Showing How Security Has (And Hasn't) Improved, After Ten Years Of Trying

Dan Kaminsky, Chief Scientist, DKH Inc.
Michael Eddington, Partner, Déjà vu Security
Adam Cecchitti, Partner, Déjà vu Security

History (As I See It)

- 1990's: Security == Cryptography and Java
- 2000's: Bad guys show up, start owning everything
  - Got particularly bad in 2003
  - We learn how software *actually* fails
- 2010's: The decade of defense
  - OK. We've gotten software to fail.
  - How do we make software *not* fail?
    - Have we gotten any better?

The Push For Metrics

- *Major* efforts going on in the corporate/defensive world to try to get some sense of Return On Investment / "What Actually Works" in defense
  - Everybody's. Getting. Owned.
  - Something's not working.
- Difference between attack and defense
  - Attack doesn't work, it's easy to tell.
  - Defense doesn't work, it's…not so easy.
- What sort of metrics would be helpful?

http://www.fuzzmark.com

Can We Get Metrics For Software Quality?

- "How hard is it for an attacker to corrupt a computer exposing this particular codebase?"
  - **Security is about systems, not about individual codebases or even just computers, but bear with me**

A Fundamental Question

- We know more things are getting compromised
  - Is this because attackers are getting better?
  - Or is this because code is getting worse?
- **Hypothesis: Software quality has improved over the last ten years.**
  - Corrolary: Code is not getting worse; attackers are getting better

What Sort Of Experiments Could We Try?

- Standard dataset usually pored through for quality metrics: CVE
  - CVE: Common Vulnerabilities and Exposures
- Information on 48,425 vulnerabilities found (and presumably fixed) over the last ten years or so
  - This is lots of data!
  - Or is it?

The Problem with CVE

- Fundamental bias
  - Conflation: Software quality with finder interest
    - Interest not only in *finding* the bugs, but *reporting* the bugs
    - Major human element (at minimum, difficult/impossible to measure as the exclusive metric)
  - Not that much finder interest *that still reports back to vendors*
    - We have a latency problem

Perhaps Static Analysis?

- Could we take the source code from a large number of projects, and look at the number of "bad things" in the source code?
  - Sort of.
  - First issue: Good luck getting source code to targets
    - Could be bypassed if companies published their static analysis rates
    - **This would actually be pretty cool, hint hint**
  - Second issue: In practice, difficult to distinguish *stylistic* concerns from *actual vulnerabilities*
    - Much bigger problem

What About Fuzzing?

- Fuzzing: The use of a large number of generally corrupt and machine generated test cases to attempt to find failure modes in software.
- **Is it possible to use the results of large scale fuzzing as a rough metric for software quality?**
  - **"Fuzzmarking"**
  - Possibly. There's even some precedent
    - BreakingPoint's Resiliency Score against networks and network devices

Lets Start With The Highest Risk Stuff

- Target Formats
  - Office Documents: .doc, .xls., .ppt
  - Portable Documents: .pdf
- These are the formats with the *highest* complexity and the *greatest* exposure
  - Also, empirically they've been the ones that are getting us owned the most
- There are two sorts of studies that can be run against these formats

http://www.fuzzmark.com

Cross Sectional Studies: Which Parsers Are (Maybe) Safest?

- Office Documents
  - Microsoft Word/Excel/Powerpoint
  - OpenOffice Writer/Calc/Impress
- Portable Documents
  - Adobe Acrobat
  - Foxit Reader
  - GhostScript
- …but there's something *way* more interesting we can do.

http://www.fuzzmark.com

Longitudinal Studies: When

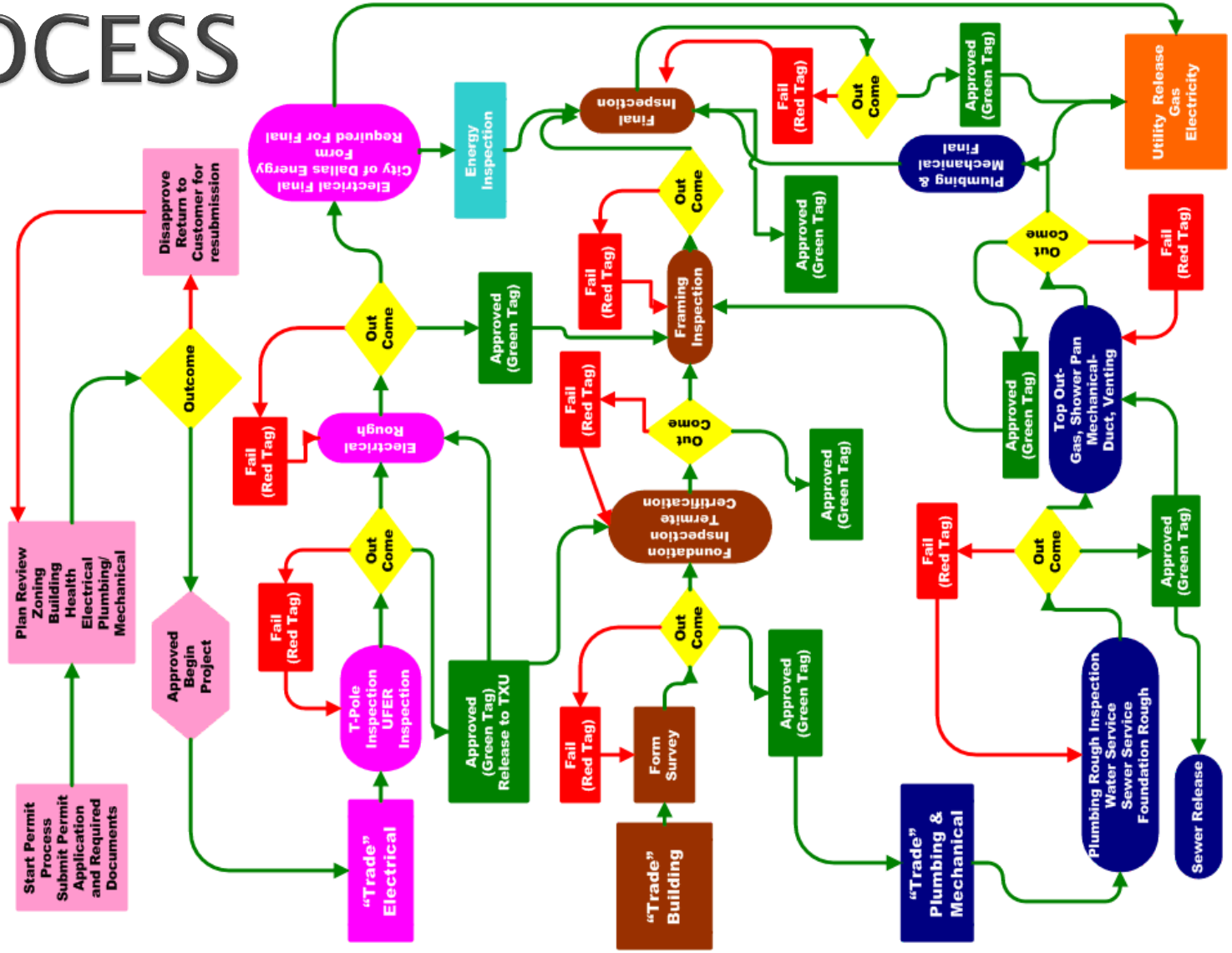
- We've been doing security for ten years.
- How much has software improved in those ten years?
  - *Has* it improved?
  - If so, *when?*
- **We've taken software circa 2000, 2003, 2007, and 2010, and run synchronized fuzz tests across all of them**
  - This is (potentially) a window into bugs that have never seen the light of a CVE
  - So what does the data look like?

188,000 Crashes In 44 Days

- OK, not really.
- 175,334 crashes in ~44 days.
- Details?
  - Lets talk first about process, procedure, and how you're going to screw this up if you try to do this.

http://www.fuzzmark.com

PROCESS



1

Obtain Sample Files

- Scavenge sample files from Google
- Internet Archive
  - Looking for files near the year 2000
  - This did not work out so well…

Generate Fuzzed Files

2

- Pre-generate all fuzzed files using Peach
- 100,000 files per format (doc, xls, ppt, pdf)

- Office: Understand OLE Storage
  - Fuzz the streams in doc/xls/ppt

- PDF: Decompress streams
  - (Eventually)

Peach Farm

- Hosted fuzzing service from Déjà vu Security
- You provide the target, we produce the crashes

For this talk…

- 88 Cores of fuzzing fun
- 5 Targets
- 4 File formats
- >250 GB of logs

Running of the Fuzzers

- Office 2010/2007
  - Thousands of Registry Keys (File recovery)
- Office/OpenOffice
  - Tens of thousands/Gigs of temp files
- OpenOffice
  - “Quick Start” keeps OO in memory
- PDF
  - Compressed/Non-Compressed

http://www.fuzzmark.com

5

Verify Logs

- Look correct?
  - Was every iteration a crash? #fail
  - No crashes?
- Complete set?
  - Runs crash
  - Machines die
  - Logs misplaced

Mine Logs

- Collect interesting data
- Mine data into SQL
  - Easy to query for data sets

How Not To Fuzz Fuzzing

- “How hard could opening a program and reading a file be?”
- Run the same 100k files across 18 programs.
  - Every file, every program, every time.
  - Must open the program
  - Must open, read, and parse the file.
  - Must record the result.

Opening the Program

- Personality
  - Some software has very low self esteem and needs constant attention.
    - “Look what I parsed!”
  - Each program ends up with it own personality based on how many deterrents there are to fuzzing.
    - Ranging from whiner to aggressive to needy.
  - Occasionally 30k / 50k runs we’d have to “sweet talk” the exe back into running.
  - Some things were predictable and others randomish.

Personality traits

- From the program
  - Auto Updaters : Version 6.2 -> 6.5
  - Auto Upgraders : Version 6 -> 7
  - Bundle ware / Nag ware
  - Recovery of crashed files
  - “Safe modes”
  - Importing
  - Error reporting / Crash Analysis software
- From the OS
  - Auto Update
  - Anti Virus
  - Scheduled scans / reboots.
  - Startup / OnRun / Preloaders

http://www.fuzzmark.com

Click here to punch clippy

- From the Program
  - Pop Ups
  - Registration
  - Surveys
  - Ads
  - Assistants

Files from the internet why not DLLs?

- Dynamic downloads
  - Add Ons
    - Fonts / international support
    - Foxit from 2007 doesn't have default JPEG support
    - Download and install at runtimes
    - Puts the program out of sync with the rest
- "Static" downloads
  - Old documents linked to things all over the web.
  - Parts of the fuzzed doc / pdf are just gone.
    - Makes for less interesting fuzzing

Parsing the file

- A few “docs” we crawled were really
  - HTML, JPEG, TIFF, PDF, JavaScript
  - Ended up with a lot of integrity checking of file
- Crawler and verifier is around 3k lines of C# code
- Normally we wouldn’t care a crash is a crash to us, but we wanted to make sure our file set was valid.

Things to check for

- Magic Number check
  - Caused false positive
  - Searched for JavaScript and HTML tags
- Basic length checks
  - Failed downloads / truncated uploads
- Hash of file
  - Resume a downloaded crawl that failed.
- Filename -> guid
  - Many files on the internet share names. “Homework1.doc”, “Expenses.xls”

Recording the results

- Data integrity becomes very important
- Collection of crashes and mining of the data requires more meta data the more complex you want to be.
- Tag the results directories with
  - Program name, year, date, suite, file type, version, machine, iterations, and a guid.
  - When in doubt add another guid. No for real.
  - This will barely be enough to manually reconstruct the run in the event things go wrong.

Data Analysis (Naïve And Totally Misleading Edition)

- 175,334 Crashes Across 26 Codebases
  - Word/Excel/Powerpoint 2003/2007/2010
  - Acrobat/GhostView 2003/2007/2010
  - Foxit 2007/2010
- One crash about every six seconds
- One crash about every ten iterations
- *So What?*

You don't fix crashes. You fix bugs.

- You don't fix the same bug over and over
- You also don't fix bugs in the order they appear
  - Severity matters
- How do you operationalize bug classification?

Told you !exploitable was important

- !exploitable: Microsoft debugger extension that:
  - A) “Bucketizes” the crash – allowing it to be matched with similar crashes
  - B) Attempts to measure the severity of the crash
- Yes, this can be done manually, but:
  - Manual analysis doesn’t scale

Unique Crashes

- 2536 Unique Major/Minor Hashes
  - 1149 in Office
  - 1125 in OpenOffice
  - 181 in Ghostview
  - 70 in Foxit
  - 10 in Acrobat

Slicing Things Down

- But that's Major/Minor
  - Most "minor hashes" represent the same underlying bug under a single major
    - Depends on the product…75-95% chance of minors representing the same bug
  - Lets be conservative -- only consider Major
- 942 Unique Major Hashes
  - 440 in Office
  - 396 in Open Office
  - 68 in Ghostview
  - 32 in Foxit
  - 5 in Acrobat

What About Severity?

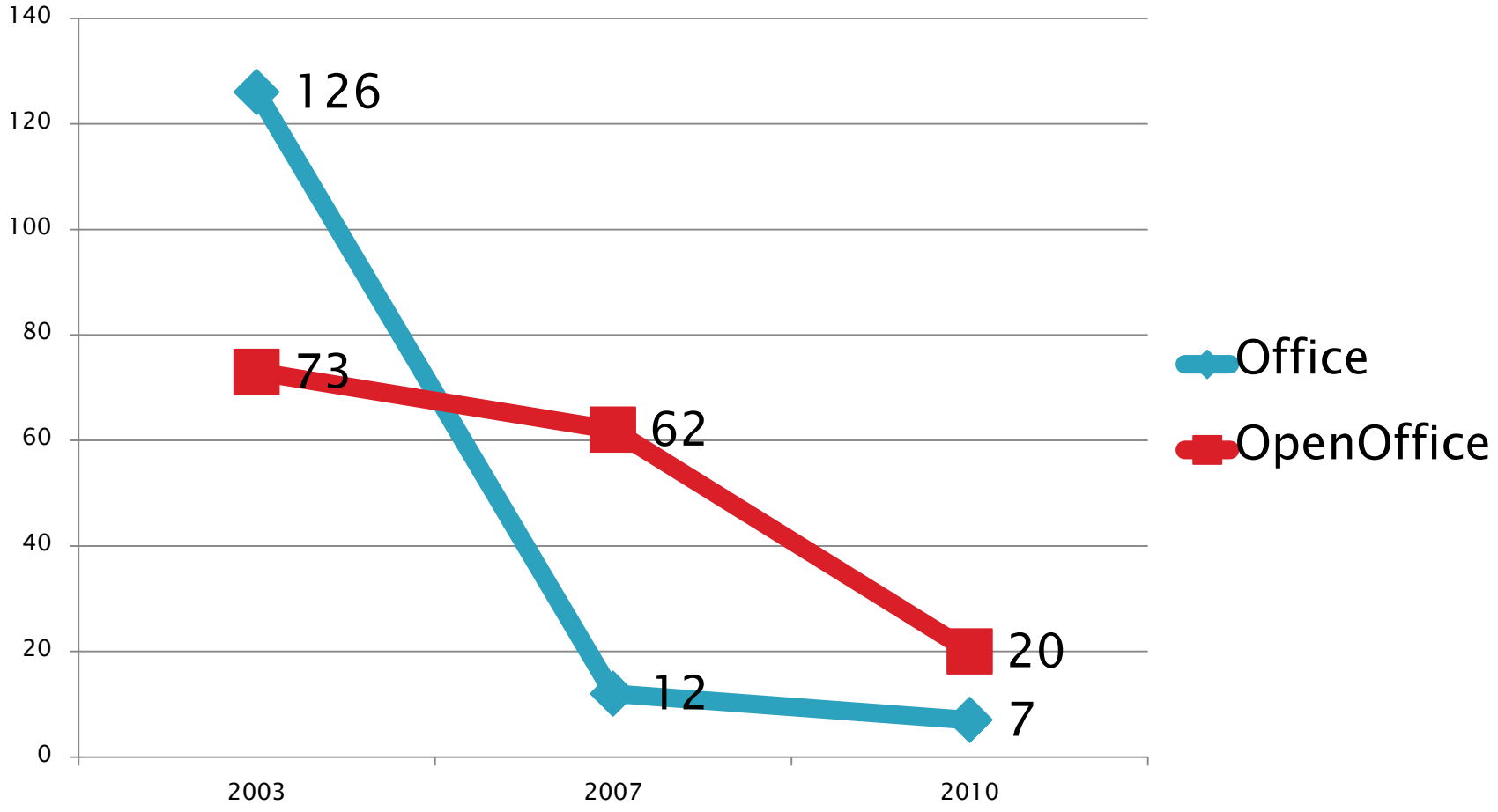
- 942 Unique Major Hashes
  - 150 EXPLOITABLE (15%)
  - 188 PROBABLY_EXPLOITABLE (19%)
  - 16 PROBABLY_NOT_EXPLOITABLE (1.6%)
  - 588 UNKNOWN (62%)

Now, lets merge Severity with our other metrics…

- This is a *cross-sectional* study
  - Is severity equally distributed across the various parsers?
- This is a *longitudinal* study
  - Is severity equally distributed across the various versions?

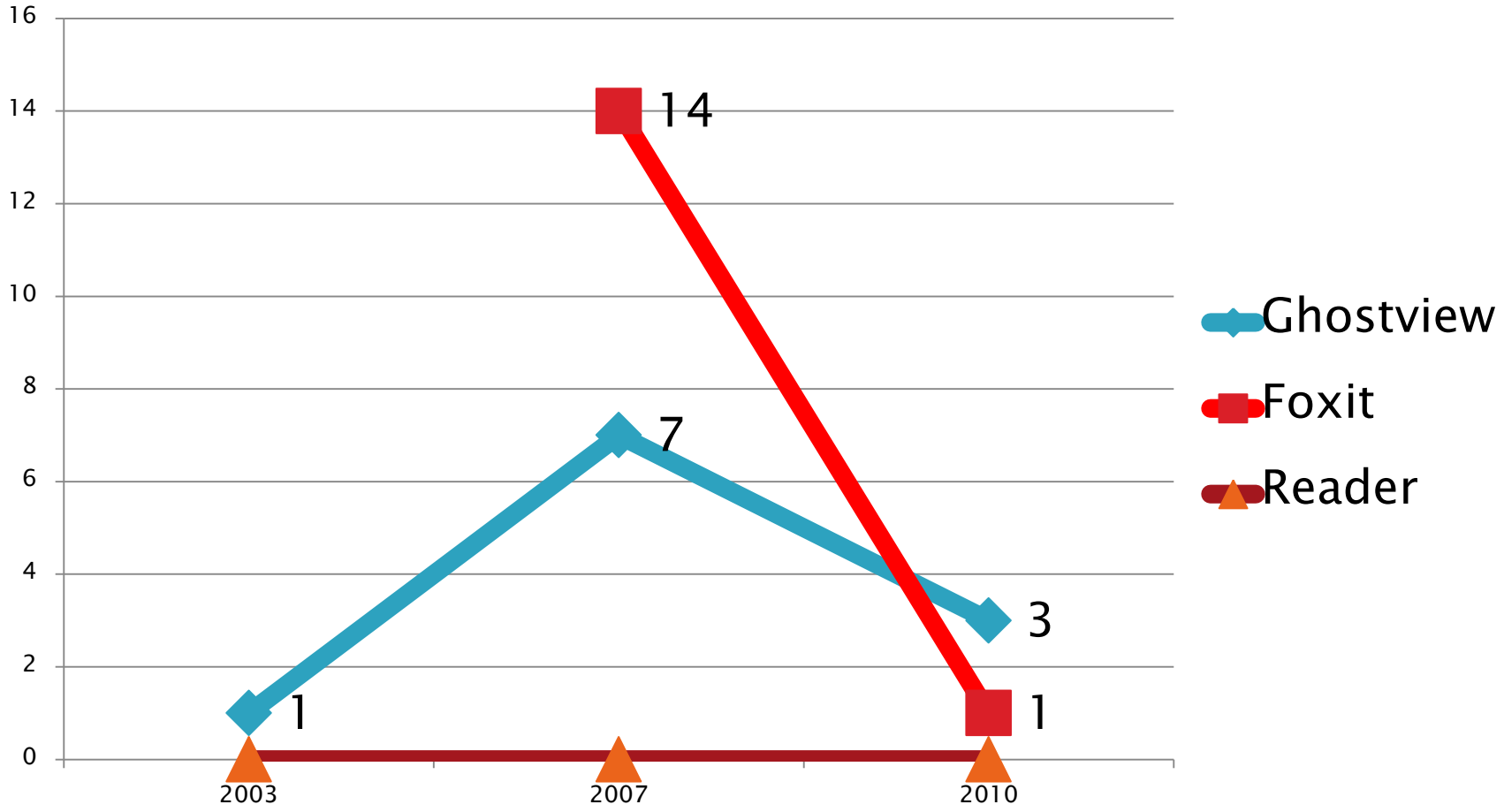
http://www.fuzzmark.com

Office vs. StarOffice 2003/7/10 (Exploitable/Probably Exploitable)



http://www.fuzzmark.com

Ghostview vs. Foxit vs. Reader 2003/7/10 (E/PE)



Systematic Fuzzing Seems To Reflect Improving Code Quality

- Fuzzmarking is implying an across the board effect – code shipped in 2003 was objectively less secure than code shipped in 2010
  - At least, in parsers for the highly targeted file formats doc/xls/ppt/pdf
  - **We would be surprised if this effect showed up in parsers for file formats that do not cross security boundaries (i.e. aren't targeted)**
    - (But aren't using XML/JSON)

Can One Compare File Formats With Fuzzmarking?

- The fundamental tension
  - We know that some file formats “invite trouble”
    - Lengths
      - Explicit (especially 32 bit)
      - Overlapping
      - Implicit
    - Jump tables
    - Diffs (“fast saving”)
  - .txt is going to end up being a safer format than .doc or .pdf
    - But what about .doc vs. .pdf vs. .rtf?

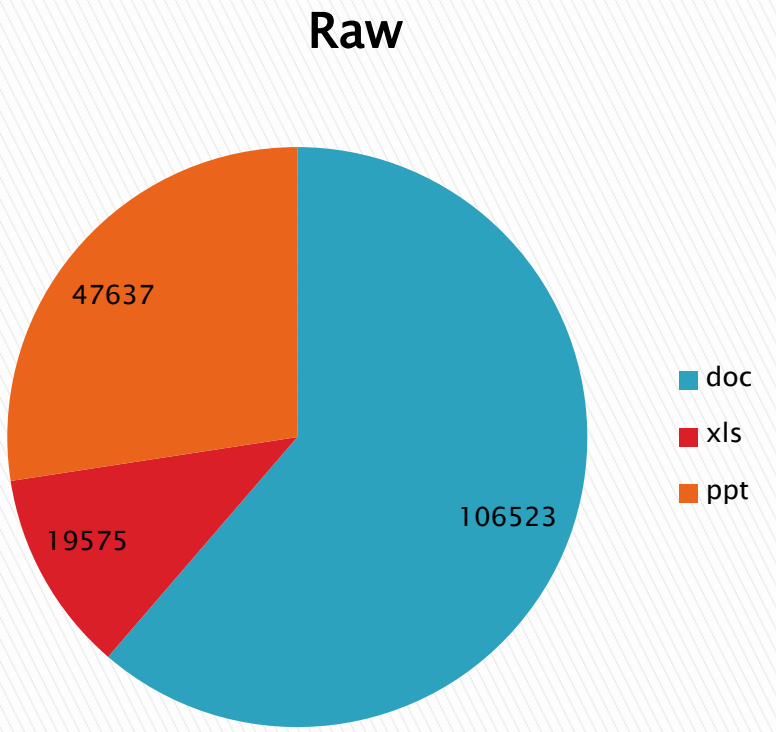
Danger!

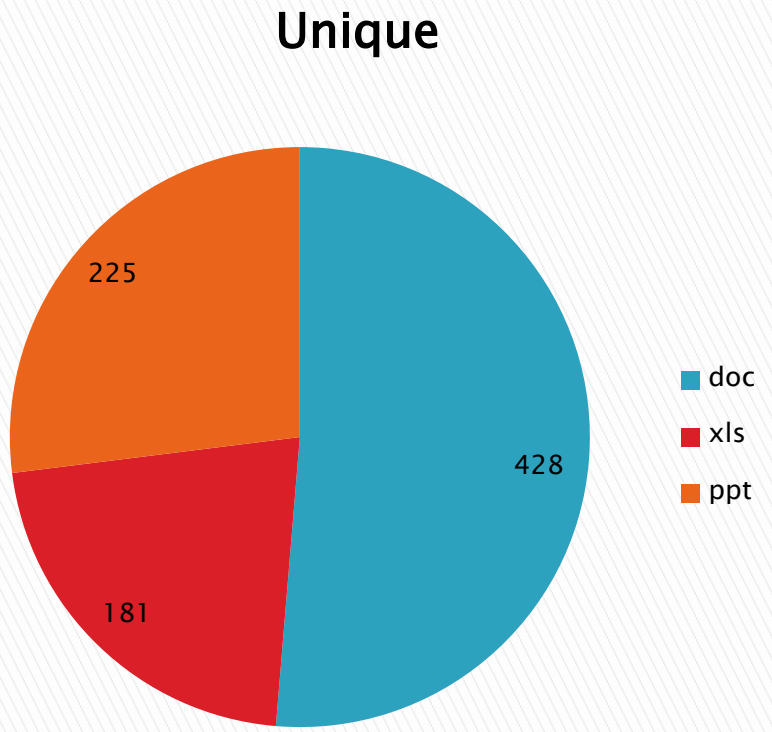
- Absolute comparisons *across file formats* suffer conflation between *raw danger of the format* and *quality of the fuzz tests*
  - Are we getting more crashes because the format has more things to find?
  - Or are we just better at finding what's there?

At least we know where we stand in this case

- In our case, it's very obvious why we have more Office crashes than PDF crashes
  - We're relatively document aware in doc/xls/ppt
  - We're just flipping bits on PDF
    - Our PDF fuzzer isn't even decompressing yet
    - We were surprised to see any crashes!
      - (And we didn't see any E/PE in Acrobat)
- What about Office Format vs. Office Format?

At First Glance, Does Seem Like .doc has more bugs than others



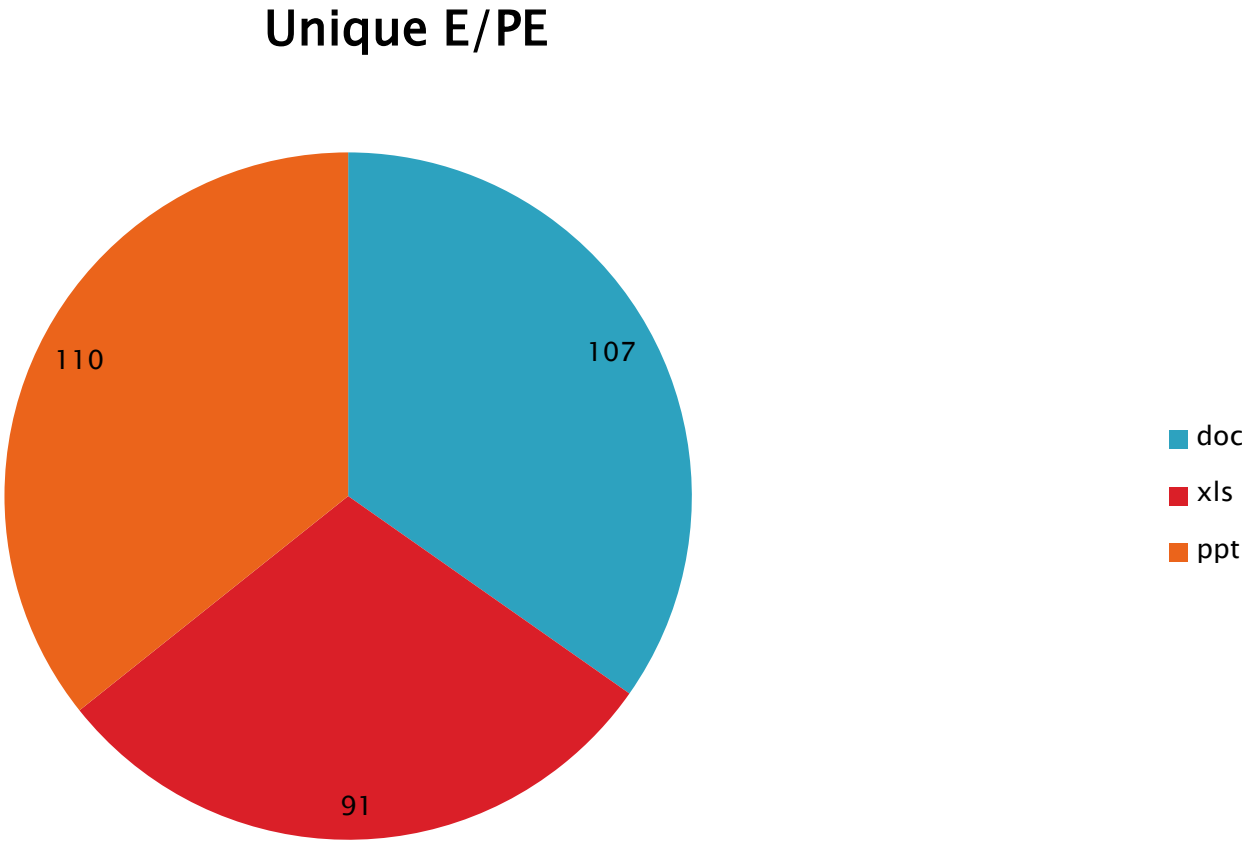


Total Crashes Per Type

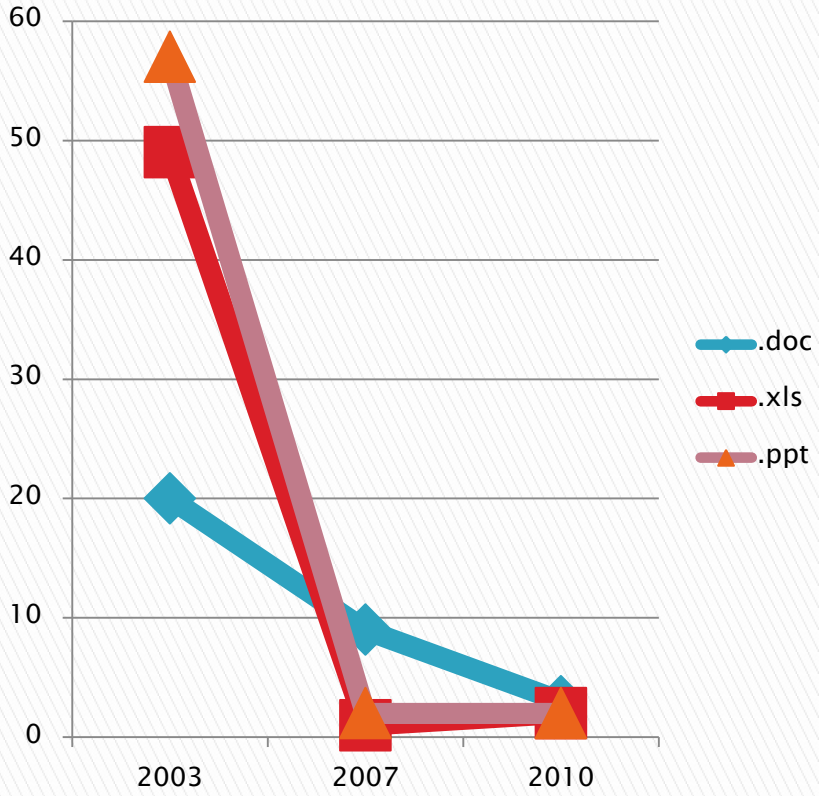
Total Unique Majors Per Type

http://www.fuzzmark.com

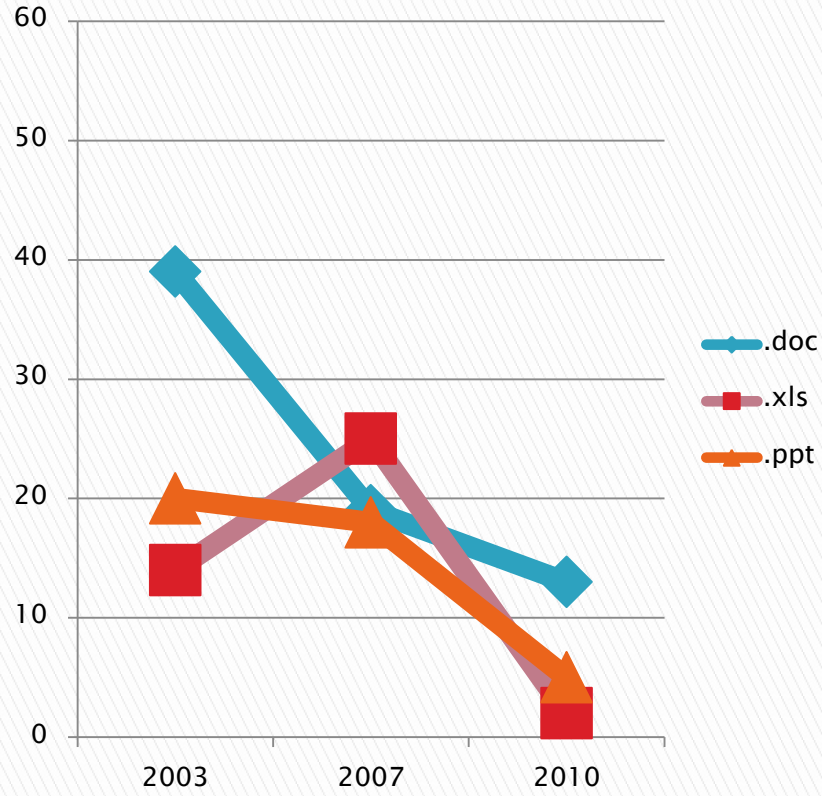
When you filter to E/PE, the formats get equal…



Only Simultaneously Viewing Severity/Version/Type Helps



Office Doc/XLS/PPT E/PE



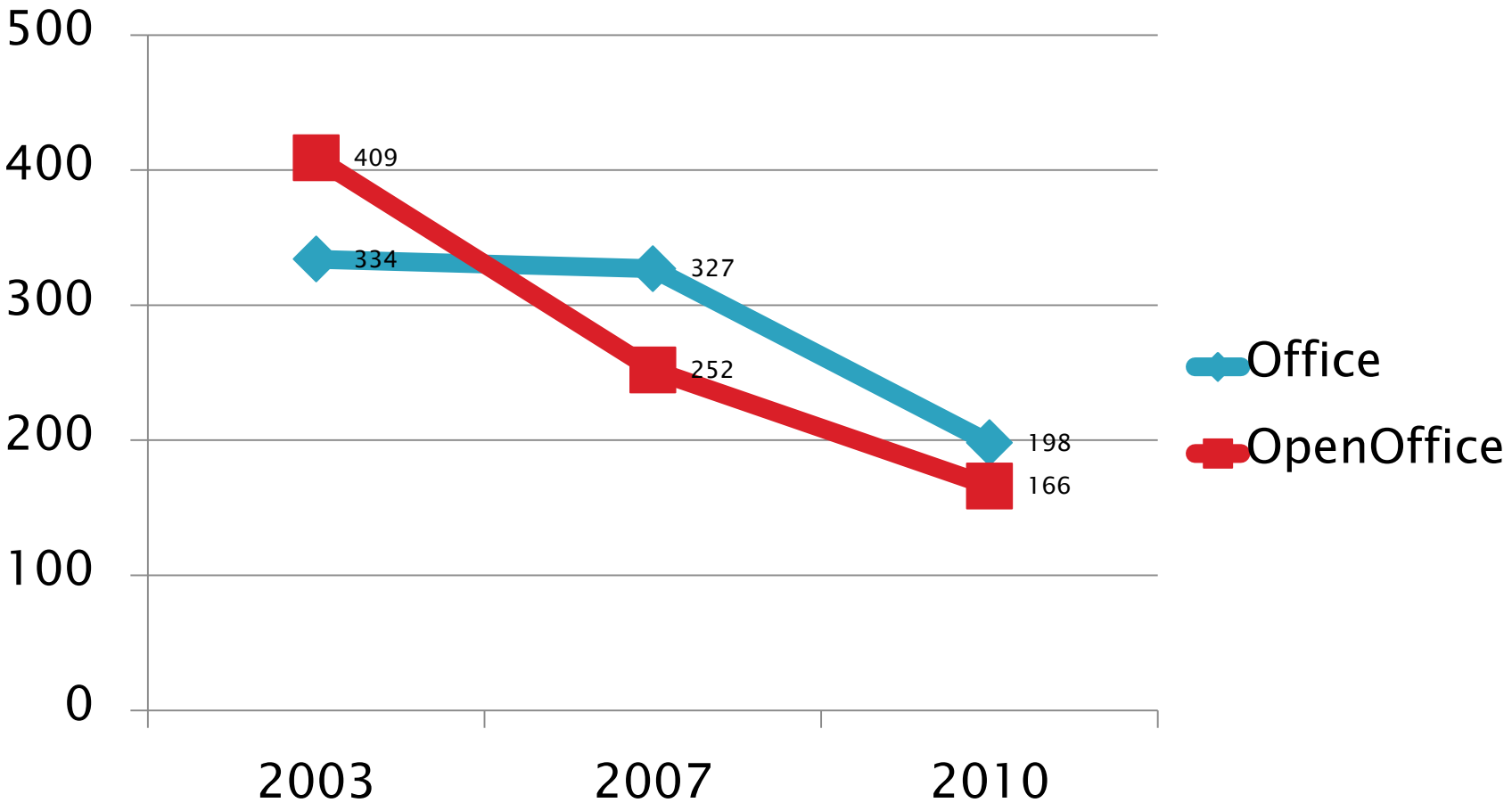
OpenOffice Doc/XLS/PPT E/PE

Analysis

- What's going on?
  - No serious correlation between Office and OpenOffice on .doc risk
- What's *not* going on?
  - .doc, .xls, and .ppt can't actually have any security differential
  - You can reach any parser from any
  - We're not going to notice that without *really* smart template generation

http://www.fuzzmark.com

Office vs. StarOffice 2003/7/10 (UNKNOWN)

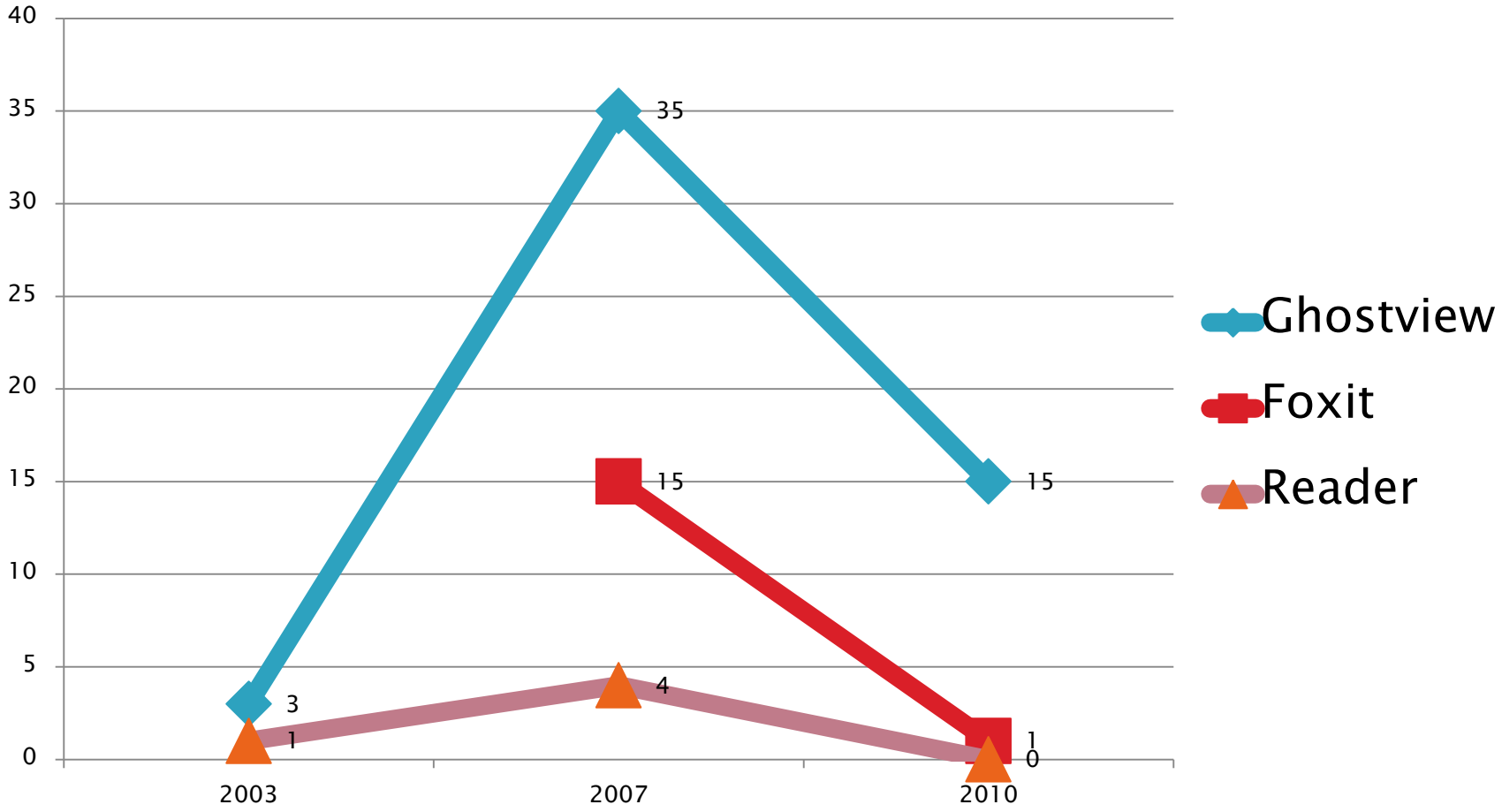


From The Unknown Data…

- 1) There's still improvement, but not as sharp
  - It's almost like Exploitable / Probably Exploitable bugs are more likely to be fixed than Unknown bugs
- 2) There's a far higher floor, even in 2010

http://www.fuzzmark.com

Ghostview vs. Foxit vs. Reader 2003/7/10 (UNKNOWN)



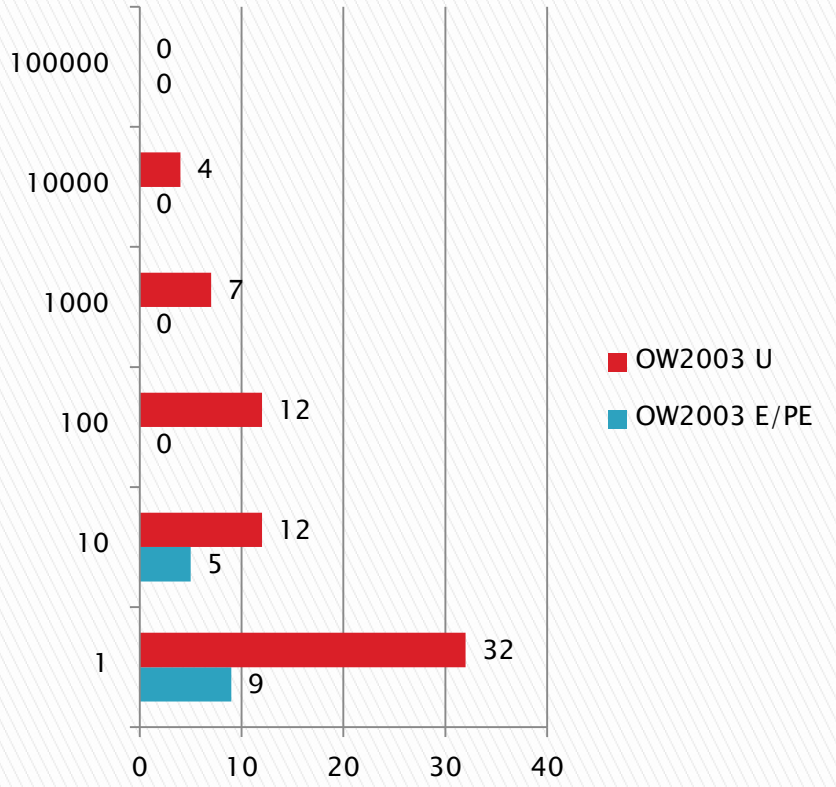
What About PDF?

- We're still seeing improvement
- PDF had an interesting artifact in E/PE – 2003 was better than 2007
  - This artifact is repeated in U – why?
    - More features in 2007 parsers
    - More *corruption resistance* in 2007 parsers
    - Template preference for 2007/2010 parsers

Another Possible Metric: Bug Rarity Profiles

- Nate Lawson showed: *Never just average everything together. Always look at the distributions.*
- Chunking down from 174K crashes to 940 unique vulns throws away a lot of data
  - It matters: Does a given bug take 10 rounds to find? Or 100,000?

Number of Crashes Per Unique Bug (A "Spectral Fingerprint" For Fuzzmarking)



Office 2003

OpenOffice Writer 2003

What About Bugs That Hit More Than One Target?

- How do you correlate the same bug across Word and Writer?
  - The stack traces are different
  - Obviously the Major/Minor hashes are different
  - *But the fuzz file is the same*
- Naïve Numbers
  - 110,637 unique files created crashes
  - 65989 (59%) crashed only one target
  - 44648 (41%) crashed more than one target

Can We Do Better With Viz?

- Can we visualize this?
  - Put the filename in the center
  - Treat the left side as “Office Word”
  - Treat the right side as “OpenOffice Writer”
  - Treat *distance from the center* as *distance from now*
  - Color by severity
  - Sort by recentness of vuln * severity
  - Make each pattern unique

http://www.fuzzmark.com

Office Word 2003/2007/2010 v. OpenOffice 2003/2007/2010 [0]

OW2003	OW2007	OW2010	fname	SW2010	SW2007	SW2003
U	U	U	e0317231-	E	E	
U	U	U	e0317231-	E	E	
		U	0599f01b-	E	E	E
		U	7885bd51-	E	E	E
		U	8765523c-	E	E	E
		U	39881f42-	E	E	U
		U	063b8578-	E	E	U
		U	006644a6-	E	E	
		U	b38d8e4b	E	E	

http://www.fuzzmark.com

Exploitable in Writer 2003, Unknown in Word 2003/2007/2010

U	U	U	a95956d2-		E
U	U	U	2587045f-		E
U	U	U	13385ec7-		E
U	U	U	1bdddddad		E
U	U	U	7adeb1a8-		E
U	U	U	01ad7cab-		E
U	U	U	483af705-		E

Easy Dereferencing Back To Major Hashes

	U		0474b7e0-	PE
	U		b3bdcbb6-	PE
	U		e25d26d5-	PE
	U		dfb36d56-	PE
	U		2d979917-	PE
	U		01ad7cab-	PE
	U		6b6cd1c4-	PE
	U		0474b7e0-	PE

0x26461d56	0x41404e1b
0x50787f70	0x0667035a
0x50787f70	0x41404e1b
0x537e727f	0x0667035a
0x537e727f	0x41404e1b
0x5470717b	0x41404e1b
0x6564095e	0x0667035a
0x77360a19	0x41404e1b

Unknown in Word 2007, PE in Writer 2010

Major Hashes

Other Neat Profiles (About 2000 Total)

8f854301-	PE		U
6b884b93-	PE		U
278b6591-	PE		U
4544d082-	PE		U
21296171-	PE		U
847ebae2-	PE		U
21296171-	PE		U
92742361-	PE		U
bc24a2bd-	PE		U
1775721d-	PE		U

	U	0599f01b-	E	E	E
	U	7885bd51-	E	E	E
	U	8765523c-	E	E	E

"Gap Years" – Unknown in SW2003, PE in SW2010

"Welcome To The Club"

Potential Issues

- Targeting an area already being swept by defenders
  - Code may look better than it actually is, because defenders are securing stuff exposed by these very same methods
  - This is ultimately the issue with *all* benchmarks
    - The only question is whether these bugs are representative of bugs that need to get fixed
    - Since these are the easiest flaws for an attacker to find, maybe this is OK

What about the fuzzer itself?

- Bitflipping only works well against file formats that are tightly packed
  - Most new formats are all textual
  - Also require complex grammars
- This is a first attempt, we intentionally didn't want to pack too much intelligence in
  - Code coverage
  - Automatic grammar extraction
  - Automatic segmentation of file formats
  - Manual generation of files
  - Integration with memory tracing

Operational Conclusions

- Bugs aren't rare
- Bugs aren't hard to find
- Cross Sectional Findings
  - Major document platforms may have bugs, but so do their competitors
- Longitudinal Findings
  - Everybody's code is getting better
  - Nobody's code is perfect
  - **Run the latest version of everything!**

Conclusion

- Has software quality improved over the last ten years?
  - Conclusion: For the set of formats tested, we find an unambiguous reduction in the number of failures, particularly when those failures show signs of being security-impacting.
- What next?
  - Better fuzzing
  - **Releasing of data!**

Oh Yeah, Data!

- We are immediately releasing for analysis the summary data from the fuzz run
  - There's lots of interesting meat to chew on / visualize
  - Go to www.fuzzmark.com for more
- We are *not* dropping 0day on anyone
  - Will provide the vendors with all the test cases they want
  - If you want the test cases, or even the stack traces, they'll have to give us permission ☺

Thanks for all the fish!

Adam Cecchetti
adam@dejavusecurity.com
Mike Eddington
mike@dejavusecurity.com

Dan Kaminsky
dan@doxpara.com

http://dankaminsky.com

http://dejavusecurity.com
http://peachfuzzer.com

http://tinyurl.com/cansecfuzz

Déjà vu Security

DKH Inc.