the throw list
        throw(com::sun::star::uno::RuntimeException (*)()) doesn't allow it to be thrown. This will cause a call to
        unexpected() which usually calls terminate().
1038    static uno::Any lcl_UnoWrapFrame(SwFrmFmt* pFmt, FlyCntType eType) throw(uno::RuntimeException())
1039    {
1040        switch(eType)
1041        {
1042            case FLYCNTTYPE_FRM:
1043                return lcl_UnoWrapFrame<FLYCNTTYPE_FRM>(pFmt);
1044            case FLYCNTTYPE_GRF:
1045                return lcl_UnoWrapFrame<FLYCNTTYPE_GRF>(pFmt);
1046            case FLYCNTTYPE_OLE:
1047                return lcl_UnoWrapFrame<FLYCNTTYPE_OLE>(pFmt);
1048            default:
        exception_thrown: An exception of type com::sun::star::uno::RuntimeException is thrown.
1049                throw uno::RuntimeException();
1050        }
1051    }
```

That doesn't actually specify what it throws

CID#1158113 FORWARD_NULL

```
226 void DocumentLinkManager::disconnectDdeLinks()
227 {
        1. Condition "!this->mpImpl->mpLinkManager", taking false branch
228     if (!mpImpl->mpLinkManager)
229         return;
230
231     const sfx2::SvBaseLinks& rLinks = mpImpl->mpLinkManager->GetLinks();
        2. Condition "i < n", taking true branch
232     for (size_t i = 0, n = rLinks.size(); i < n; ++i)
233     {
234         ::sfx2::SvBaseLink* pBase = *rLinks[i];
235         ScDdeLink* pDdeLink = dynamic_cast<ScDdeLink*>(pBase);
        3. Condition "!pDdeLink", taking true branch
        4. var_compare_op: Comparing "pDdeLink" to null implies that "pDdeLink" might be null.
236         if (!pDdeLink)
        CID 1158113 (#1 of 1): Dereference after null check (FORWARD_NULL)
        5. var_deref_model: Passing null pointer "pDdeLink" to function "sfx2::SvBaseLink::Disconnect()", which dereferences it. [show details]
237             pDdeLink->Disconnect();
238     }
239 }
```

Somebody got confused on checking the result of dynamic_cast

CID#704127 CONSTANT_EXPRESSION_RESULT

```
7159 void WW8DopTypography::WriteToMem(sal_uInt8 *&pData) const
7160 {
7161     sal_uInt16 a16Bit = fKerningPunct;
7162     a16Bit |= (iJustification << 1) & 0x0006;
7163     a16Bit |= (iLevelOfKinsoku << 3) & 0x0018;
```

CID 704127 (#1 of 1): Wrong operator used (CONSTANT_EXPRESSION_RESULT)

operator_confusion: "(this->f2on1 << 5) & 2" is always 0 regardless of the values of its operands. This occurs as the bitwise operand of '|='. Did you intend to use right-shift ('>>') in "this->f2on1 << 5"?

```
7164     a16Bit |= (f2on1 << 5) & 0x002;
7165     a16Bit |= (reserved1 << 6) & 0x03C0;
7166     a16Bit |= (reserved2 << 10) & 0xFC00;
7167     Set_UInt16(pData,a16Bit);
7168
7169     Set_UInt16(pData,cchFollowingPunct);
7170     Set_UInt16(pData,cchLeadingPunct);
7171
7172     sal_Int16 i;
7173     for (i=0; i < nMaxFollowing; ++i)
7174         Set_UInt16(pData,rgxchFPunct[i]);
7175     for (i=0; i < nMaxLeading; ++i)
7176         Set_UInt16(pData,rgxchLPunct[i]);
7177 }
```

typo, should be 0x0020 not 0x002, wrong for 14 years

Defect Density

Open Source Defect Density

LibreOffice: 7,102,667 line of code and 0.00 defect density

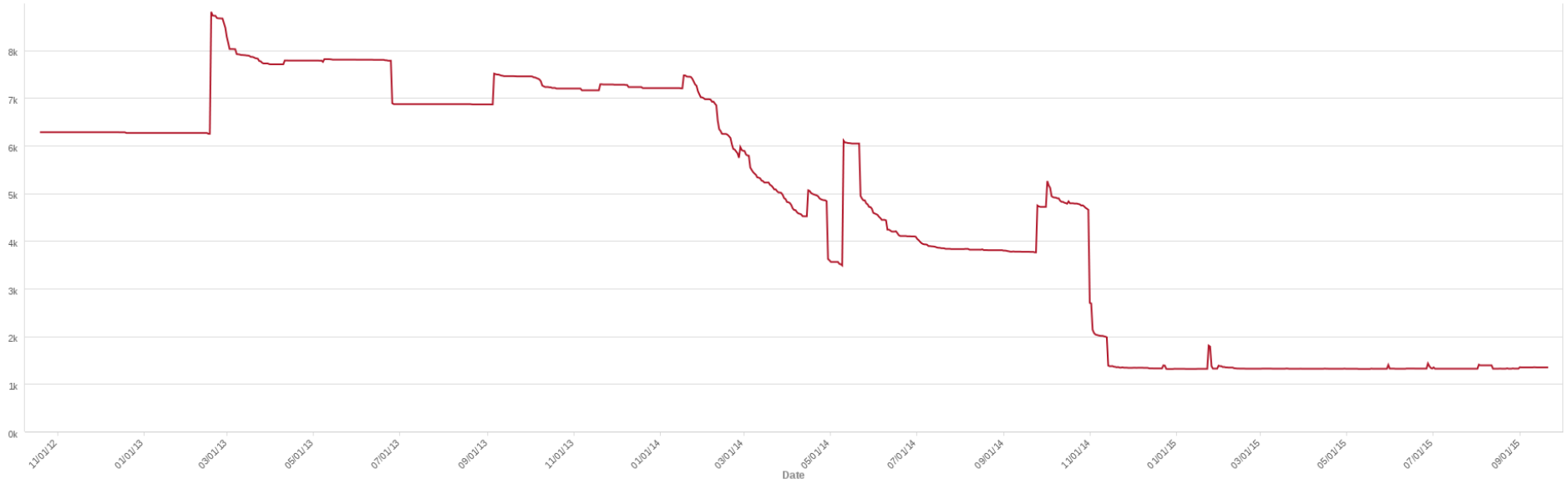
Open Source Defect Density By Project Size

Line of Code (LOC)	Defect Density
Less than 100,000	0.35
100,000 to 499,999	0.5
500,000 to 1 million	0.7
More than 1 million	0.65

Note: Defect density is measured by the number of defects per 1,000 lines of code, identified by the Coverity platform. The numbers shown above are from our 2013 Coverity Scan Report, which analyzed 250 million lines of open source code.

Last Years density at conference time was 0.08

Defects over time



Here, “ignored” third party module warnings are counted.

Process integration

- Now run about twice a week
  - Those are the nums of slots coverity makes available to a project of this size
- Typically back to back
  - One to collect warnings
  - One after warnings fixed
- Results now mailed to the list
- Takes about 4-6 hours to build
- Takes about 12+ hours to analyze server-side


Crash Testing

What it does

- Loads a bunch of documents
  - 118 different columns for formats in output
  - Some are now sort of pointless, e.g. staroffice binary format
  - See if anything crashes or triggers an assert
- Saves a bunch of documents
  - Exports to 12 different formats from all the compatible import formats
  - Export to doc, docx, odb, odg, odp, ods, odt, ppt, pptx, rtf, xls, xlsx

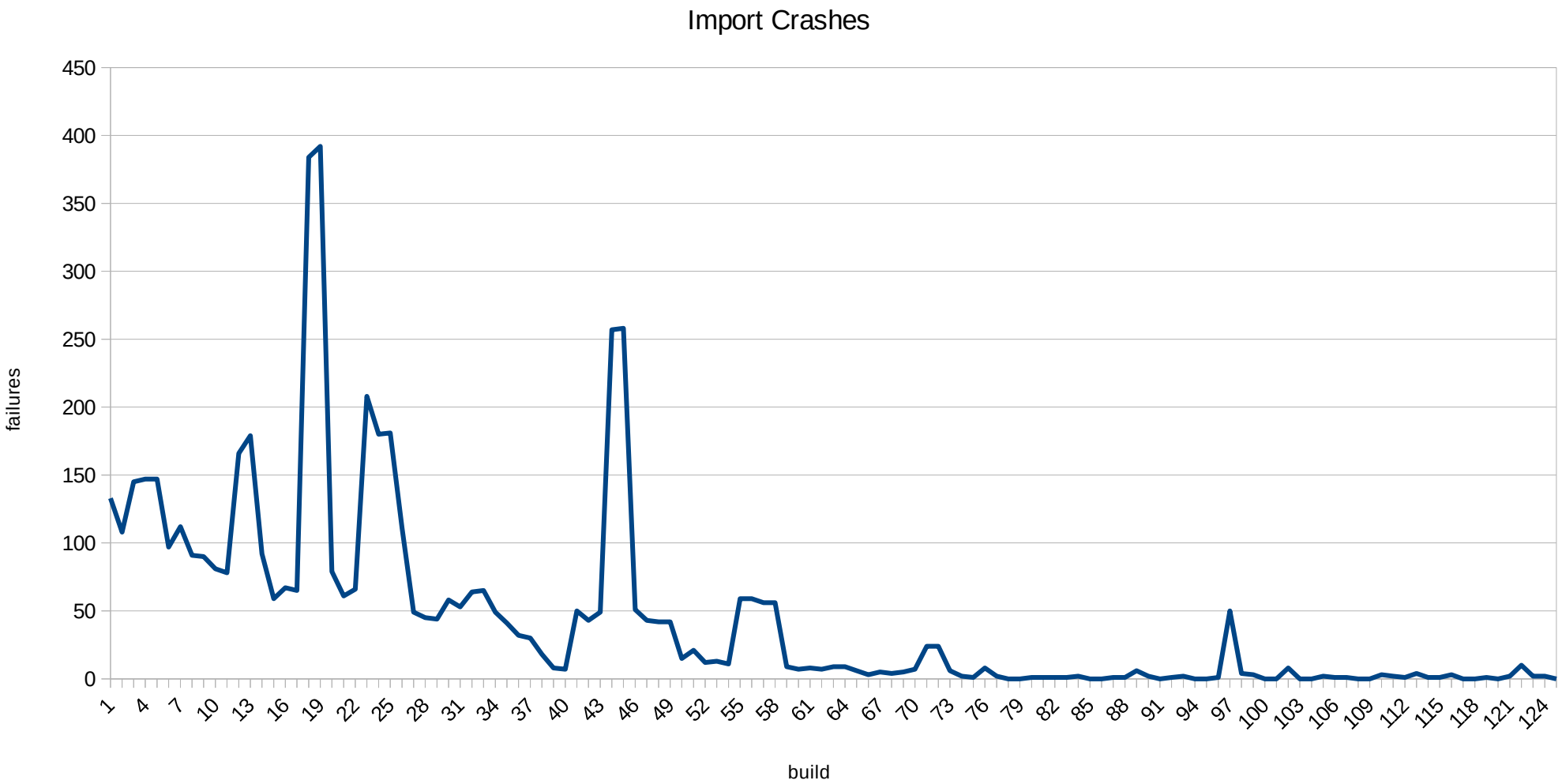
Process integration

- Typically run once or two a week
  - Takes about two days to complete
- Approx 80,000 documents in the document horde
  - Mostly populated from get-bugzilla-by-mimetype
  - + cloudon test documents
  - + w3c svg test documents
  - + various interesting documents that have caused trouble for some app or other in the past

Horde Updating

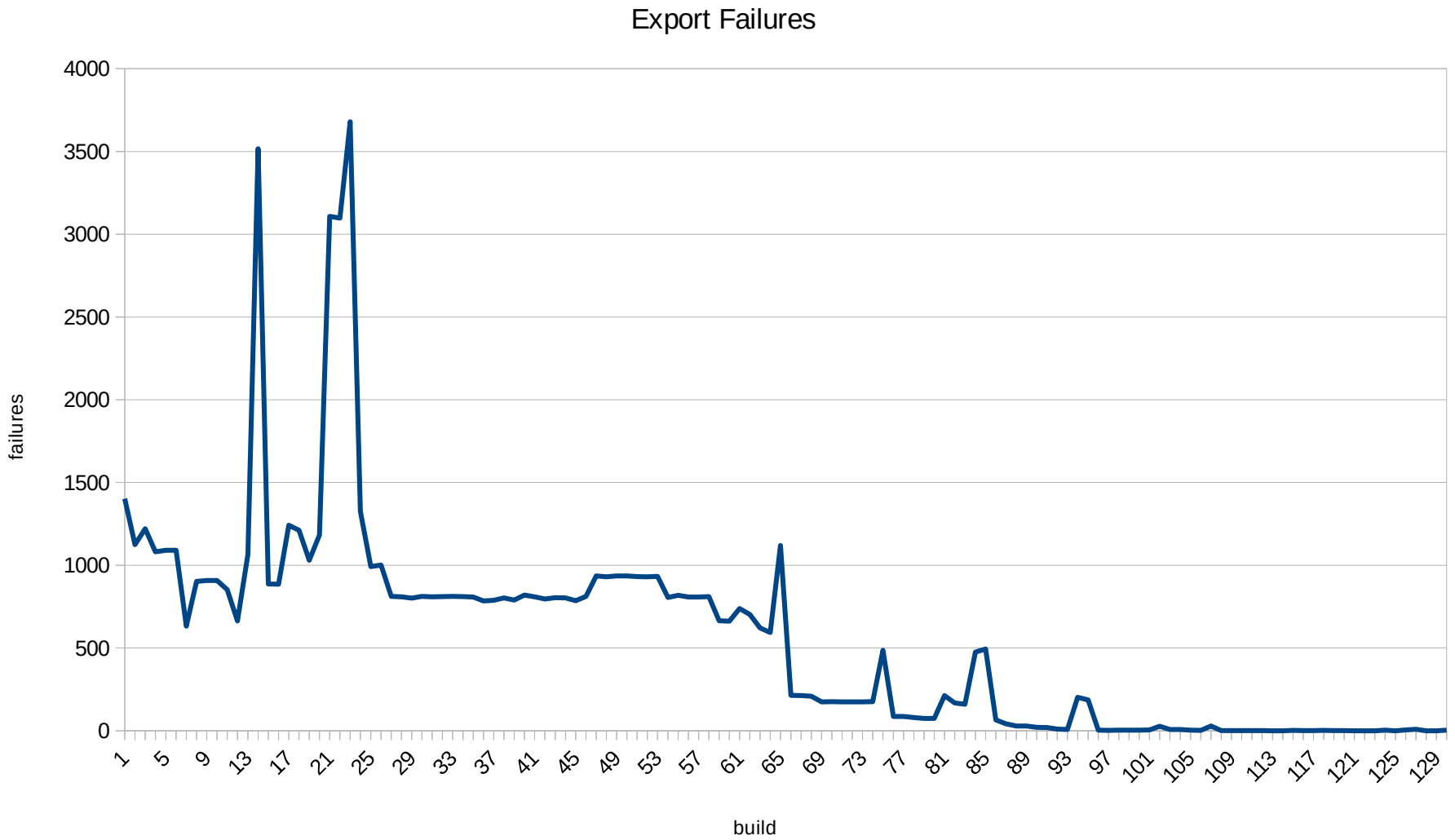
- Typically fairly rarely
- Full update takes about 12/13 hours
- Downloads are cached, so only new documents are updated
- Bugzilla is trusted wrt the mime-type
  - Lots of miscategorized stuff
  - Doesn't really matter, rtfs pretending to be docs, etc
  - Just made doc import filter look a little worse than it was

Import Failure Trends



Build 1 is 31 Oct 2013, final build was 16 Sep 2015

Export Failure Trends



Build 1 is 31 Oct 2013, final build was 16 Sep 2015

Triple 0 week

- 20 – 27 August 2015
- 0 coverity warnings
- 0 import failures
- 0 export failures

Then everyone came back from their Summer holidays

This week

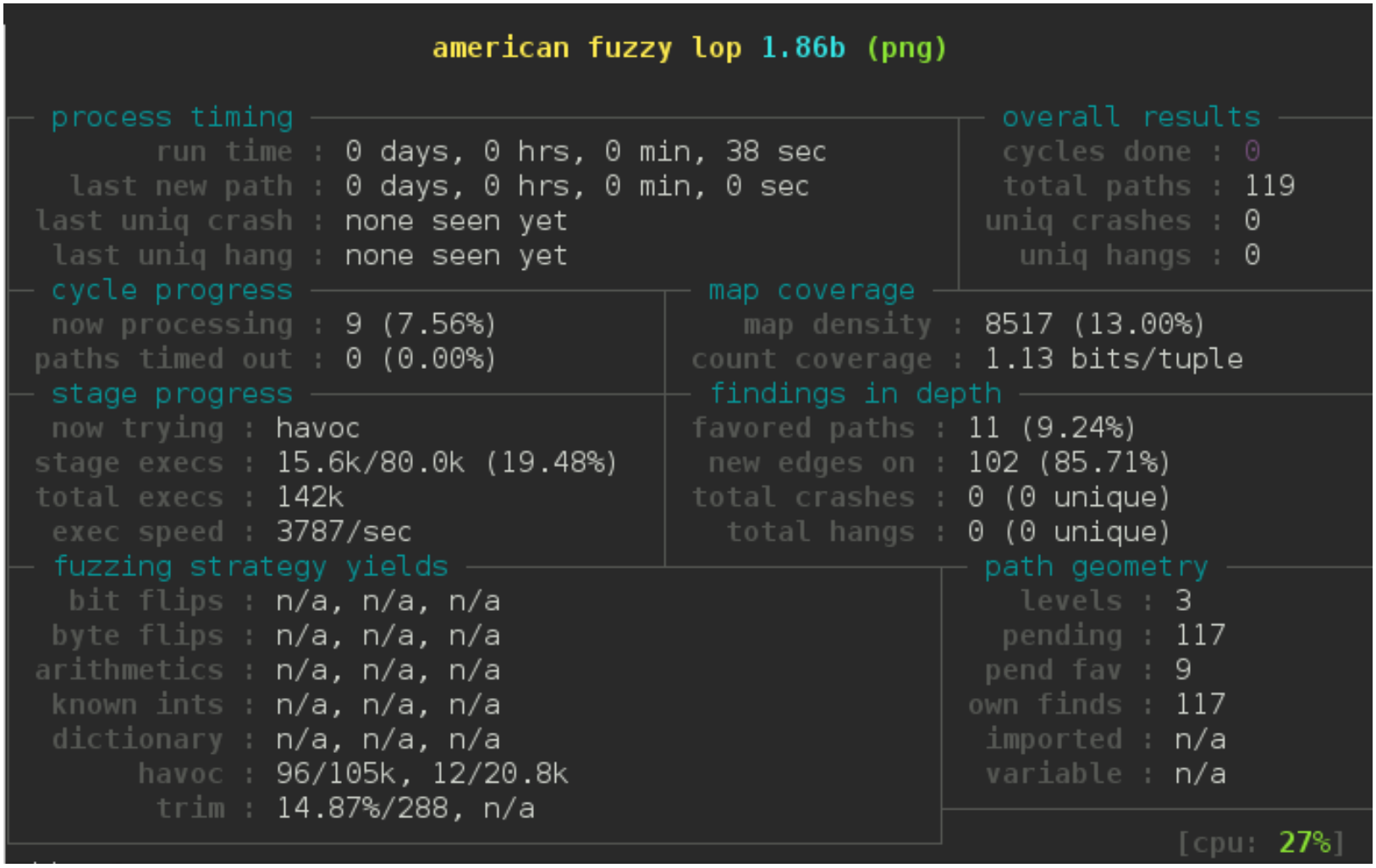
- 4 (fixed) coverity warnings, pending next build
- 0 import failures
- 4 export asserts (2 unique asserts)
- Fairly typical


Taking the battle onwards

Generating troublesome documents

- Fuzzing
- Played with CERT bff for a while, some small results
- American Fuzzy Lop is much more fun
  - Build with afl-clang/afl-clang++
  - “coverage-assisted fuzz testing tool”
  - Generates new documents that trigger new internal states in the target
  - Got to love the UI

Screen Shot


american fuzzy lop 1.86b (png)
process timing
run time : 0 days, 0 hrs, 0 min, 38 sec
last new path : 0 days, 0 hrs, 0 min, 0 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
overall results
cycles done : 0
total paths : 119
uniq crashes : 0
uniq hangs : 0
cycle progress
now processing : 9 (7.56%)
paths timed out : 0 (0.00%)
map coverage
map density : 8517 (13.00%)
count coverage : 1.13 bits/tuple
stage progress
now trying : havoc
stage execs : 15.6k/80.0k (19.48%)
total execs : 142k
exec speed : 3787/sec
findings in depth
favored paths : 11 (9.24%)
new edges on : 102 (85.71%)
total crashes : 0 (0 unique)
total hangs : 0 (0 unique)
fuzzing strategy yields
bit flips : n/a, n/a, n/a
byte flips : n/a, n/a, n/a
arithmetics : n/a, n/a, n/a
known ints : n/a, n/a, n/a
dictionary : n/a, n/a, n/a
havoc : 96/105k, 12/20.8k
trim : 14.87%/288, n/a
path geometry
levels : 3
pending : 117
pend fav : 9
own finds : 117
imported : n/a
variable : n/a
[cpu: 27%]

Speed #1

- Crucial thing is to be able to cycle **fast**
- under 100 execs a second is super cruddy
- soffice.bin is ponderous to startup
  - 0.18 executions a second for pngs
  - Configuration loading and parsing is expensive
- Custom no ui, no config, application
  - After much hacking
  - 40 executions a second for pngs
  - Approximately 200 times faster

Speed #2

- “Persistent mode”
- Don't exit after each document
- Just loop over the same document again and again
- SIGSTOP to afl controller to signal ready again
- Build with afl-clang-fast/afl-clang-fast++
- Makes something of a difference
- 3000-4000 executions per second with custom loader
  - So that's approx 20,000 faster

Process/Results to date

- Between stock crash testing runs afl runs
- 64 core box
- Currently 20+ instances running for the last month or so
- Mostly on a different file format, can run multiple for a single file format
- Crashes rare
- Rich source of hangs
- Using afl-cmin minimized corpus of crash testing as input

Thanks for your time