APACHECON DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Fuzzing Apache OpenOffice

An Approach to Automated Black-box Security Testing

Rob Weir
April 7th, 2014

Presented For The Apache Foundation By
LINUX FOUNDATION

Who is Rob?

1) Rob Weir from Westford Massachusetts
2) rob@robweir.com, @rcweir, http://www.linkedin.com/in/rcweir
3) PMC member on Apache OpenOffice and Incubator
4) Senior Technical Staff Member at IBM

Talk Outline

1) Intro
2) Fuzzing Theory
3) Previous Fuzzing with OpenOffice.org
4) Current Approach
5) Results with AOO 4.1
6) Future Opportunities
7) The End

What is fuzzing?

- Feeding a program random data in order to induce faults.
- Black box fuzzing assumes nothing about the expectations of the program.
- White box fuzzing knows about the underlying formats and protocols.

Theoretical Basis

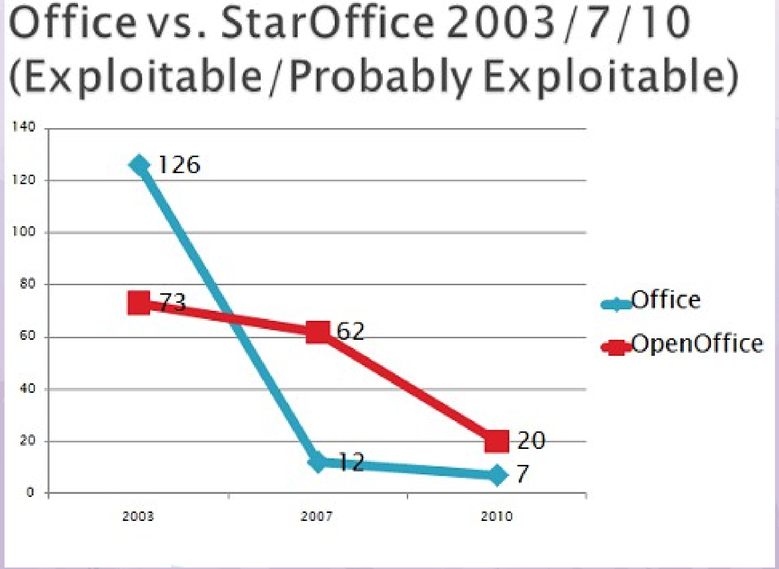
http://upload.wikimedia.org/wikipedia/commons/f/f1/Monkey-typing.jpg

My first fuzzing

- In January 2000, with my Permutator tool, used to test the C++ port of Apache Xalan!
- Take input XSLT, make random changes, run Xalan in a process with custom debugger attached, catch runtime faults, repeat.
- Same basic idea has been elaborated on over the years, but that's essentially it.

Historically a strength of OpenOffice

We have a good historical record of reducing the number of exploitable crashes.



http://dankaminsky.com/2011/03/11/fuzzmark/

Toolset

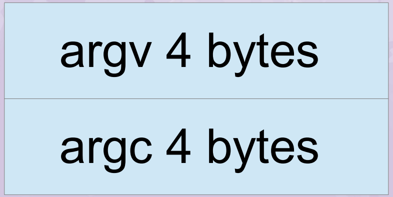
- Bz-attachment-extract.py (custom)
- PeachMinset (from Peach Fuzzer)
- Failure Observation Engine 2.0 (from CERT)
- VMWare/Windows 7 64-bit/AOO 4.1 Beta

What we're looking for

```
void foo()
{
        byte x[9];
        memcpy(x,”123456789XYZ”);
}

void main(int argc, char*argv[])
{
        foo();
}
```

Stack in main immediately before call to foo:



argv 4 bytes

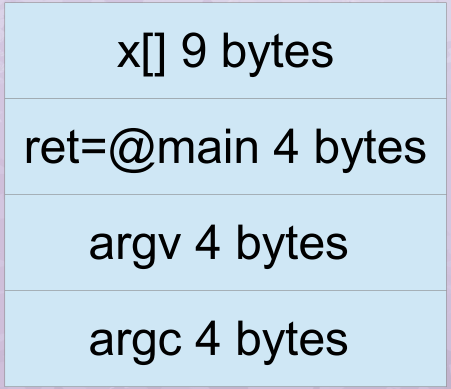
argc 4 bytes

What we're looking for

```
void foo()
{
        byte x[9];
        memcpy(x,”123456789XYZ”);
}

void main(int argc, char*argv[])
{
        foo();
}
```

Stack in foo immediately before call to memcpy:



x[] 9 bytes

ret=@main 4 bytes

argv 4 bytes

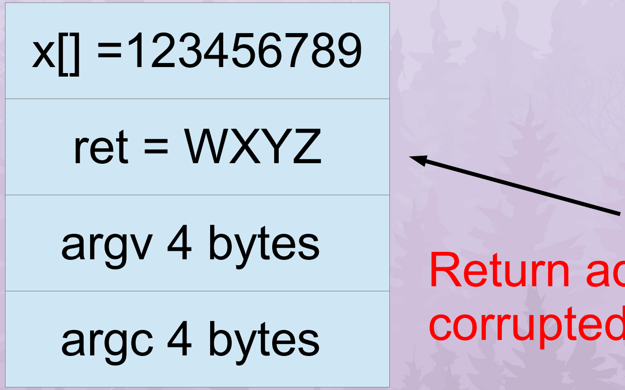
argc 4 bytes

What we're looking for

```
void foo()
{
        byte x[9];
        memcpy(x,”123456789WXYZ”);
}

void main(int argc, char*argv[])
{
        foo();
}
```

Stack in foo immediately after call to memcpy:



x[] =123456789

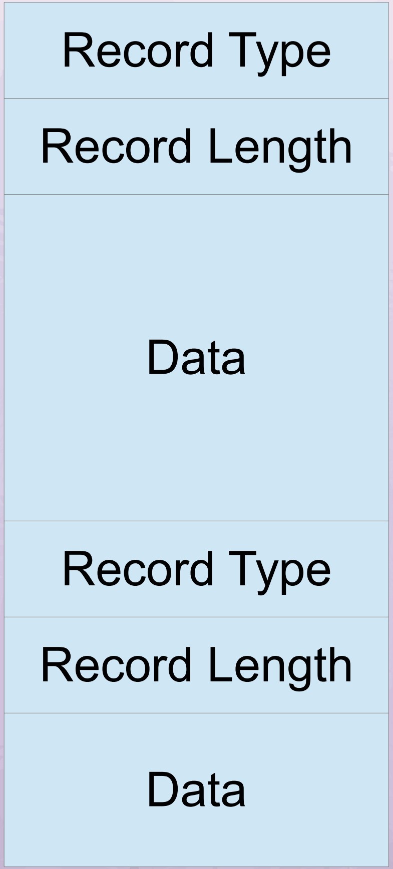
ret = WXYZ

argv 4 bytes

argc 4 bytes

Return address corrupted.

Ancient File Formats



Record Type

Record Length

Data

Record Type

Record Length

Data

Often processed like:

- Switch on record type
- Malloc the specified size
- Cast to a pointer to appropriate struct based on type
- Repeat

Very efficient... when the data is correct.

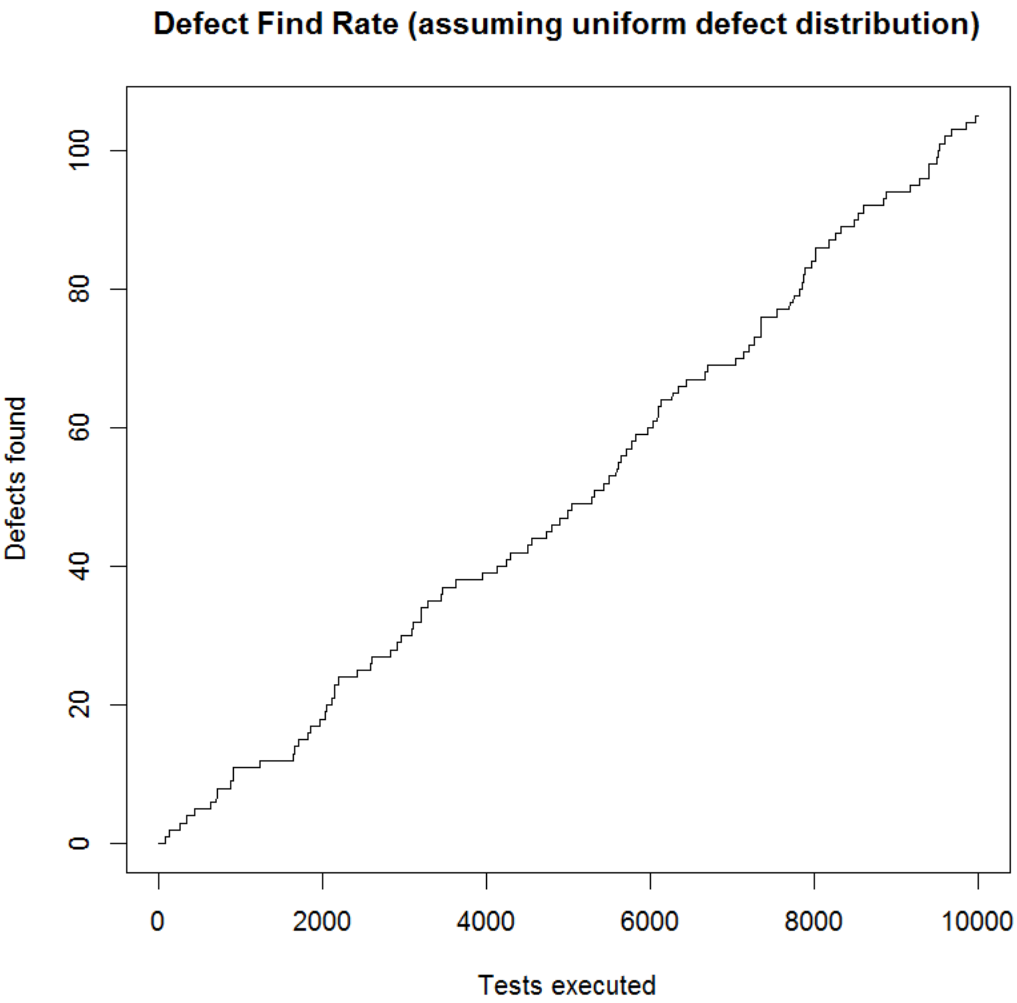
A Large State Space

1	2	3	4	5

5 byte file has 256^5 ~ 10^12 ways to mutate it

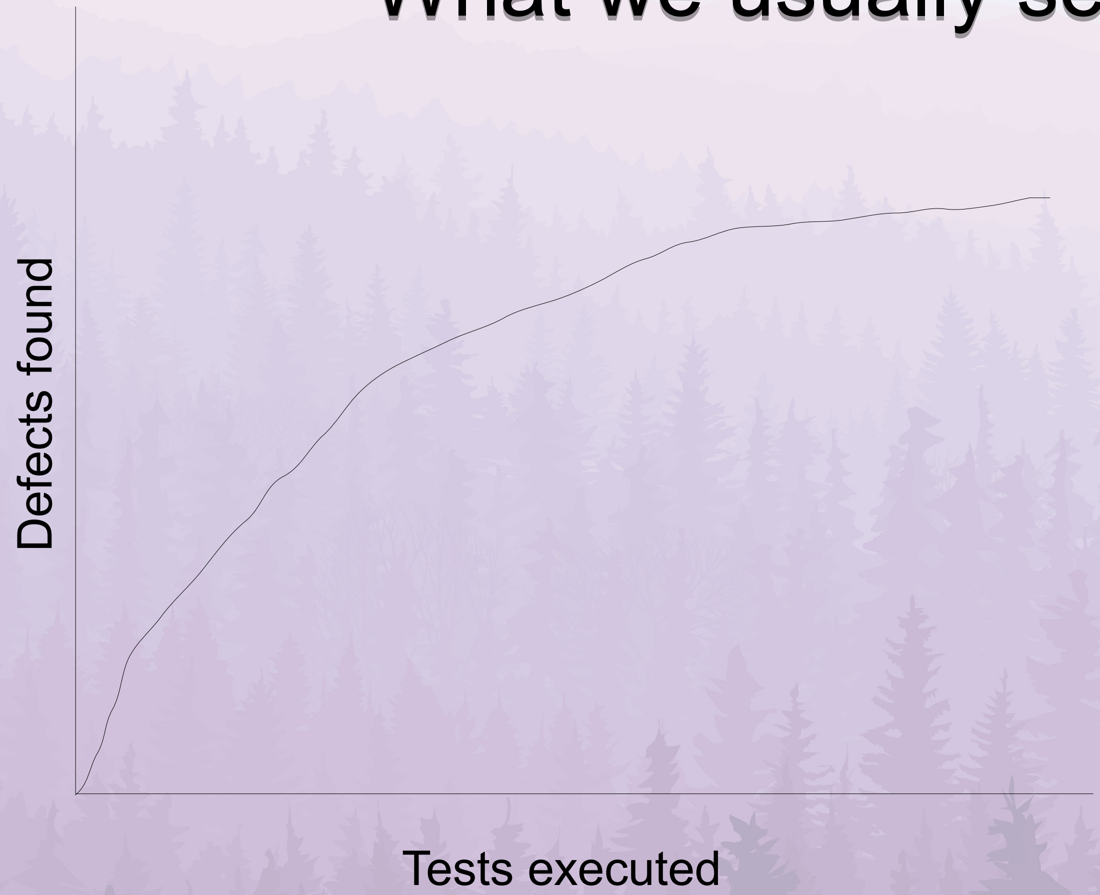
But a typical document is 100KB or more in length ~ 10^2466037 combinations

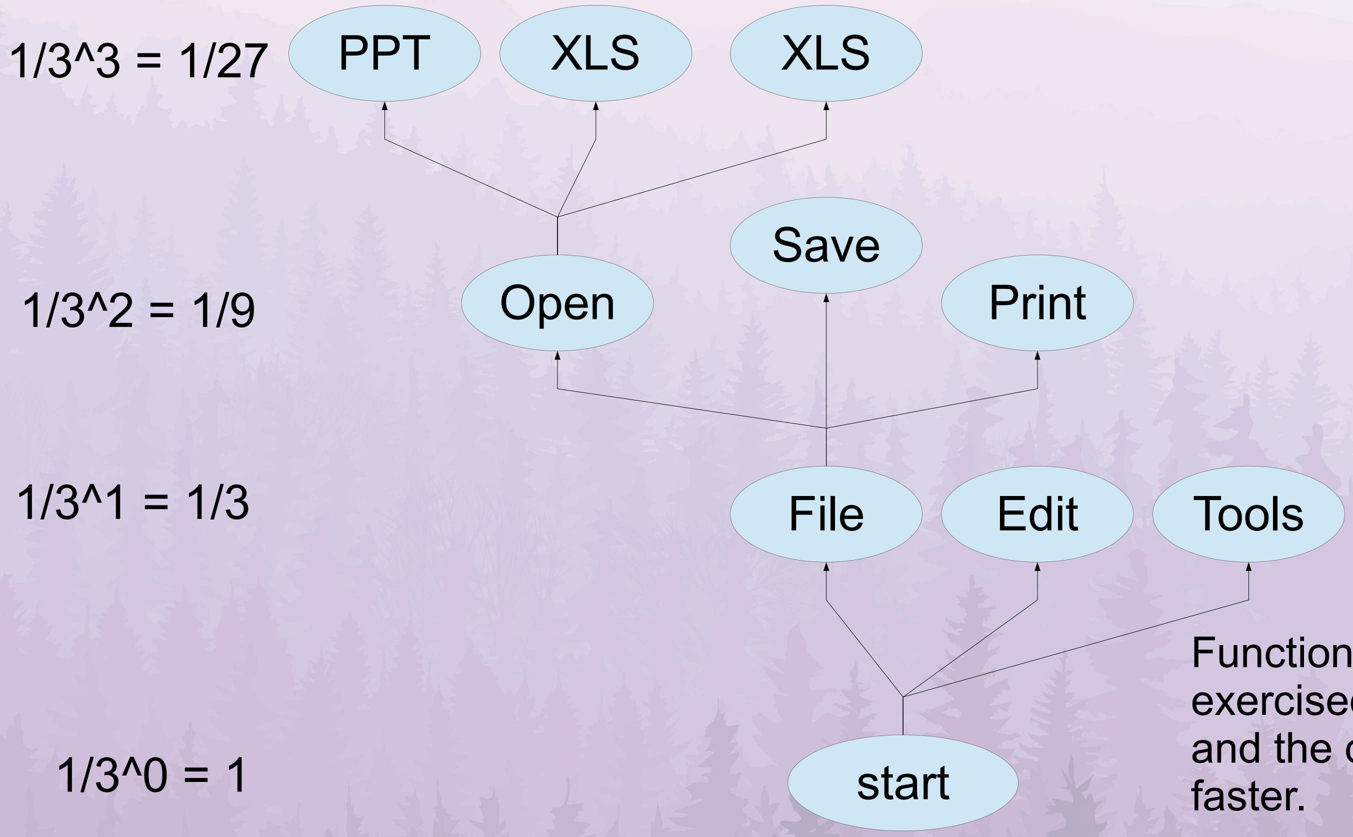
We need to be smart about this or we'll be here all night!



Not a very encouraging dynamic.

What we usually see in QA





Functionality lower in the tree is exercised more frequently
and the defects there are found faster.

A Key Insight

- We can mutate existing documents taken from our Bugzilla
  - We have a large number of documents created over many years in many versions of OpenOffice
  - Broad feature coverage
  - Emphasizes documents that are in product areas that are currently or have been buggy. (Cockroach theory)

bz-attachment-extract

https://svn.apache.org/repos/asf/openoffice/devtools/bz-tools/bz-attachment-extract.py

- Hard-coded to use the AOO instance of BZ, but should be easily adaptable.
- “Nice”, pauses 15 seconds between each download.
- Works off a text file of issue ID's which you can easily get from exporting a CSV from a BZ query.
- Caches the issue's XML so repeated invocations will faster if hitting the same issue.
  - But currently no check for staleness.

What did we get?

- 9,602 total files
- 1328 doc files
- 425 ppt files
- 369 xls files
- 11,211 binary image files

Most were screenshots
not problem images.

Second Insight

- Redundancy makes this inefficient
  - Do we really want to test 10,000 JPG files but only 4 SVM image files?
- We could weight file extensions equally
  - But that fails to account for different complexity of formats
- Solution is to maximize code coverage, pick the minimum set of test files that covers the same code as the entire set of files.

PeachMinSet

- Part of Peach Fuzzer: http://peachfuzzer.com/
- Loads each file, doing an instruction trace and then post-processes the traces to tell you what the minimum file set is.
- A bit temperamental. Required some duct tape and WD40 to work with AOO. Contact me if you want the gory details.

Minset Results

- 225/1328 doc files = 17%
- 144/425 ppt files = 34%
- 46/369 xls files = 40%
- 234/11,211 binary image files = 2%

Total 649 of 13,333 = 5%, so overall a 20x improvement

Failure Observation Engine

- Windows Fuzzing Framework from CERT
- http://www.cert.org/vulnerability-analysis/tools/foe.cfm
- A sister project for Linux, Basic Fuzzing Framework (BFF) is also available: http://www.cert.org/vulnerability-analysis/tools/bff.cfm

Basic FOE Workflow

- Take a seedfile and appply specified fuzzer to it
- Pass fuzzed file to AOO command line
- If a fault is detect then hook in debugger
  - If crash is dupe then skip, else:
  - Pass crash details onto Microsoft's !exploitable to classify the crash
  - Write out crash dump plus the fuzzed and original file
  - Optionally, try to “minimize” the fuzzed file to create a minimal test case.
- FOE learns which files and fuzzing parameters lead to the most crashes.

AOO 4.1 Beta Results

- 4 VMs ran for 1 week
- ~10 tests/minute for each VM
- 4*10*7*24*60 = ~ 400K tests
- Many crashes, over 70 classified as *EXPLOITABLE* by !exploitable.
- But only 4 root causes, which are fixed in the 4.1 GA release.

I can provide more detail in Denver on the actual fuzzing results <u>if</u> AOO 4.1 is released by then.

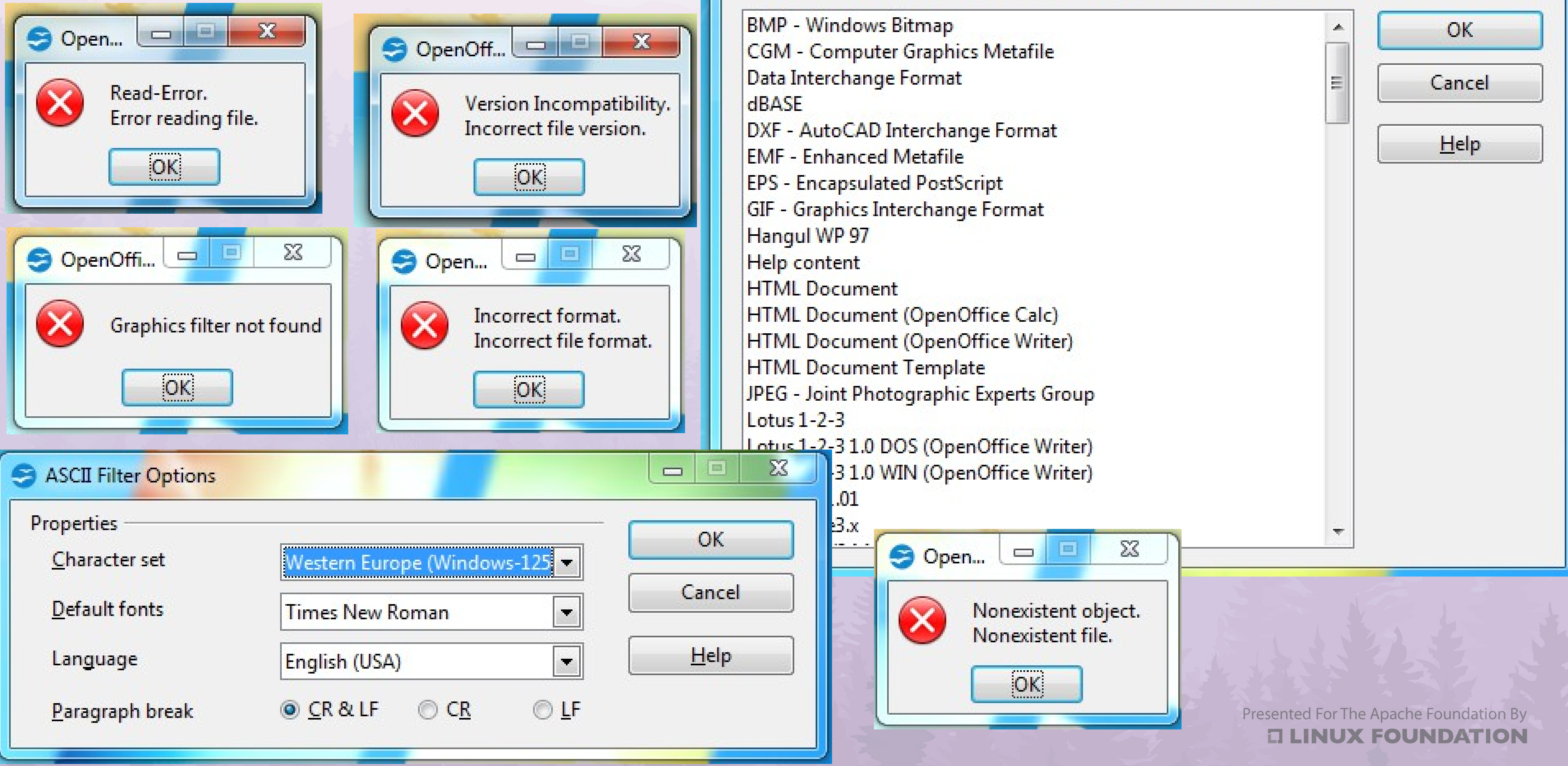
One Approach of Many

- Fuzzing is only one approach, but is not a silver bullet.
- Static analysis, e.g., Coverity is another, complementary, tool.

- We might also consider retiring some of the rarely used binary formats to reduce exposure, or at least make them optional at install time.

Time Permitting: Random Observations

I assume this all makes sense to developers. But to users?



Open...

Read-Error.
Error reading file.

OpenOff...

Version Incompatibility.
Incorrect file version.

OpenOffi...

Graphics filter not found

Open...

Incorrect format.
Incorrect file format.

Filter Selection

C:\FOE2\fuzzdir\campaign_suf2r3\iteration_mwvo7y\foe-crash-veia4k\sf_78a718465c148fe5e

BMP - Windows Bitmap
CGM - Computer Graphics Metafile
Data Interchange Format
dBASE
DXF - AutoCAD Interchange Format
EMF - Enhanced Metafile
EPS - Encapsulated PostScript
GIF - Graphics Interchange Format
Hangul WP 97
Help content
HTML Document
HTML Document (OpenOffice Calc)
HTML Document (OpenOffice Writer)
HTML Document Template
JPEG - Joint Photographic Experts Group
Lotus 1-2-3
Lotus 1-2-3 1.0 DOS (OpenOffice Writer)
3 1.0 WIN (OpenOffice Writer)
.01
3.x

ASCII Filter Options

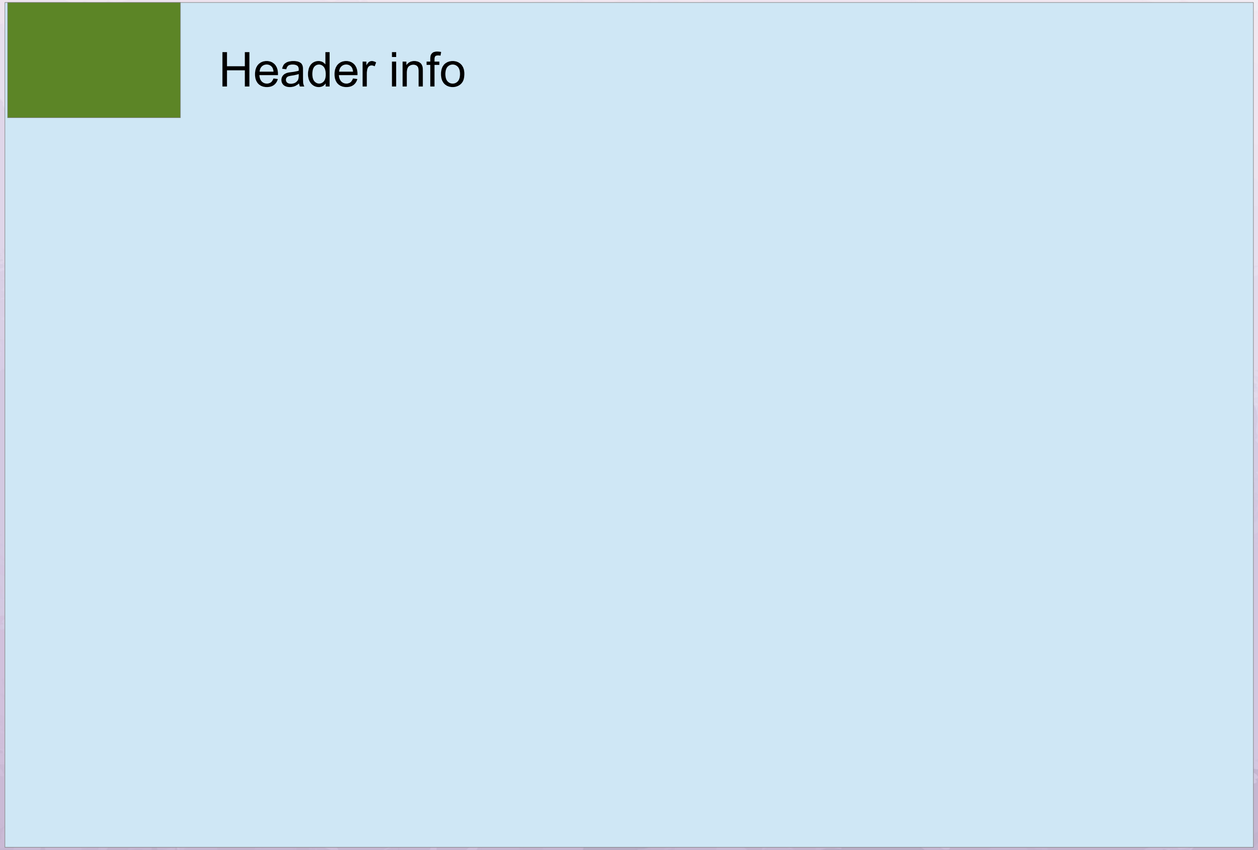
Properties

Character set: Western Europe (Windows-125
Default fonts: Times New Roman
Language: English (USA)
Paragraph break: ● CR & LF ○ CR ○ LF

Open...

Nonexistent object.
Nonexistent file.

Fuzzing a Raster Image



Header info

It is like shooting a jellyfish!

Fuzzing XML

- Most random mutations of XML files cause the file to be rejected. We need to be clever to induce faults in processing of ODF and OOXML, e.g.:
  - Replace numeric attribute values with 0, -1, 1, 2^16-1, -2^16, NaN, INF, -INF
  - Replace string attribute values with “”, “   ”, a large string (16K)
  - Interchange xml:id and idref's
  - Interchange two subtrees
  - Replace character data
  - Schema-directed fuzzing?

Headless Execution

- Idea is to increase test execution rate
- Focus on parsing code, not layout code
- But maybe faults are in layout code also?
- Possibilities for unit-level fuzzing as well

The End

APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Presented For The Apache Foundation By
LINUX FOUNDATION