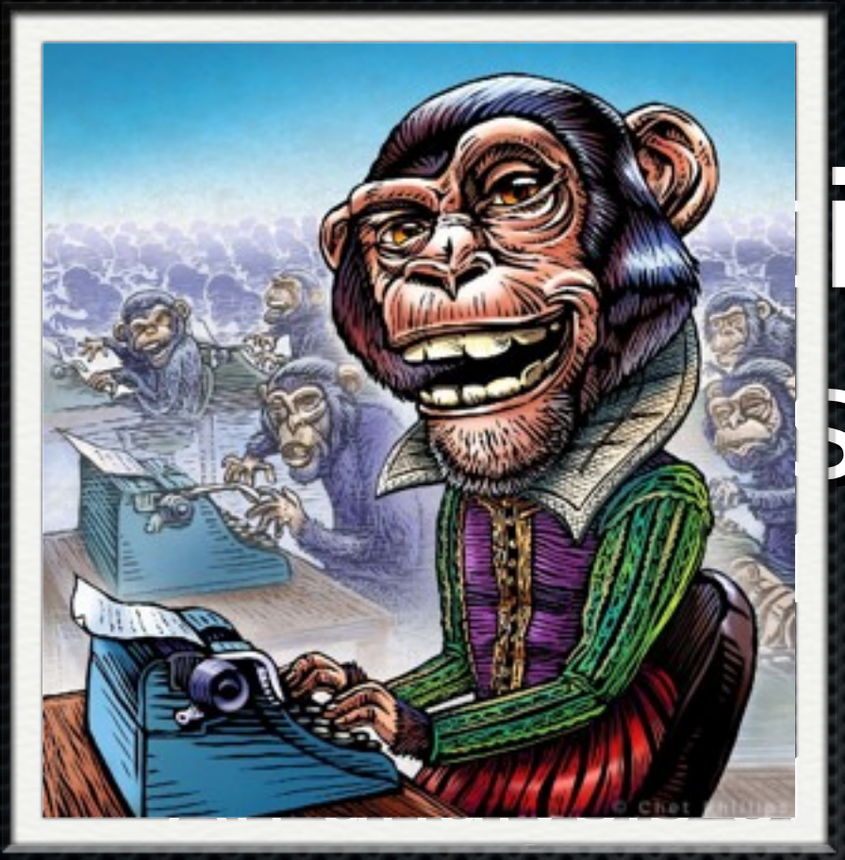


ing an Army of
s

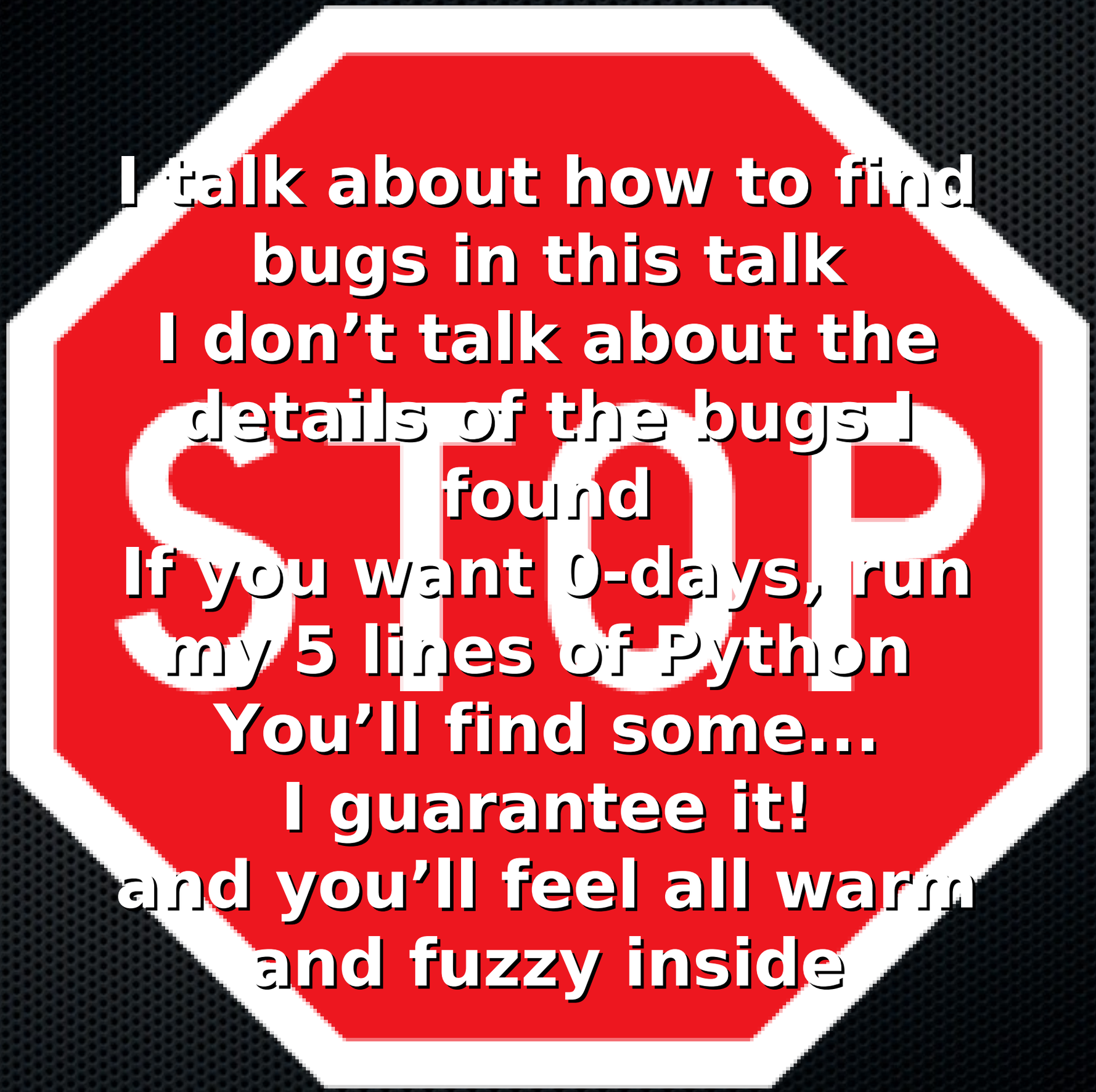
fuzzing 4 products with 5 lines of

fuzzing 4 products with 5 lines of Python

Charlie Miller

Independent Security Evaluators

cmiller@securityevaluators.com


I talk about how to find bugs in this talk
I don't talk about the details of the bugs I found
If you want 0-days, run my 5 lines of Python
You'll find some...
I guarantee it!
and you'll feel all warm and fuzzy inside

Who I am

- First to hack the iPhone, G1 Phone
- Pwn2Own winner, 2008, 2009
- Author
  - Mac Hackers Handbook
  - Fuzzing for Software Security Testing and Quality Assurance
- Media whore

Overview

- The fuzzing setup
- Fuzzing PDF’s, Preview and Adobe Acrobat Reader
- Fuzzing PPT’s, OpenOffice and MS PowerPoint
- Fuzzing “truths” revealed

About this talk

- Most fuzzing talks take one of two forms
  - I fuzzed and found this/these bugs
  - Here is a new, smarter way to fuzz
- These talks are about success, but real fuzzing is about failure, *i.e.* most test cases ***don't*** crash the target
- Very few talks that give realistic pictures of actual fuzzing
  - By sharing results, both positive and negative, we can learn about fuzzing and improve our techniques

Other talks to check out

- **Fuzz by Number**, Charlie Miller, 2008, http://cansecwest.com/csw08/csw08-miller.pdf
- **!exploitable and Effective Fuzzing Strategies as a Regular Part of Testing**, Jason Shirk, 2009, http://dragos.com/psj09/exploitable%20and%20Effecti
- **Effective Fuzzing Strategies**, David Molnar and Lars Opstad, 2010, http://www.cert.org/vuls/discovery/downloads/CERT-pre

Questions to ponder

- How many crashes can you expect?
  - How many of these are unique?
  - How many are “exploitable”?
- How important is the initial file when fuzzing?
- Are some bugs harder to find than others?
- How do post analysis tools compare?
- When have you fuzzed enough?
- How hard do various vendors fuzz and how many bugs do they find?

A Historical Perspective

- Microsoft Windows Vista File Fuzzing effort
  - 15 months, 350mil iterations, 250+ file parsers
  - ~1.4mil iterations per parser (on average)
  - 300+ issues fixed
- This talk
  - 3 months, 7mil iterations, ~4 parsers
  - ~1.8m iterations per parser (on average)
- However, quality is more important than quantity
  - My quality is purposefully very poor, should find much less than MS!

The Fuzzing Setup

Fuzzing types

- Dumb fuzzing (mutational)
  - Take a good input (file/packet/command line/etc) and add anomalies to it
  - Very easy to conduct
- Smart fuzzing (generation based)
  - Create invalid inputs from “scratch”, *i.e.* RFC, RE
  - Very hard, but explores every detail of protocol

Compromise for the lazy

- Dumb fuzzing with lots of different initial files
  - Single dumb fuzzing session will only fuzz the protocol ‘features’ present in the initial file
    - With enough initial files, hopefully you can fuzz all the ‘features’
- Still are screwed by things like CRC, compression, *etc.*

Selection of initial files

- Download every file you can find on the Internet
- Find the minimal subset that has the same code coverage as the large set
  - Example: PDF
    - Found 80,000 PDF’s on Internet
    - Used Adobe Reader + Valgrind in Linux to measure code coverage
    - Reduced to 1,515 files of ‘equivalent’ code coverage
    - Same bang as fuzzing all 80k in 2% of the time

The 5 lines of Python

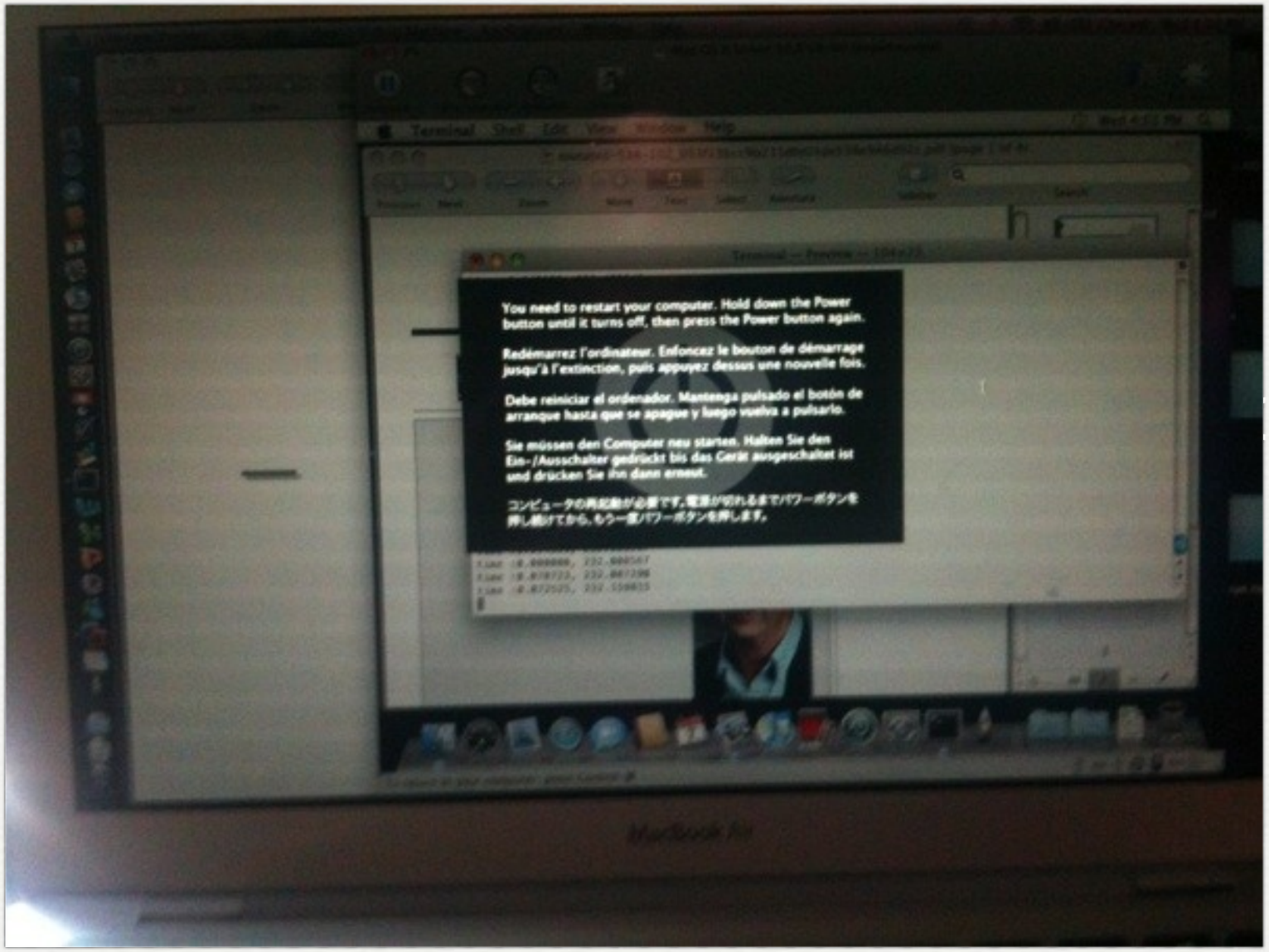
- Just change random bytes to random values
- Don’t insert bytes, remove bytes
- Easiest (dumbest) conceivable way to fuzz
- Shouldn’t find any bugs...

```
numwrites=random.randrange(math.ceil((float(len(buf)) / FuzzFactor)))+1for j in
range(numwrites):rbyte = random.randrange(256)rn =
random.randrange(len(buf))buf[rn] = "%c"%(rbyte);
numwrites=random.randrange(math.ceil((float(len(buf)) / FuzzFactor)))+1for j in
range(numwrites):rbyte = random.randrange(256)rn =
random.randrange(len(buf))buf[rn] = "%c"%(rbyte);
```

Other details

- Ran it in a parallelized way using my fuzzing framework, Tiamat
- Used 1-5 Mac OS X computers, some of them virtual
  - Including 2 Pwn2Own prizes!
- Open/Closed files using AppleScript
- Monitored CPU activity to know when to launch next file
- Ran fixed number of iterations of each file (2000 or 1200)
  - Estimated for 3 week runs
- Recorded repeatable crashes (either native or with glibmalloc)

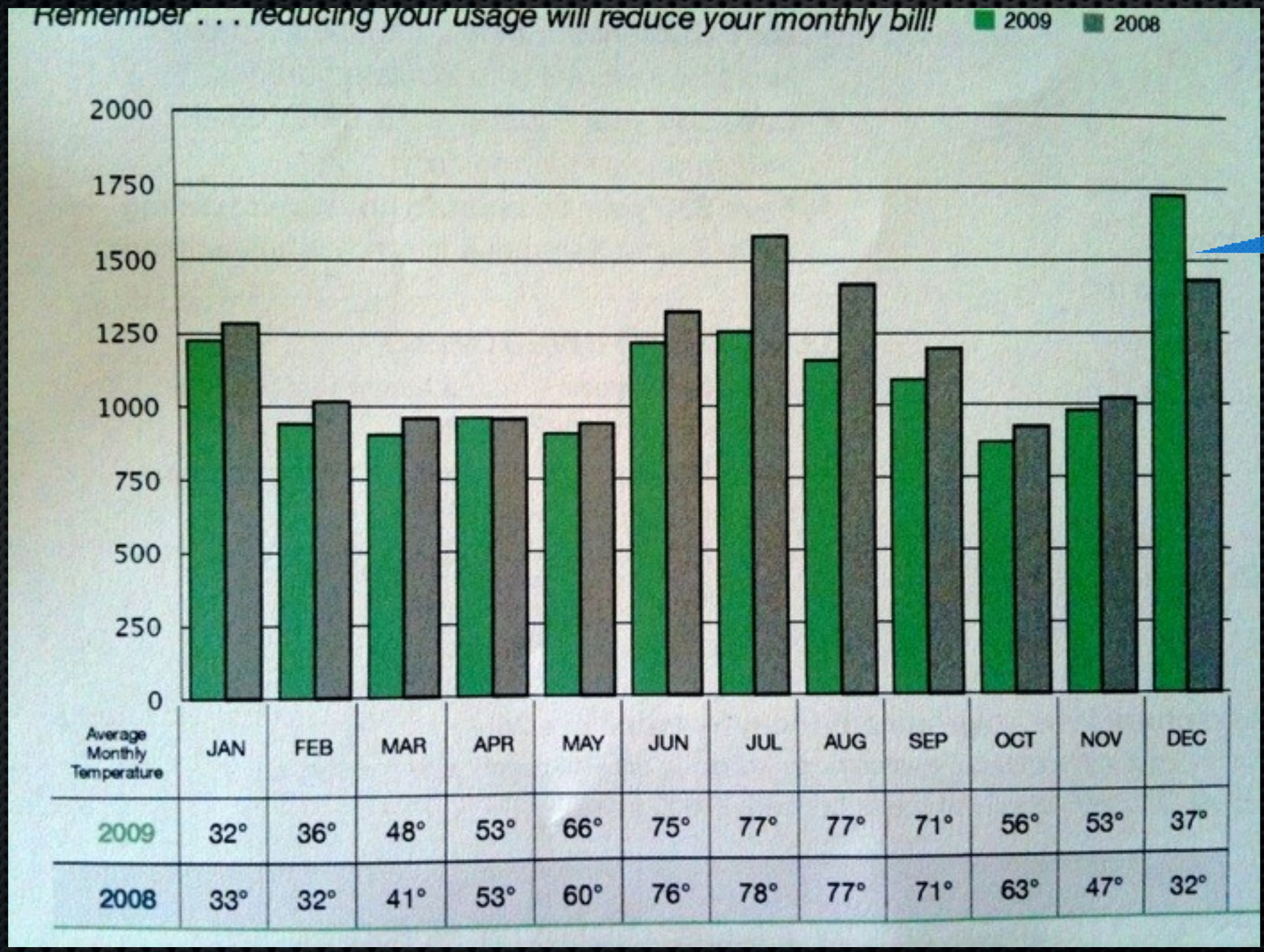
Originally, this slide was my whole talk...


You need to restart your computer. Hold down the Power button until it turns off, then press the Power button again.
Redémarrez l'ordinateur. Enfoncez le bouton de démarrage jusqu'à l'extinction, puis appuyez dessus une nouvelle fois.
Debe reiniciar el ordenador. Mantenga pulsado el botón de arranque hasta que se apague y luego vuelva a pulsarlo.
Sie müssen den Computer neu starten. Halten Sie den Ein-/Ausschalter gedrückt bis das Gerät ausgeschaltet ist und drücken Sie ihn dann erneut.
コンピュータの再起動が必要です。電源が切れるまでパワーボタンを押し続けてから、もう一度パワーボタンを押します。

thly

o not

The vendors could at least pay my power bill!



Average Monthly Temperature	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
2009	32°	36°	48°	53°	66°	75°	77°	77°	71°	56°	53°	37°
2008	33°	32°	41°	53°	60°	76°	78°	77°	71°	63°	47°	32°

Tools used

- libgmalloc: uses guard pages to find when heap overflows first occur (like libefence), OS X
- CrashWrangler: Apple tool to bin crashes and determine exploitability, OS X
- memcheck: Valgrind tool which simulates program execution and records invalid memory operations, Linux and OS X
- !exploitable: MS tool used to bin crashes and determine exploitability, Windows

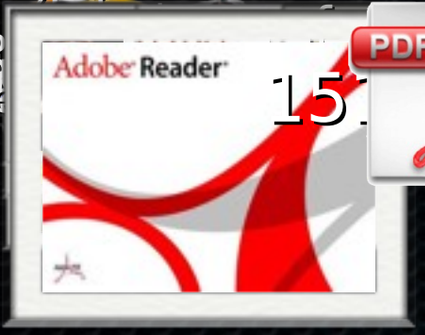
Final thoughts: Fuzzing as Filtering

- Fuzzing isn’t about creating and running test case, it’s about filtering
- Start with a ton of test cases
- Filter those to the ones that cause a crash
- Filter those to the ones that represent unique crashes
- Filter those to the ones that are exploitable

Adobe Acrobat Reader (PDF) (PDF)

Reader stats

- Reader 9.2.0
- 3,036,000 test cases tested
- Maximum test cases/min 132
- Minimum test cases/min 7



151 [illegible] 66,00[illegible]zzed files

3 exploitable

34 unique

20 unique*

2 Process terminated

*valgrind failed frequently
cause acrobat has problems even with clean files

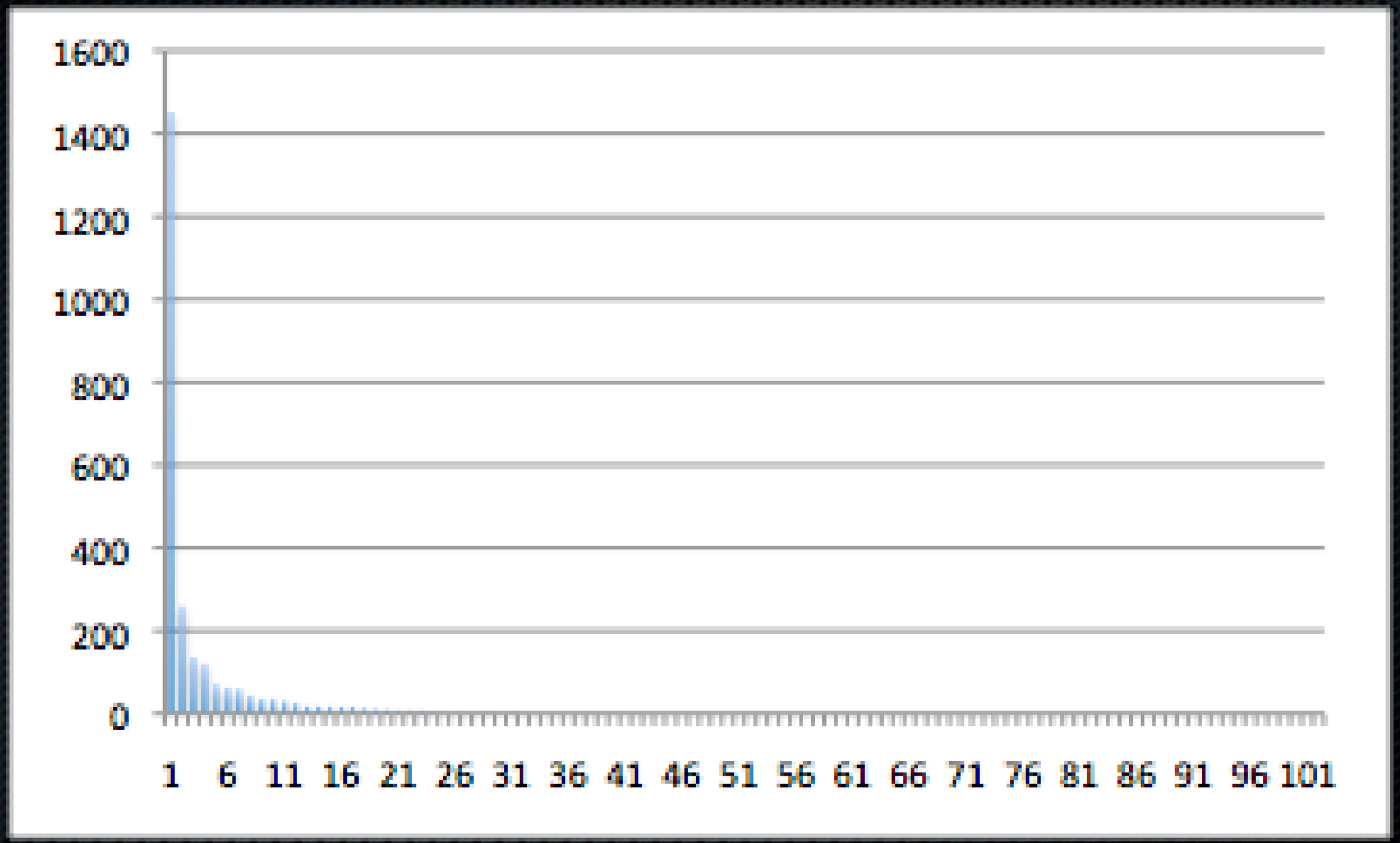
Points of interest

- Acrobat Reader, under valgrind, has lots of errors
  - Examples: mismatched malloc/free
  - Don’t know how it runs normally….
    - Ignoring these errors makes you miss lots of crashes
- 100 unique EIPs, around 20-40 repro’d with binning tools
- 3-4 exploitables, according to tools
- Disagreement about what files cause what crashes and which are exploitable

Comparisons of exploitables

Crash	***!exploitable***	***Crashwrangler***	***Valgrind***
Crash 1	Exploitable	is_exploitable=yes	Process terminated
Crash 2	Exploitable	is_exploitable= no	Valgrind failed
Crash 3	Exploitable	is_exploitable= no	Uninitialized variable
Crash 4	Exploitable	is_exploitable= no	Process terminated
Crash 5	Probably Exploitable	is_exploitable= no	Valgrind failed
Crash 6	Probably Exploitable	is_exploitable= no	Valgrind failed
Crash 7	Probably Exploitable	is_exploitable= no	Valgrind failed
Crash 8	Probably Exploitable	is_exploitable= no	Valgrind failed
Crash 9	Probably Exploitable	is_exploitable= no	Invalid write
Crash 10	Probably Exploitable	is_exploitable= no	Uninitialized variable
Crash 11	Probably Exploitable	is_exploitable= no	Invalid write
Crash 12	Probably Exploitable	is_exploitable= no	Invalid write
Crash 13	Probably Exploitable	is_exploitable= no	Invalid write
Crash 14	Probably Exploitable	is_exploitable= no	Valgrind failed
Crash 15	not on win	is_exploitable=yes	Uninitialized variable
Crash 16	not on win	is_exploitable=yes	Uninitialized variable

Number of times each crash occurred



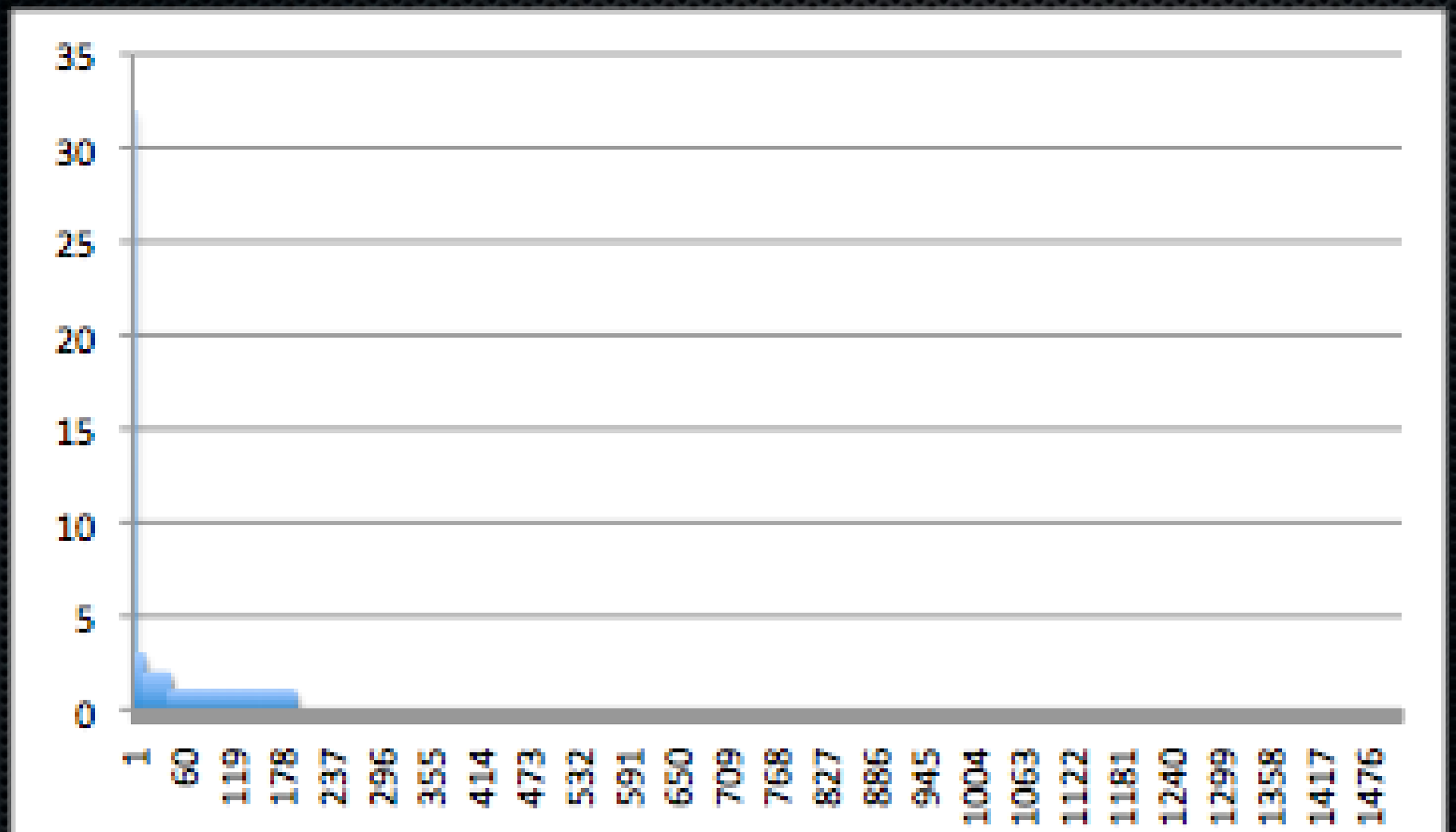
Reader crash rarity info

- 100 different crashes
- 57 were found exactly once
  - Either rare or lots of manifestations of one bug
- 81 were found less than 10 times
  - Rare?
- 7 were found more than 60 times
  - Common bugs
- 4 were found more than 100 times
  - Very common bugs
- One crash found 1452 times
  - This one crash is responsible for 56% of crashes in the testing

Choice of initial file

- 1515 different files
- Crashes at 100 different EIP’s
- All crashes occurred when fuzzing only 192 files
  - No crashes from 87% of initial files!!!
- All files but one found between 1-3 crashes
- 1 file found 32 crashes (all but one with invalid EIP)
  - Probably one (really nasty) bug
  - These bugs all coalesced when used libgmalloc

Crashes found per file



To find an exploitable

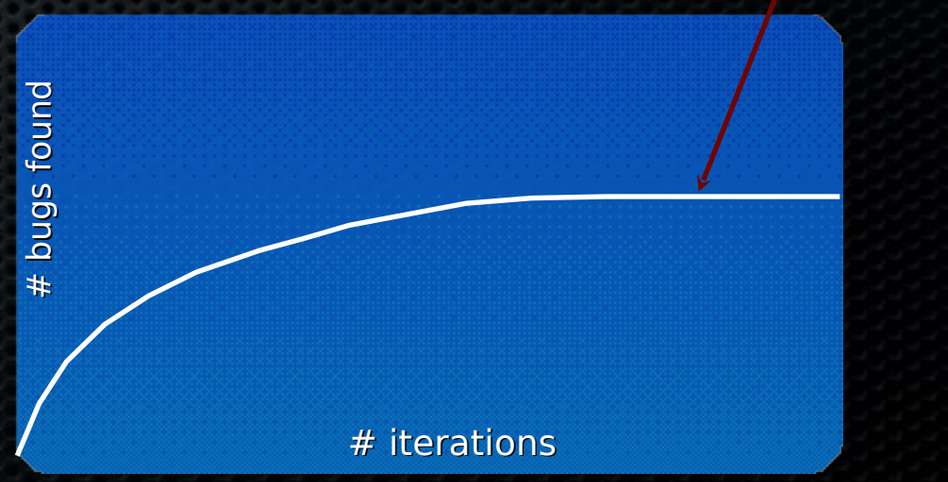
- For the 4 !exploitables, lets see info about other crashes that crashed at the same EIP
  - Crash 1: 2 files, each crashed once or twice
  - Crash 2: 2 files, each crashed there once
  - Crash 3: 2 files, each crashed there twice
  - Crash 4: 42 files crashed there from 1 to 63 times
- Earliest test case to find one of these was the 486th iteration
- Last was the 1923rd iteration (of 2000)

More on finding exploitables

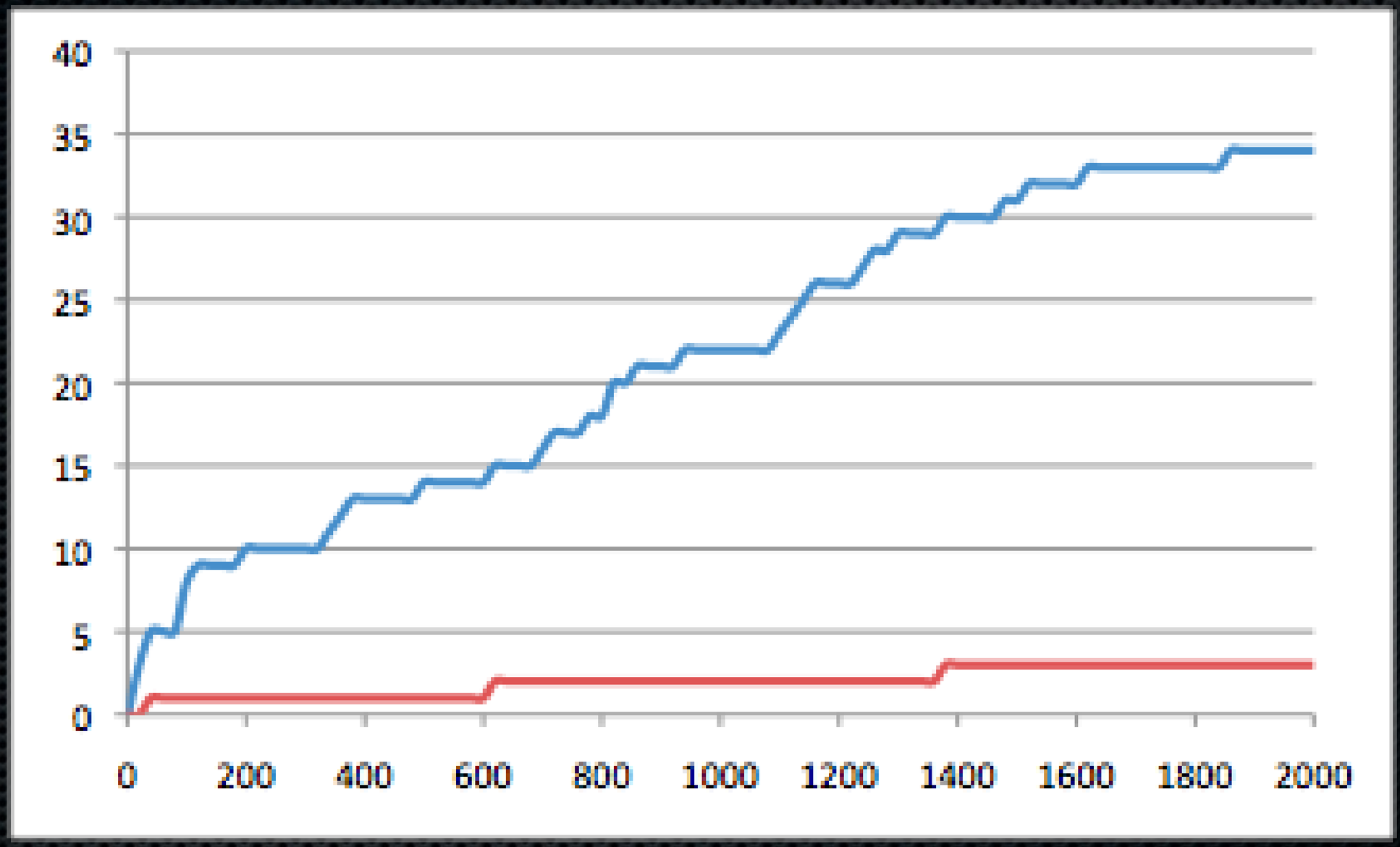
- Find the first 3 exploitables in 9 test cases out of 3 million
- Only 6 of 1515 files generate the first 3
- If you fuzz each file 500 times, you find 1 exploitable
  - 1000 times, you find 2
  - 1500 times, you find 3
  - 2000 times, you find 4
- What happens if you iterate 3000, 100000, 100000000?

How many iterations (theoretical)

- Run long enough, your fuzzer will find every bug (it is capable of finding)
- Presumably, this gets harder and harder
- End up with some idealized graph of iterations vs bugs found
- When this curve becomes sufficiently flat, stop fuzzing

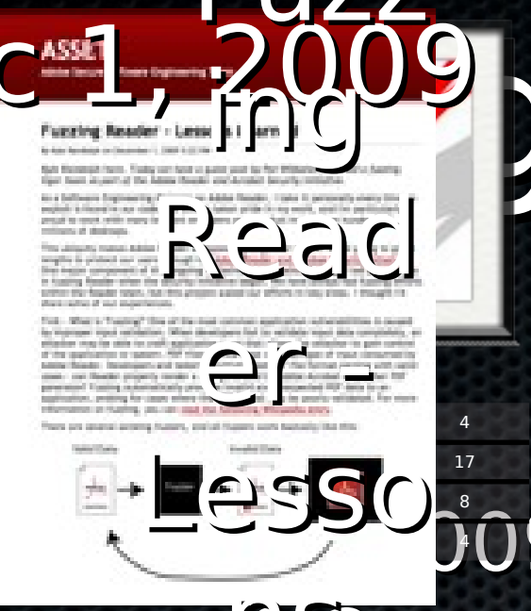


iterations to find crashes



More iterations would have probably found more bugs
(Curve isn’t flat yet)

“Fuzz



c 1, 2009

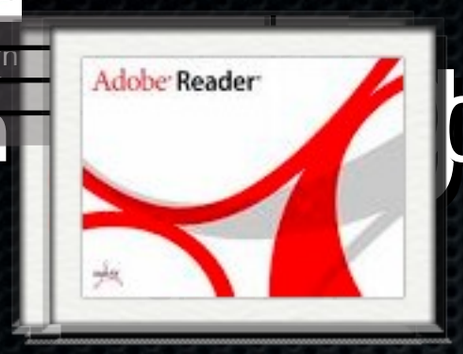
ing Reader-Lesso

gs Learned”

gress?

009

Ja press?



Probably not exploitable	4
Unknown	17
Probably exploitable	8
Exploitable	4

Oct 13, 2009

9.2

Preview (PDF) (PDF)

Preview General Info

- Default Mac OS X PDF viewer
  - Tested: Mac OS X 10.6.1
- These bugs show up in Safari too
- 2,790,000 test cases tested
- Maximum testcases/min: 160
- Minimum testcases/min: 4
- Total run time: Approximately 3 weeks

46 Use of
alized variables

(data)

2,720,000 fuzzed files

1056 “pro’d

281 unique

61 “exploitable”

1373 unique EIPs

157,337 crashes (5.6%)

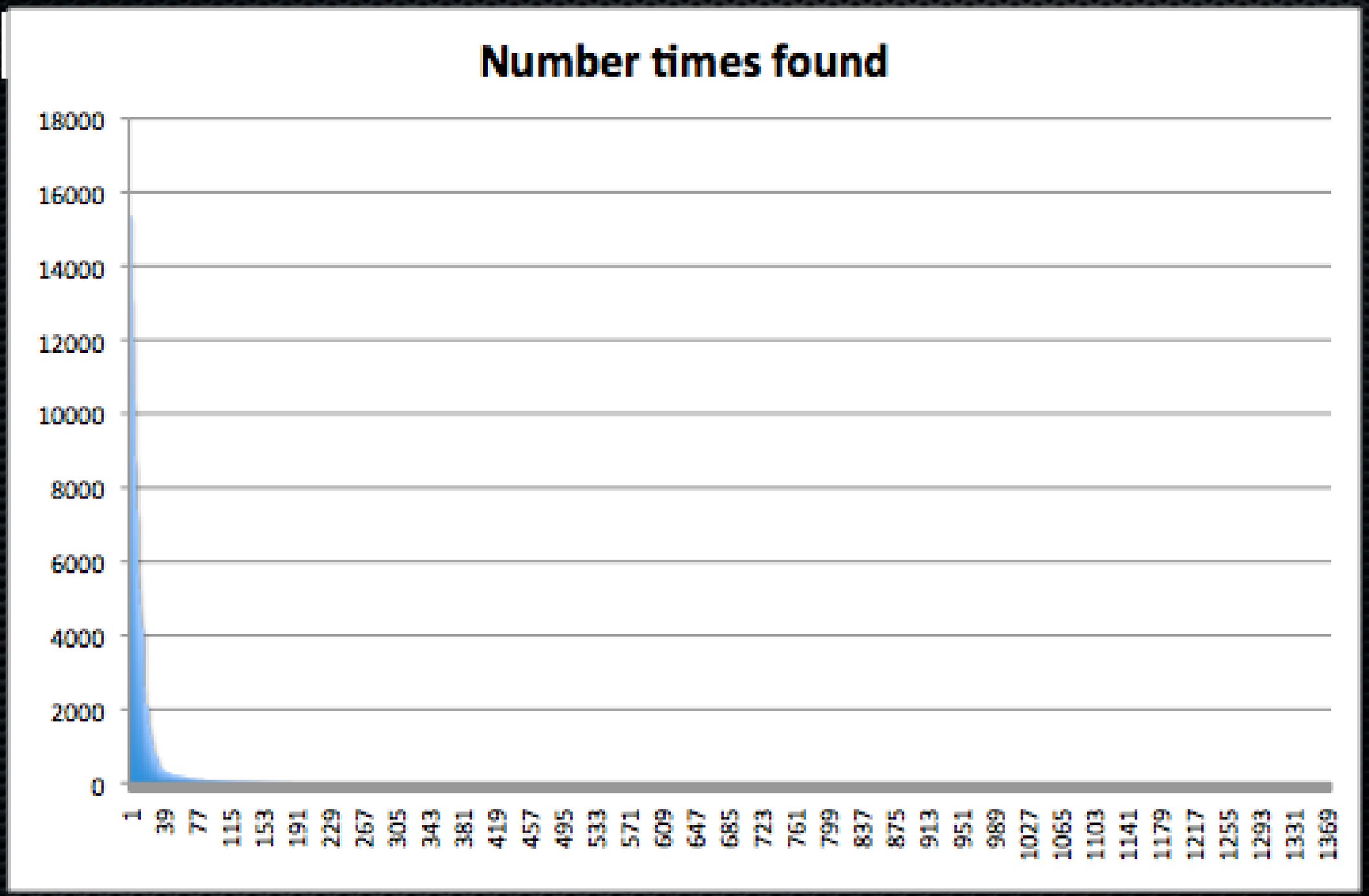
228 unique

36 Process terminated

Talking points

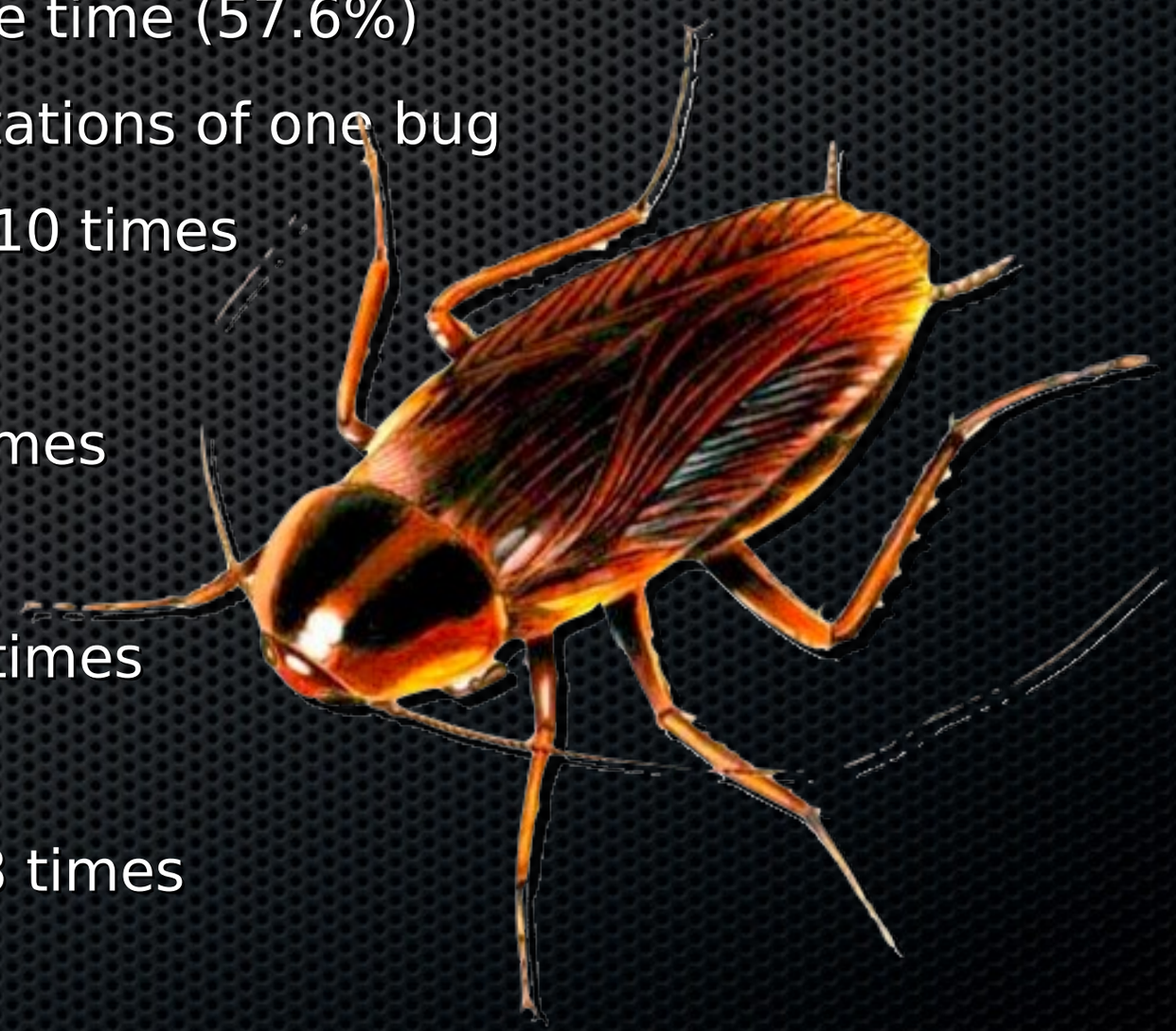
- Umm...they haven’t fuzzed this
- no !exploitable since Preview OS X only
- Around 250 unique crashes
- Around 60 exploitable
  - This is an overestimate, at least one bug manifests itself in lots of crashes and libgmalloc fails to bin it properly

Number of times each crash occurred (by EIP) (

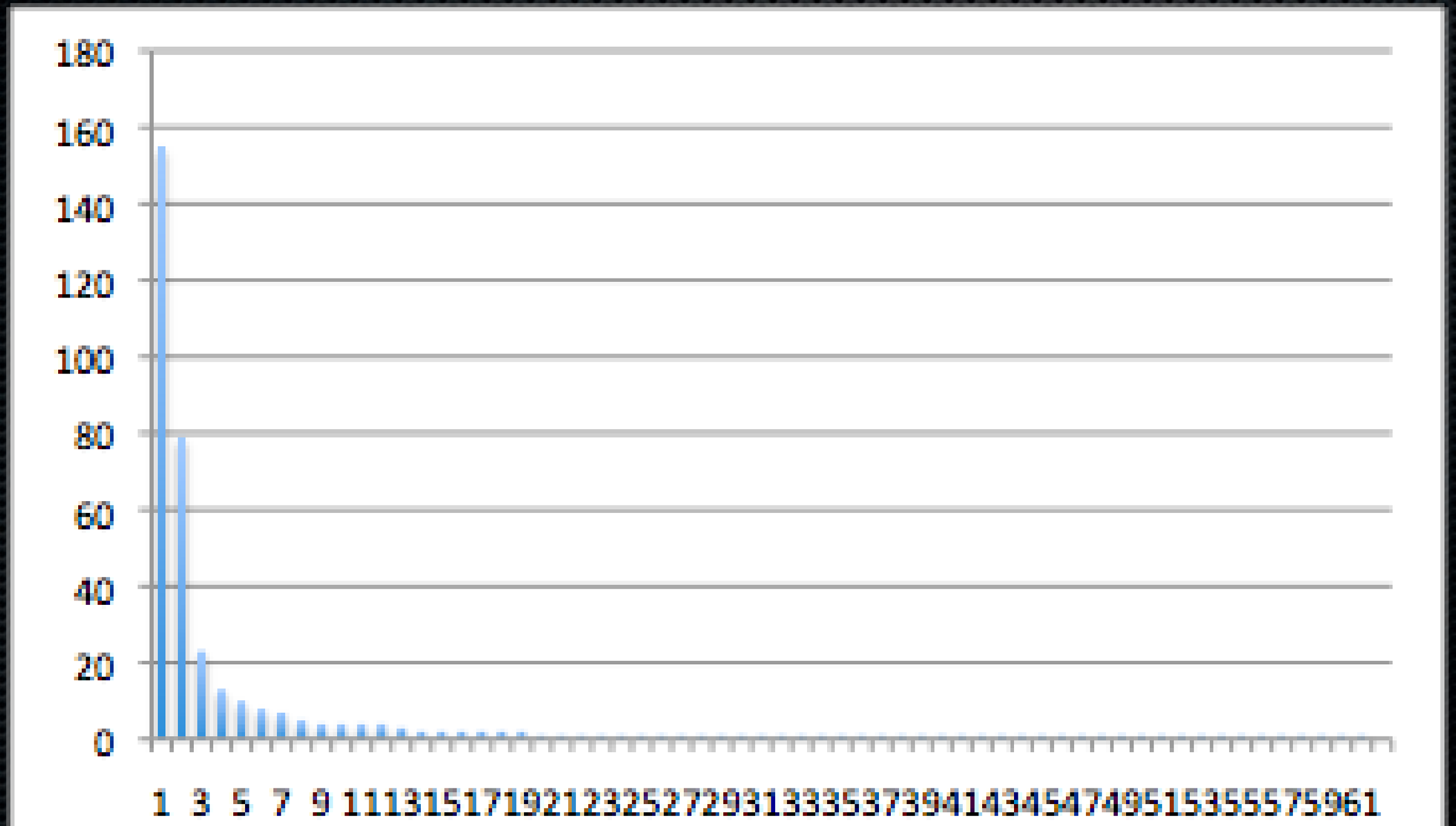


More crash rarity info

- Crashes at 1373 unique EIP’s
- 791 EIP’s were found exactly one time (57.6%)
  - Either rare or lots of manifestations of one bug
- 341 were found between 1 and 10 times
  - rare?
- 82 were found more than 100 times
  - Common bugs
- 26 were found more than 1000 times
  - Very common bugs
- One EIP found in crashes 15,368 times



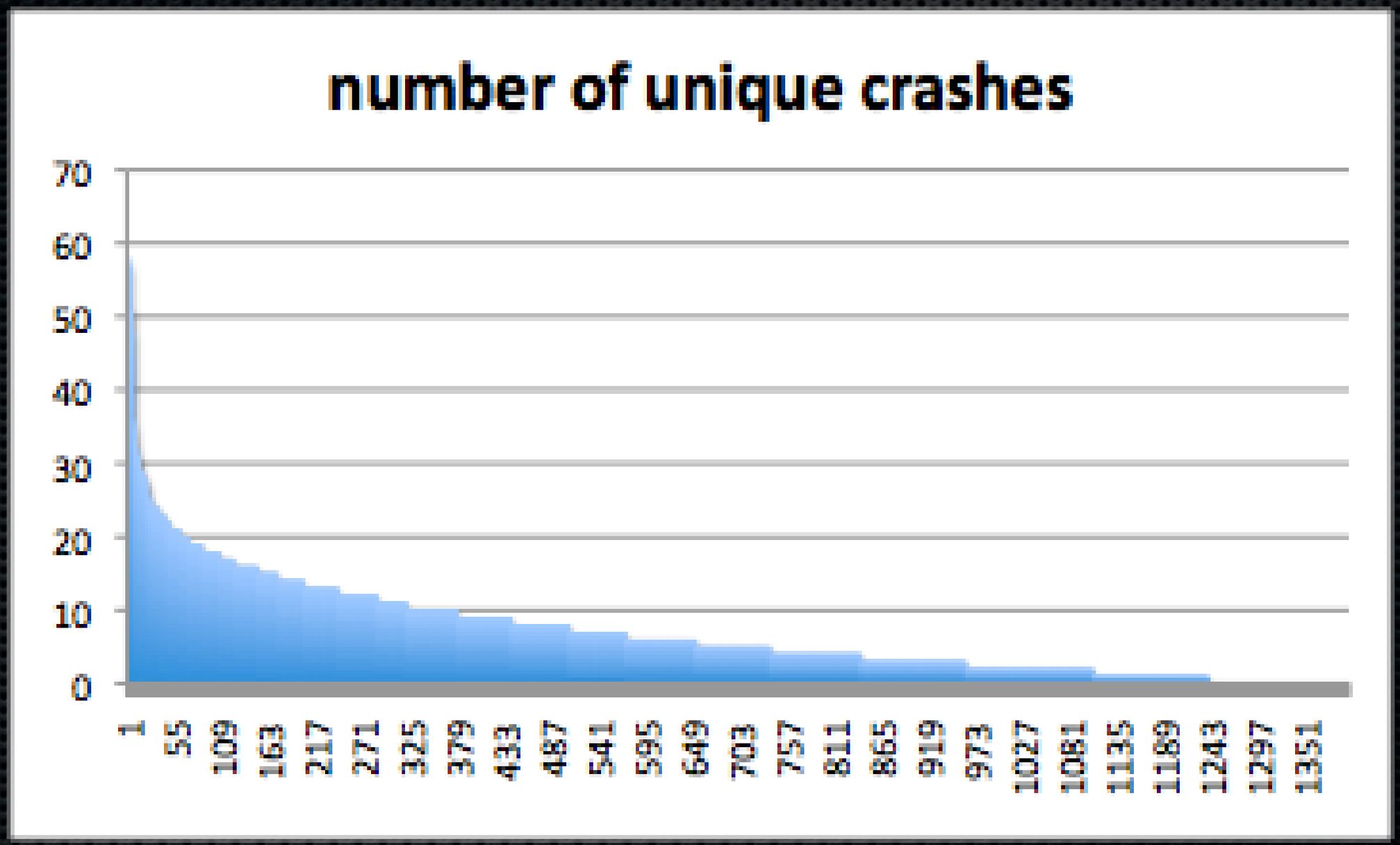
times exploitable crash occurred



“Exploitables”

- “exploitable” crashes at 61 EIP’s according to libgmalloc+crashwranger
- 1 EIP was found 155 times
- 42 were found only once
  - lots of rare ones or a few nasty ones
- 56 were found less than 10 times

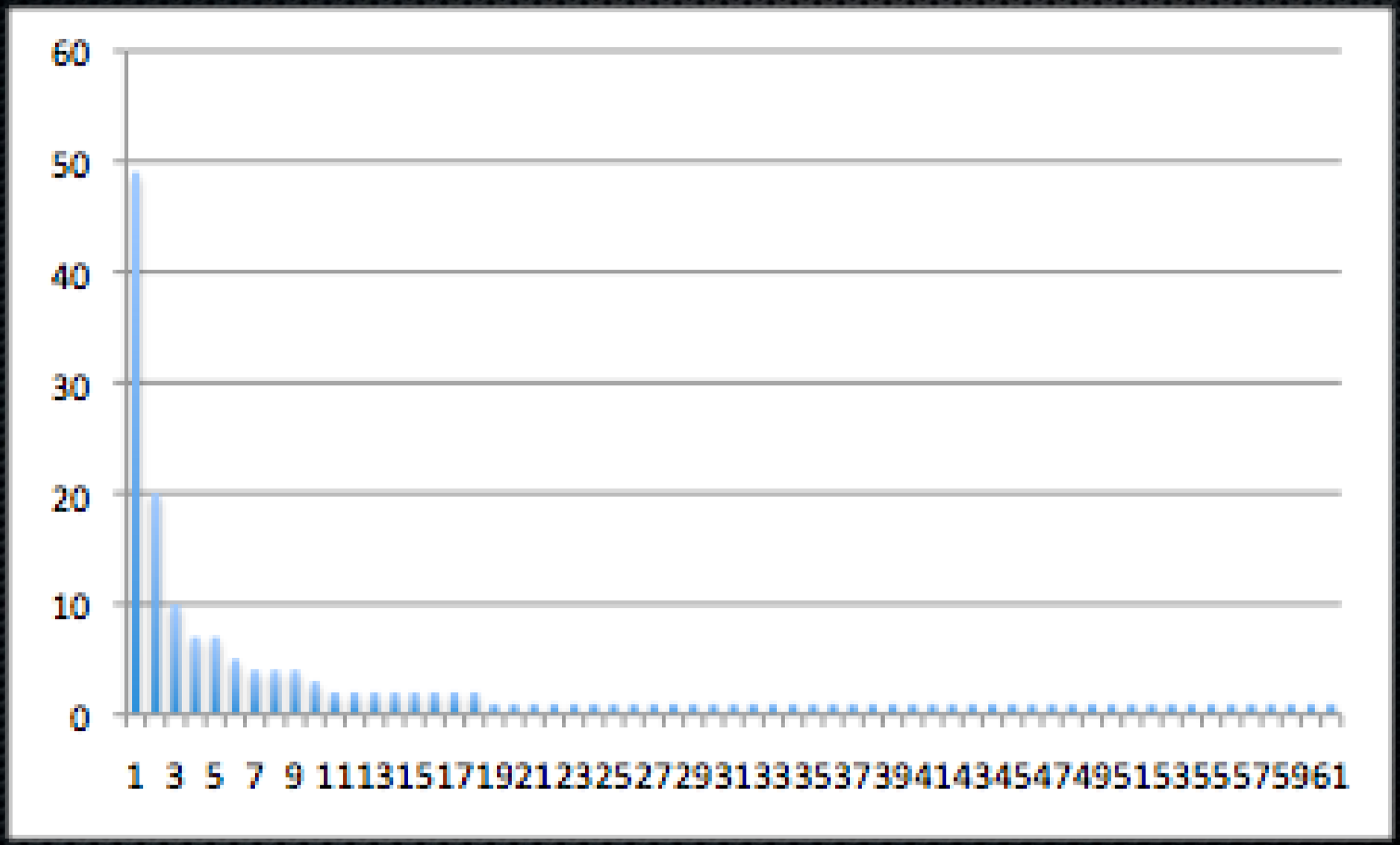
of crashes at EIP by initial file



File choices

- 1395 files
- 1 file found 58 “unique” crashes, by EIP
- 68 files (5%) found 20 or more different crashes
- 162 files (12%) found ***no*** crashes
- 440 files (31%) found 2 or fewer crash

Number of files which find each exploitable



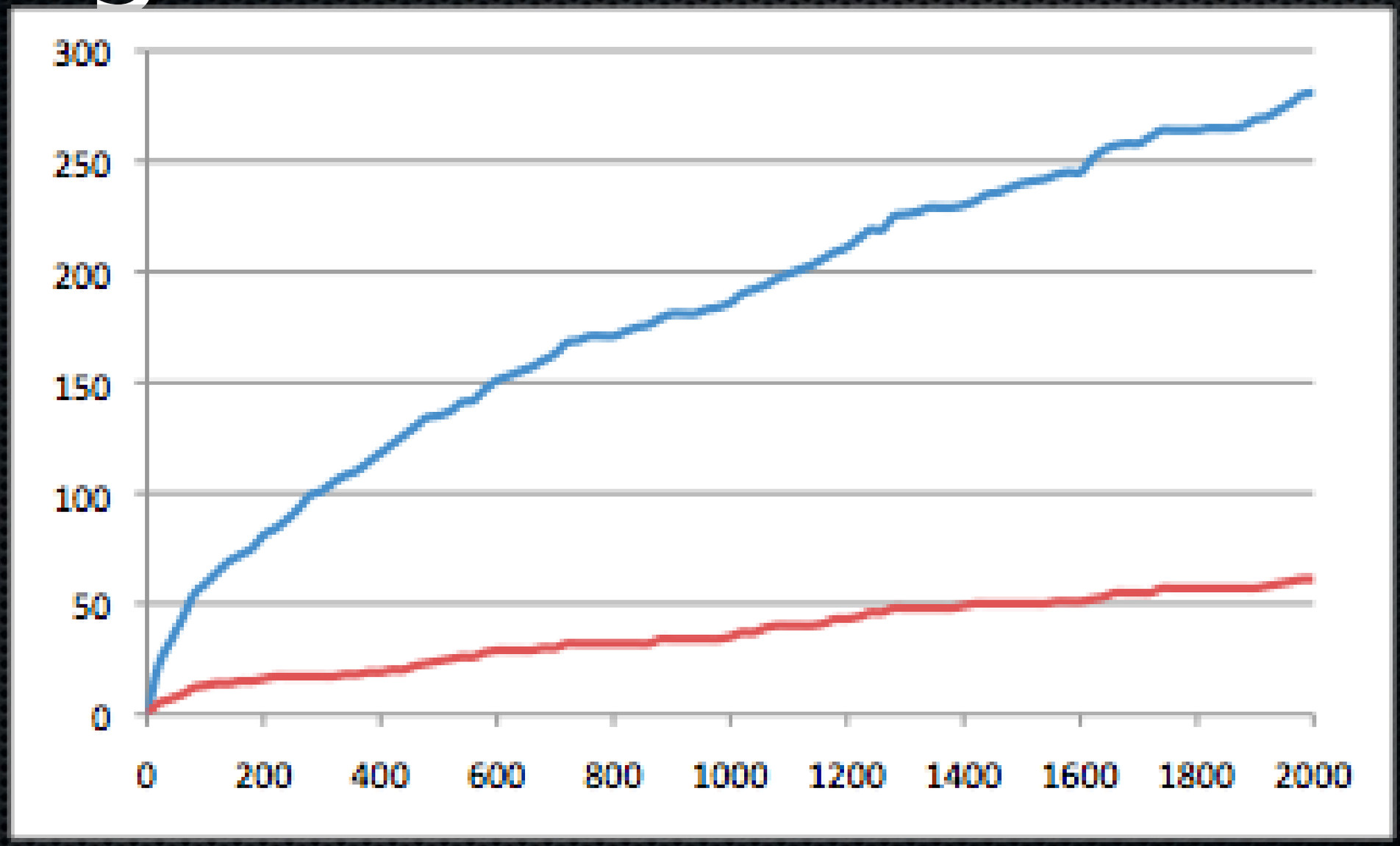
Files and exploitables

- 61 crashes exploitable (by EIP)
- 49 files found the most common exploitable crash
- Only 2 crashes were found by more than 10 starting files
- 42 (69%) crashes were found by exactly one starting file
- 50 (82%) crashes were found by at most two starting files

More on file choices

- These 1399 files were not randomly chosen, they are very special!
- Yet, even with these, almost a third find almost nothing
- So…If you randomly pick files to fuzz with, you probably won’t find any interesting bugs

Should have fuzzed longer



crashes and exploitables found by iteration

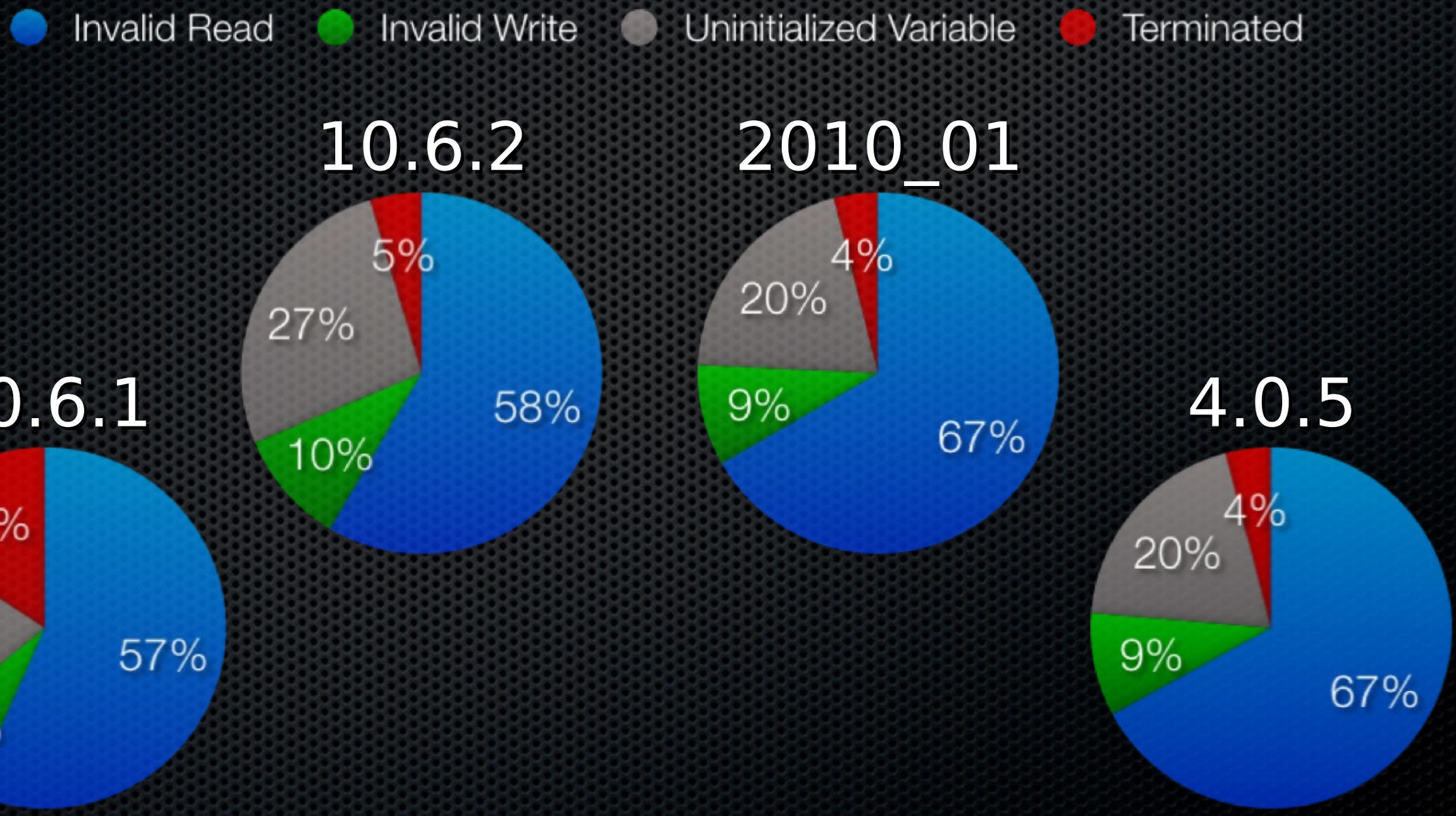
Fixes through time (unique by valgrind)

safari 4.0.5

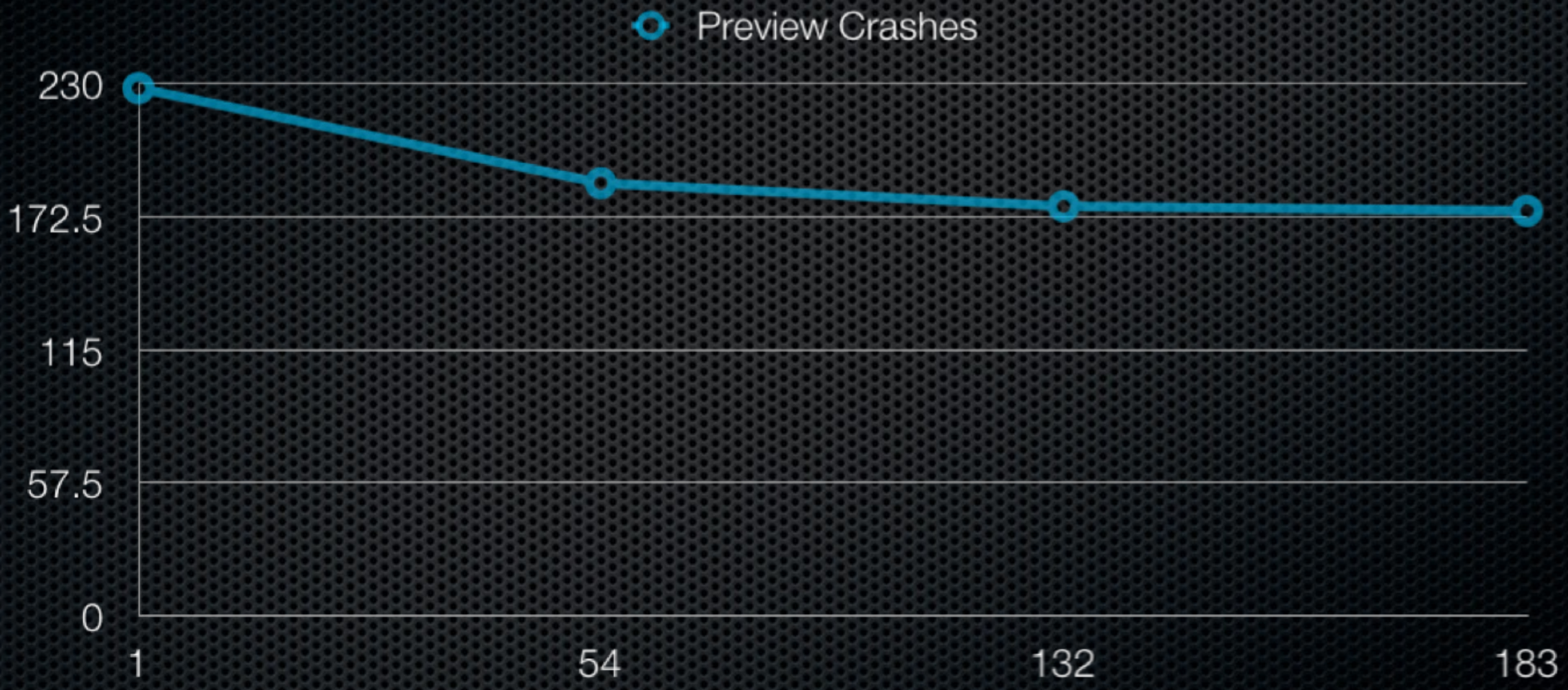
(by EIP)

(by EIP)

Fixes through time, by type

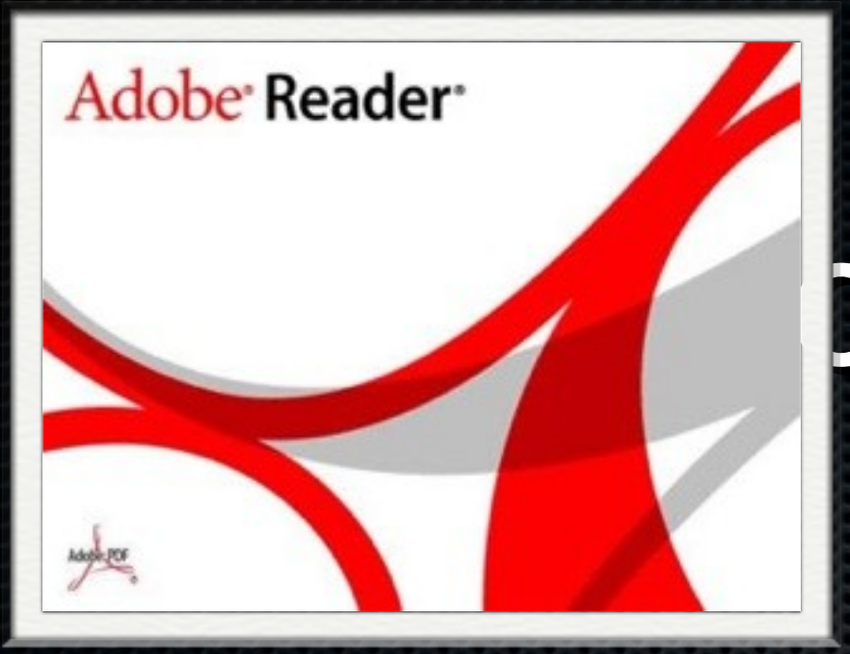


All bugs will be gone by...



according to linear regression all bugs will be fixed sometime in 2012

- iPhone 3.1.2, not jailbroken
- iPhone doesn’t have Preview, but MobileSafari will display PDF’s
  - Much of the complexity of PDF’s is ignored, e.g. fonts
- Recall Preview had 281 unique crashes (libgmalloc)
- 22 crashed MobileSafari, all at unique pc
  - 7.8% of crashes affected both
  - None of the corresponding Preview crashes were “exploitable”



owdown

100 crashes
30-40 unique
3-10 exploitable

1373 crashes
230-280 unique
30-60 exploitable

Open Office (PPT)
(PPT)

OpenOffice

- OpenOffice 3.1.1, impress
- 610,400 test cases tested
- Maximum testcases/min: 15
- Minimum testcases/min: <1
- Total run time: Approximately 3 weeks

10 exploitable

105 crashes
27 unique

186 crashes
36 unique

7 Process
terminated

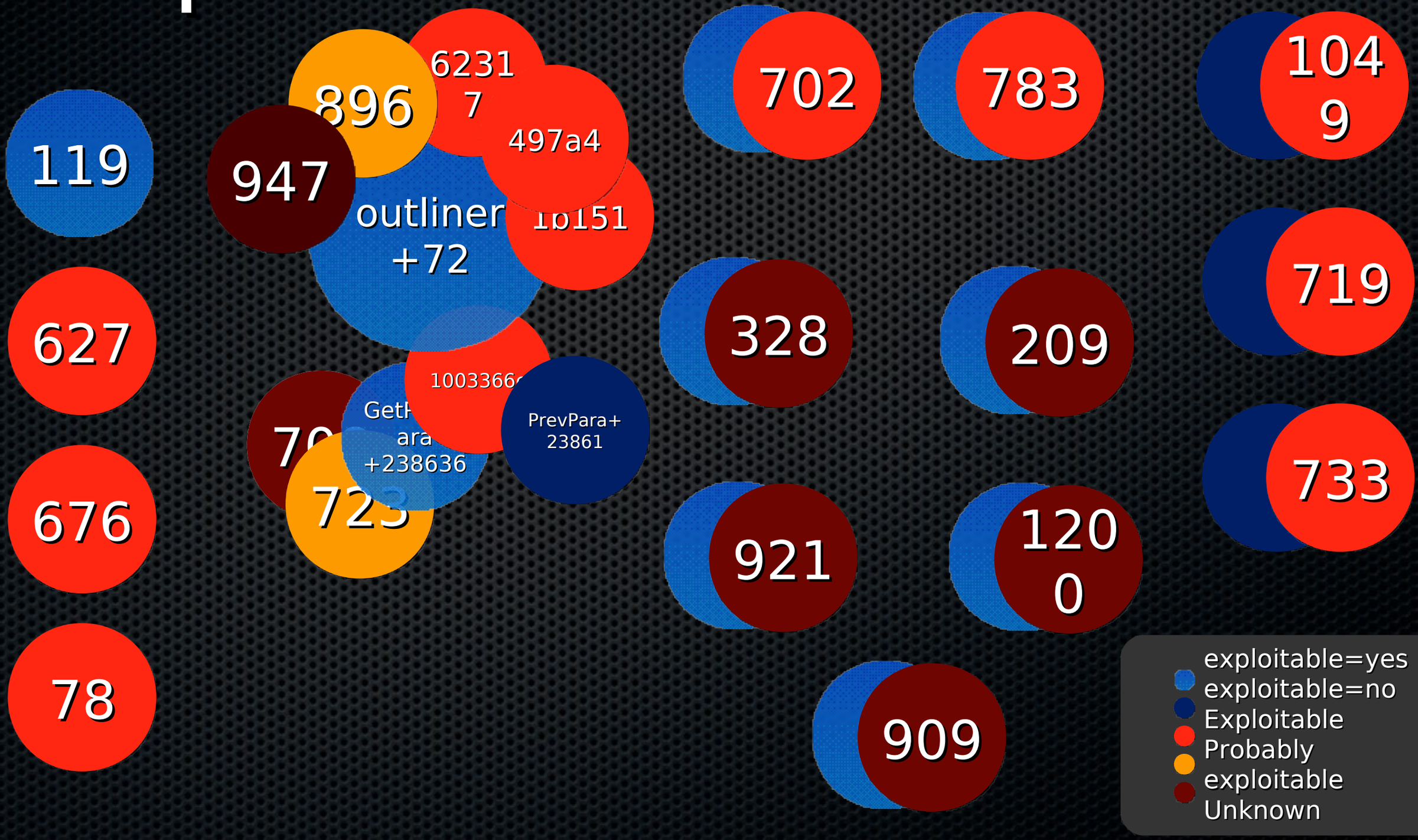
Some thoughts

- Around 200 crashes
- Don’t know why half don’t crash under libgmalloc
- Around 30-70 unique crashes
- Around 10-12 exploitable crashes, as reported by tools

Comparisons of exploitables

Crash	***!exploitable***	***Crashwrangler***	***Valgrind***
-921-	Unknown	is_exploitable=yes	Invalid read
-1200-	Unknown	is_exploitable=yes	Invalid read
-896-	Unknown, Probably, Exploitable	is_exploitable=yes	Invalid read, uninit
-723-	Unknown, Probably, Exploitable	is_exploitable=yes	Invalid read
-209-	Unknown	is_exploitable=yes	Invalid read
-328-	Unknown	is_exploitable=yes	Invalid read
-909-	Unknown	is_exploitable=yes	Invalid write
-702-	Exploitable	is_exploitable=yes	Invalid write
-783-	Exploitable	is_exploitable=yes	Invalid write
-119-	no crash	is_exploitable=yes	Invalid write
-1049-	Exploitable	is_exploitable=no	Terminated
-719-	Exploitable	is_exploitable=no	Invalid read
-733-	Exploitable	is_exploitable=no	Invalid write

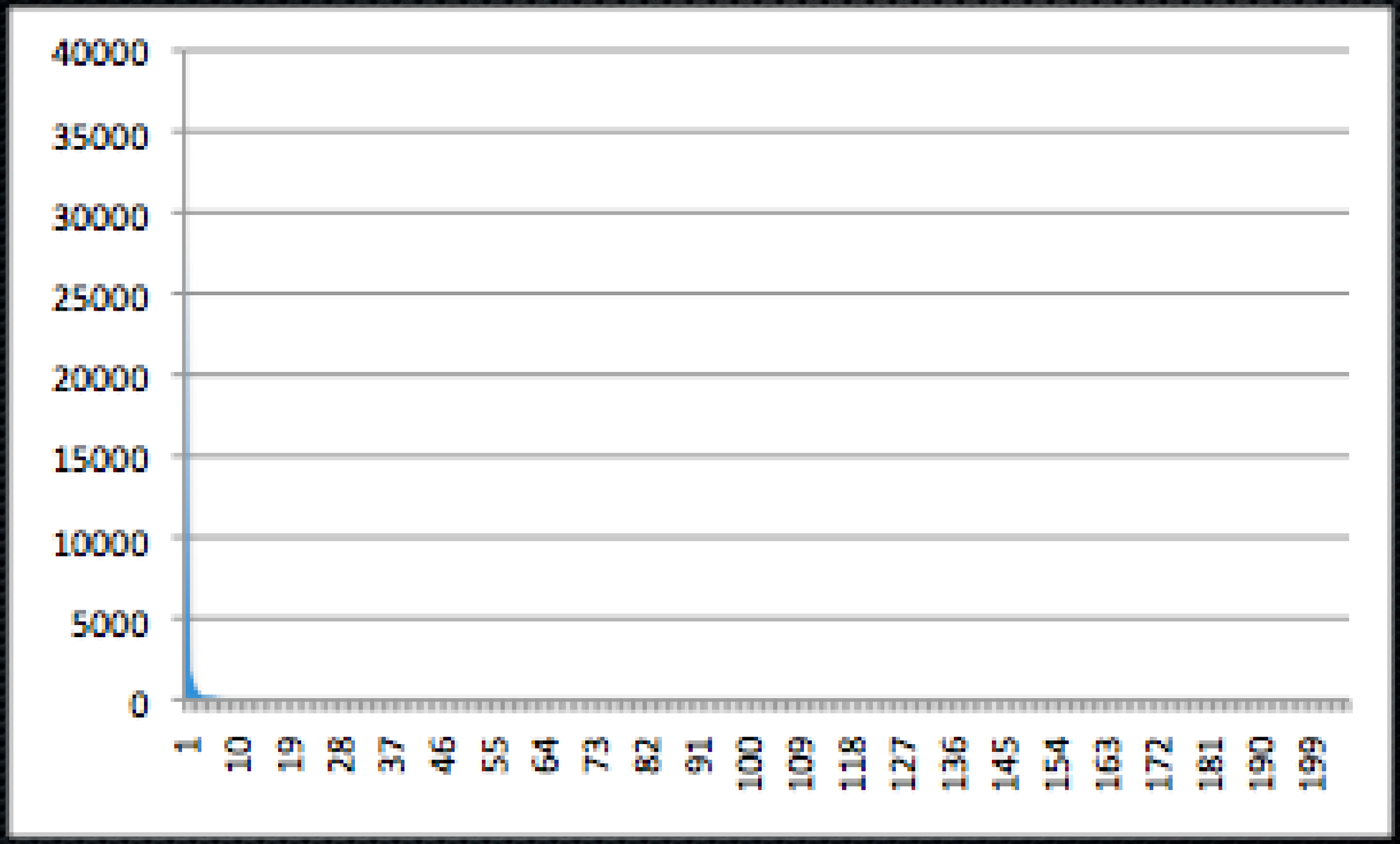
crash binning exploitables



Crash binning fail

- These 2 tools disagree more than they agree
- valgrind disagrees on the binning too...
- At least one (and possibly both) of these tools suck at binning crashes
- At least one (and possibly both) of these tools suck at determining exploitability

Crash rarity



Stupid outlier

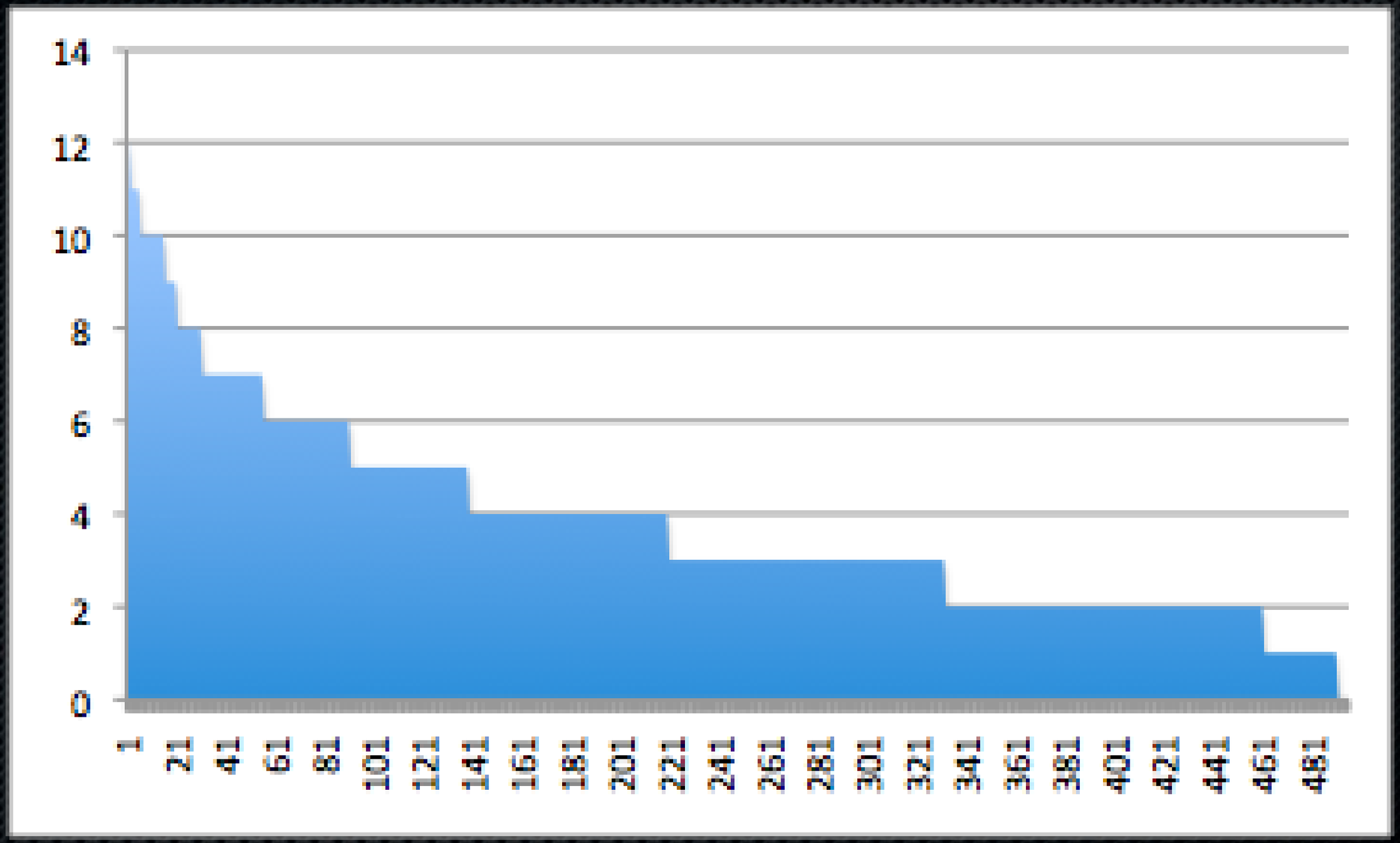
OO crash rarity

- 600,000 test cases, 205 different crashes
- 149 were found exactly once (73%)
- 186 were found less than 10 times (91%)
- 6 were found more than 200 times
- 2 were found more than 1800 times
- One crash found 36,288 times
  - This one crash is responsible for 90% of crashes in the testing

Choice of initial file

- 496 different files
- Crashes at 205 different EIP’s
- All but 5 files found at least one crash
- 2 files found 12 crashes
- Here choice of initial file doesn’t seem so important

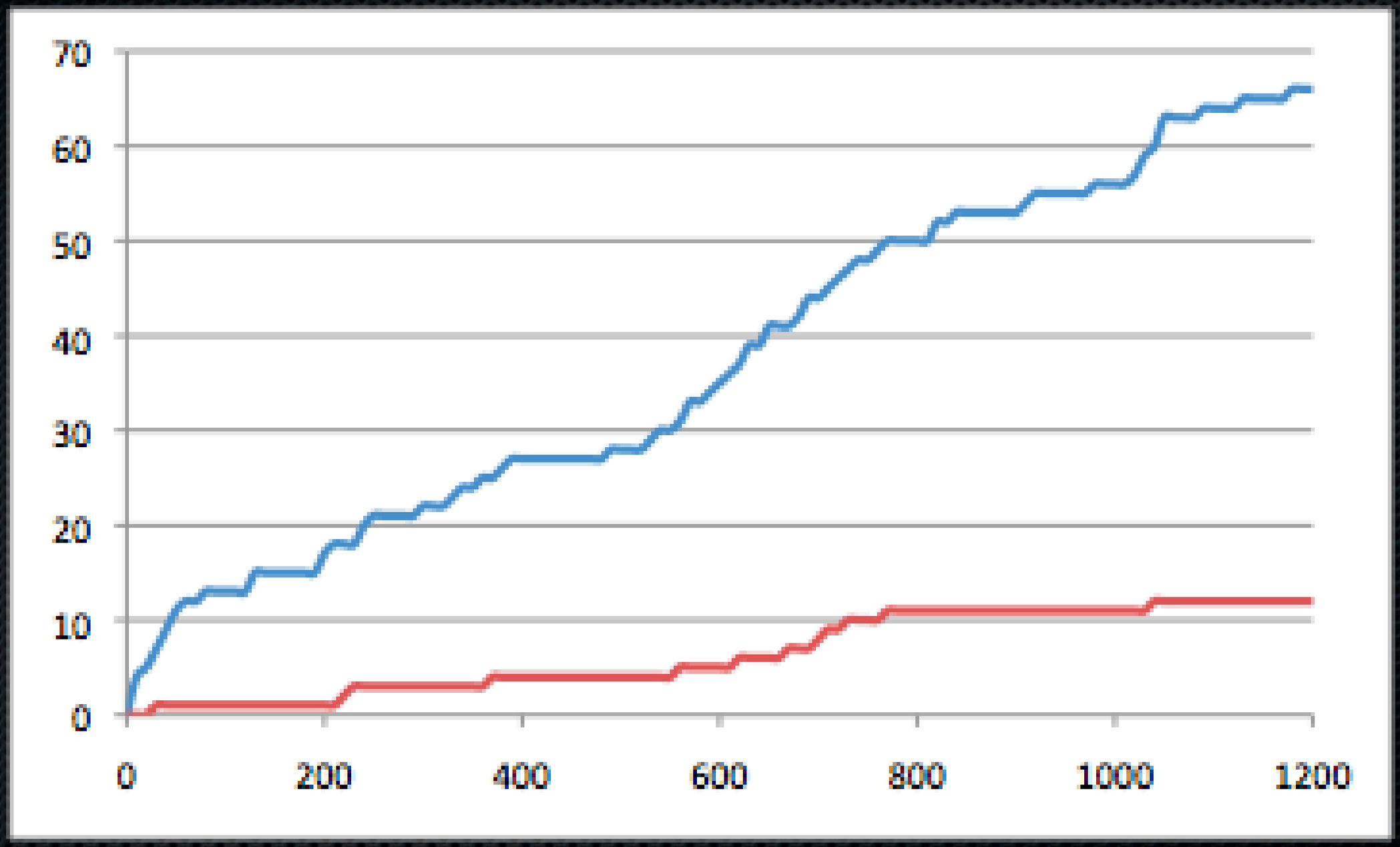
Crashes from initial file



Files to find exploitables

- 496 initial files, 12 exploitable crashes (! exploitable)
- One crash was found by 13 files (2.6%)
- 2 crashes were found by 3 files (0.6%)
- Rest were found by exactly one file (0.2%)
- Very rare to download a file, fuzz it, and discover exploitable bugs

Time to unique crash (! exploitable)

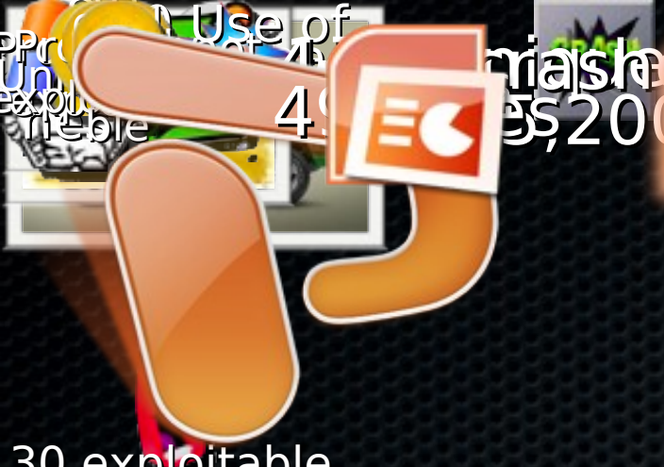


terations to find crashes (blue) vs. exploitable (re

Microsoft Office (PPT) (PPT)

MS Office PowerPoint

- MS PowerPoint 2008 for Mac, 12.2.3 (091001)
  - MS Office PowerPoint 2007 SP2 MSO (12.0.6425.1000) for !exploitable purposes
- 595,200 test cases tested
- Maximum testcases/min: 34
- Minimum testcases/min: 1
- Total run time: Approximately 3 weeks



30 exploitable

146 crashes
82 unique

157 crashes
56 unique

2 Other

PowerPoint thoughts

- Didn’t see nearly as many crashes in Windows PowerPoint as in PowerPoint for Mac
  - Significantly different code base?
    - Reliance on different OS libraries, memory management?
- Almost every Windows crash was unique (24/28)
- Seem to be a high percentage of “exploitable” crashes

Hand checking

Test case	!exploitable	crashwrangler	Hand check on Mac
-541-	exploitable	no	Probably not
-235-	exploitable	yes	Looks exploitable
-1173-	exploitable	no	Probably not
-1035-p	exploitable	no	Probably not
-840-	exploitable	no	Probably not
-1071	exploitable	no	Probably not
-269	Probably exploitable	no	Probably not
-600	Probably exploitable	no	Probably not
-115	Probably exploitable	no	Probably not
-1035-f	Probably exploitable	yes	Looks exploitable
-407	Probably exploitable	no	Probably not
-215	Probably exploitable	no	Probably not
-830	Unknown	no	Dunno
-1186	Unknown	no	Probably not
-1007-	Unknown	no	Probably not
-801	Unknown	yes	Probably not
-27-	Unknown	no	Probably not
-1195-	Unknown	no	Probably not
-246	Unknown	no	Probably not
-625	Unknown	no	Dunno
-500-	Unknown	yes	Looks exploitable
-1126-	Probably not exploitable	no	Probably not
-274-M	Probably not exploitable	yes	Looks exploitable
-1069	Probably not exploitable	no	Probably not

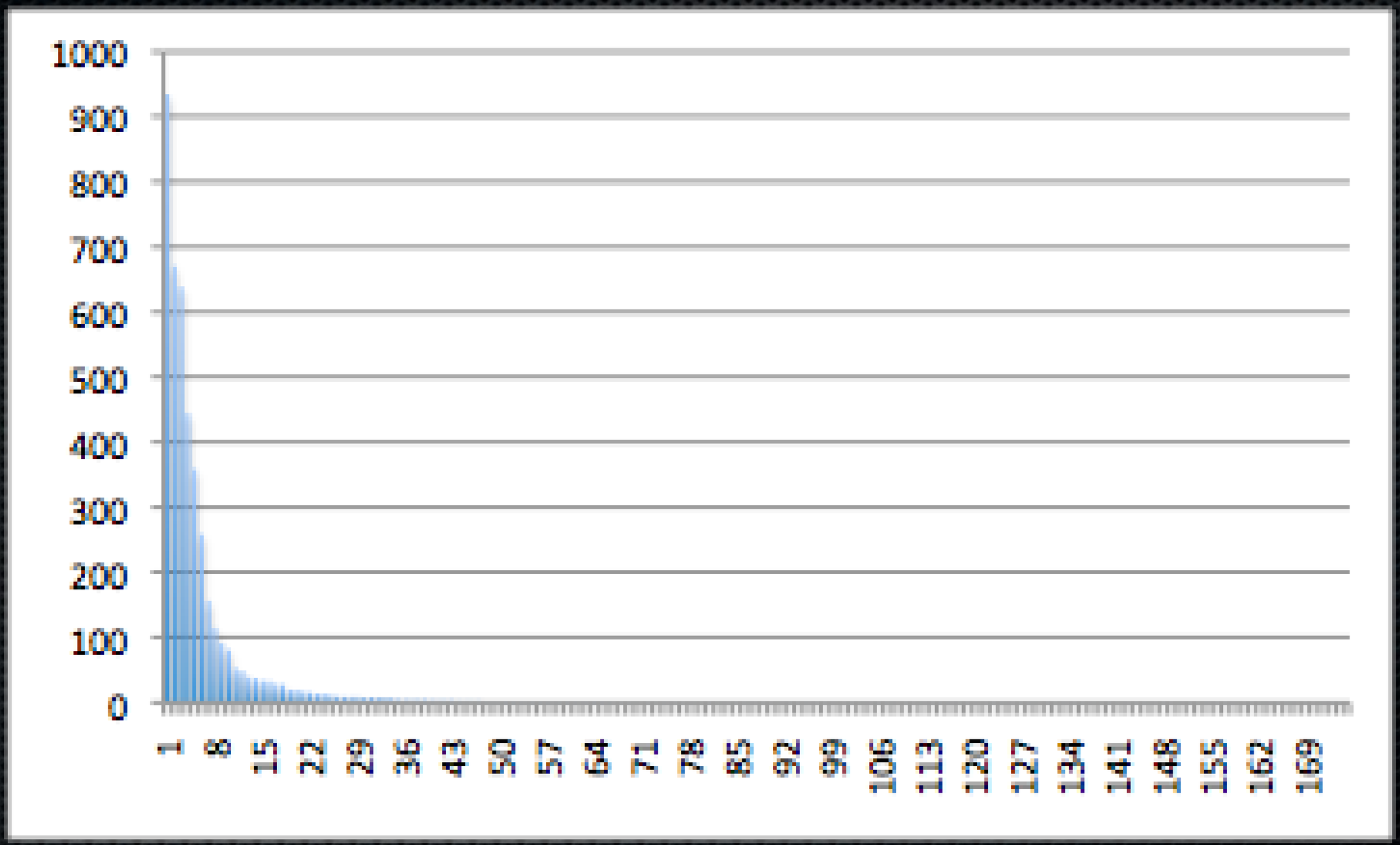
More hand checking

- If you disregard the “dunnos”
  - Crashwranger agrees with me over 95% of the time!
    - One single false positive, one false negative
  - !exploitable agrees 26% of the time
    - Hand checking was on Mac not Windows
    - !exploitable had both Type 1 and Type 2 errors

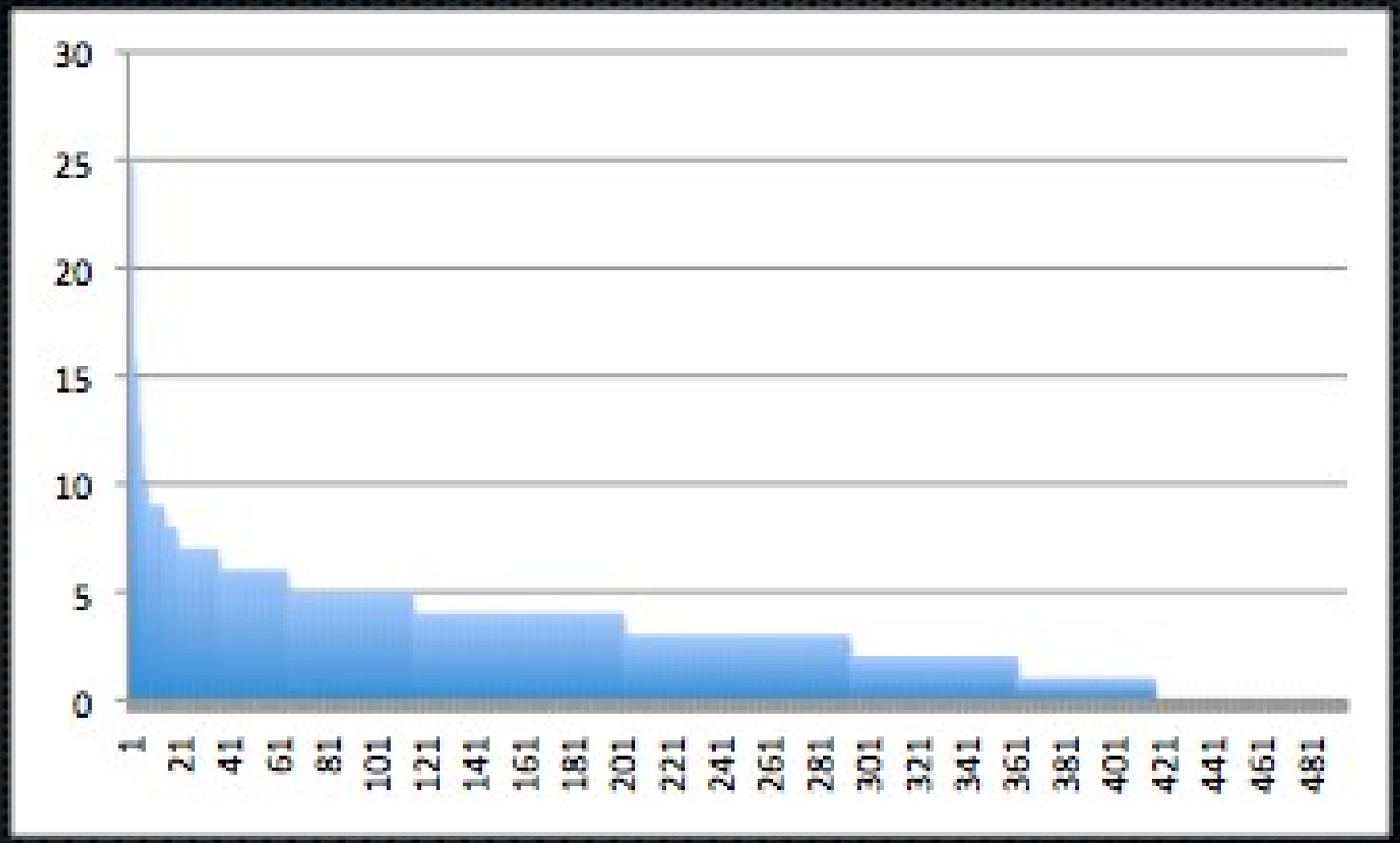
Crash rarity

- 174 crashes (by EIP)
- 108 found only once (62%)
- 149 found less than 10 times (86%)
- 8 crashes found more than 100 times
- 1 crash found 935 times

Crash rarity



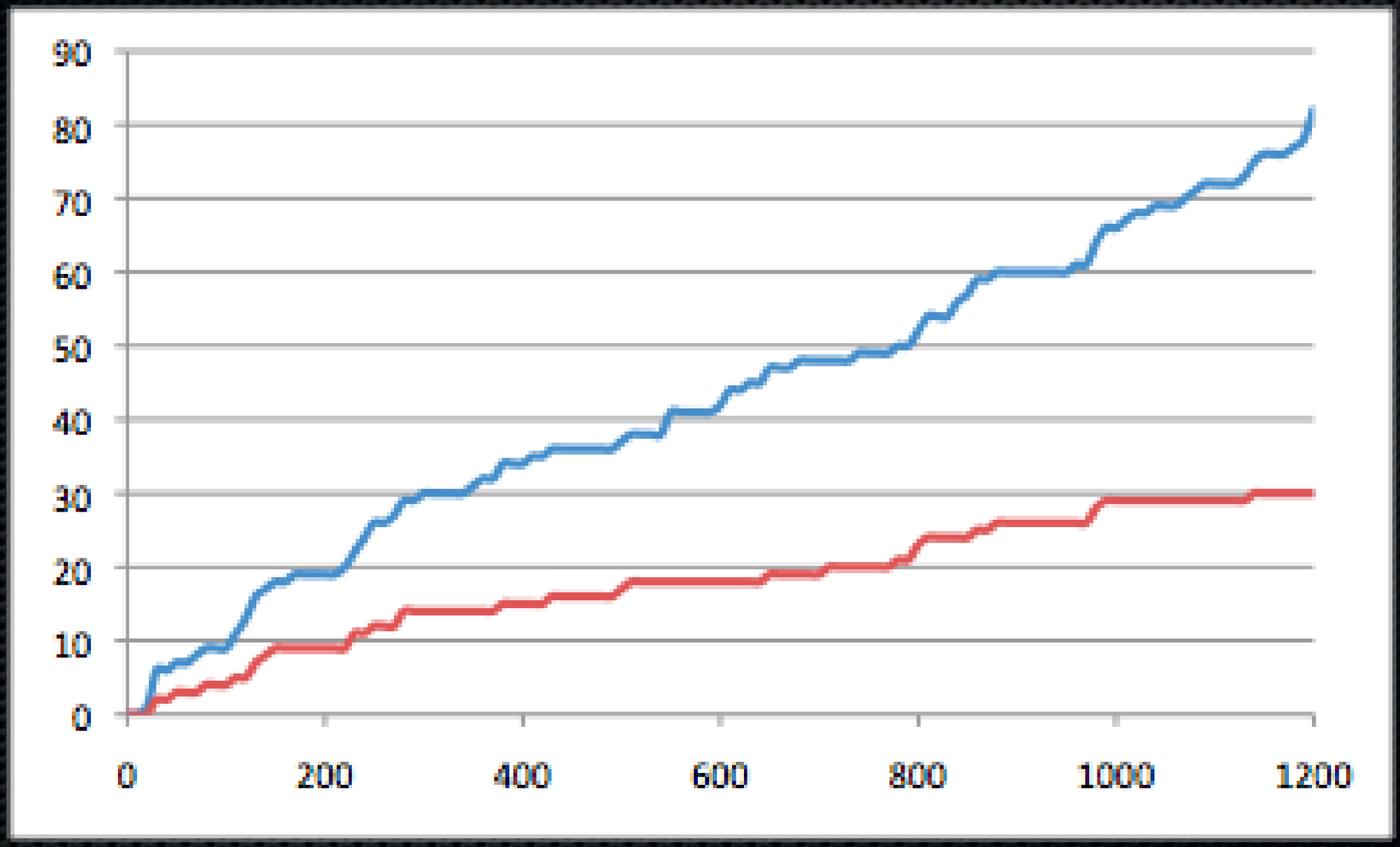
Unique crashes by file



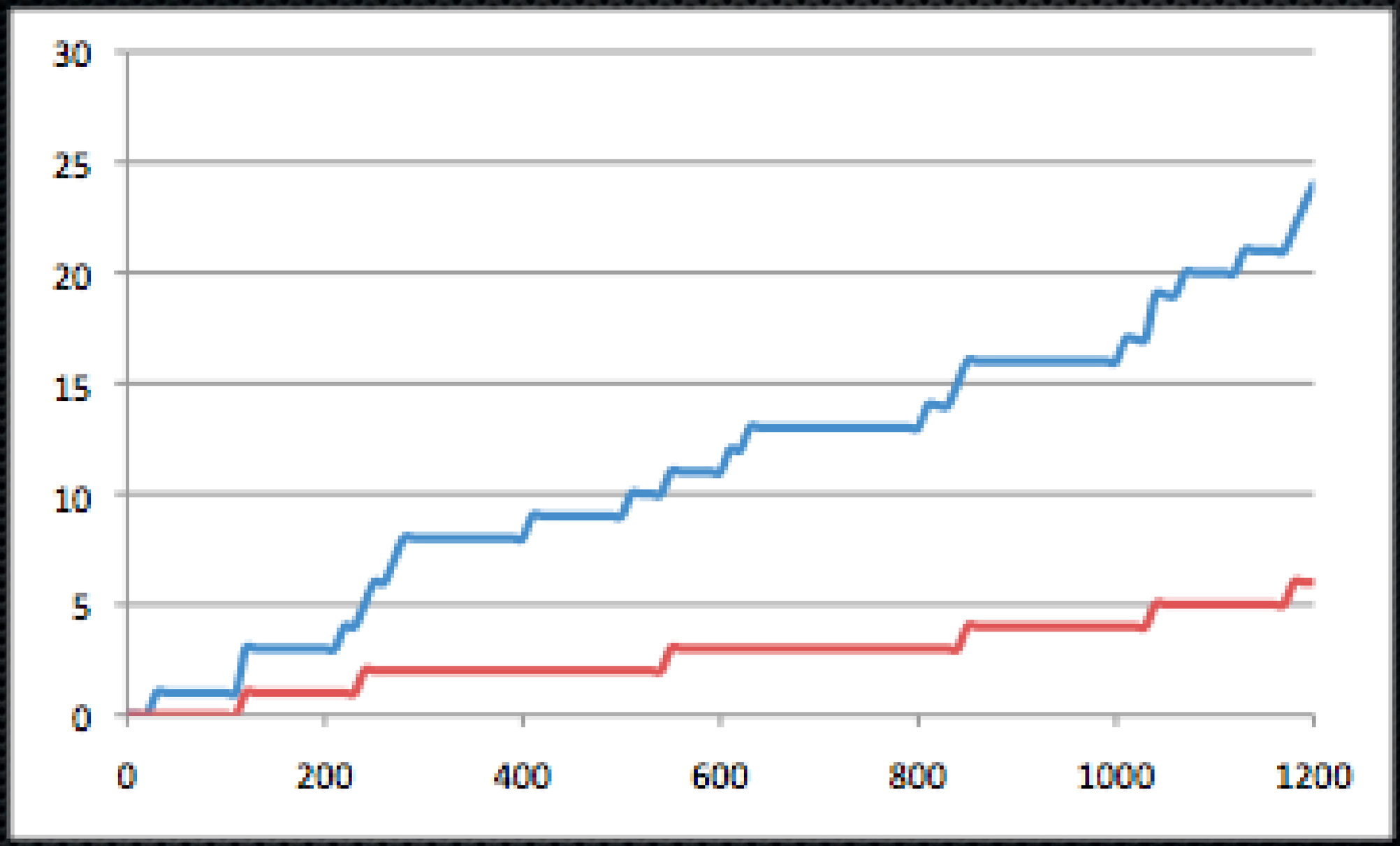
More crashes by file

- 79 files found nothing
- 203 found 2 or fewer crashes
- 7 files found 10 or more crashes
- 1 file found 25 crashes

Crashes by iteration number (OS X)



Crashes by iteration # (Win)



PPT showdown

205 crashes
30-70 unique
10-12 exploitable

174 crashes
30-80 unique
6-30 exploitable

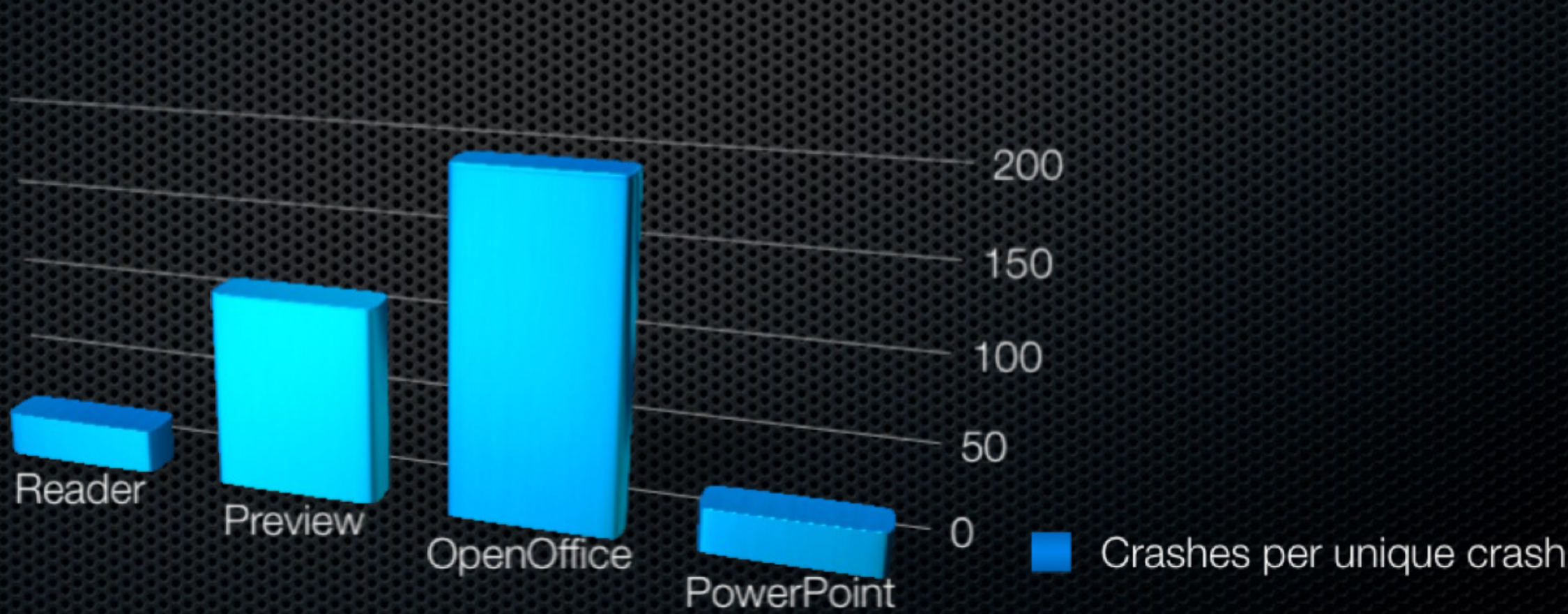
Fuzzing “truths” revealed

Caveats

- Only 4 data points
- I present the data, you draw your own conclusions

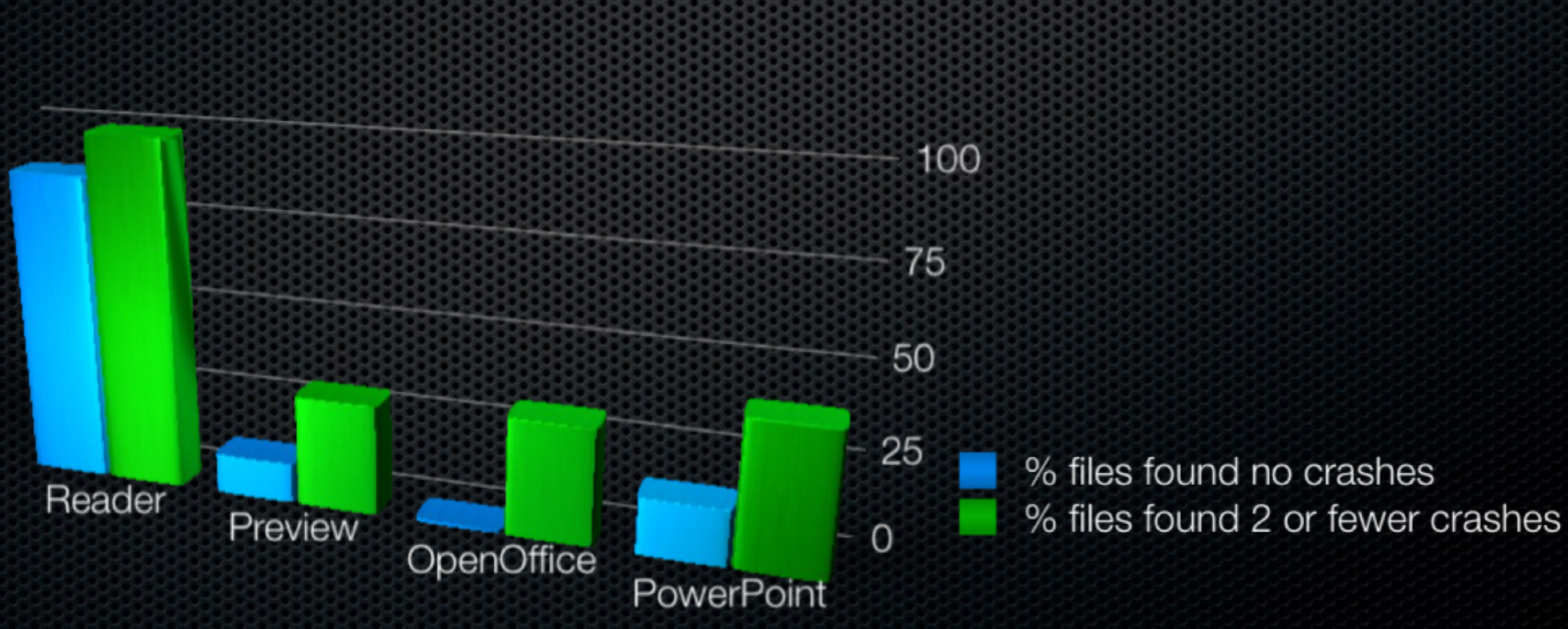
Crashes per unique crash (by EIP)

- Expect lots of crashes between unique crashes
  - Anywhere between 25 and 200, depending on the program



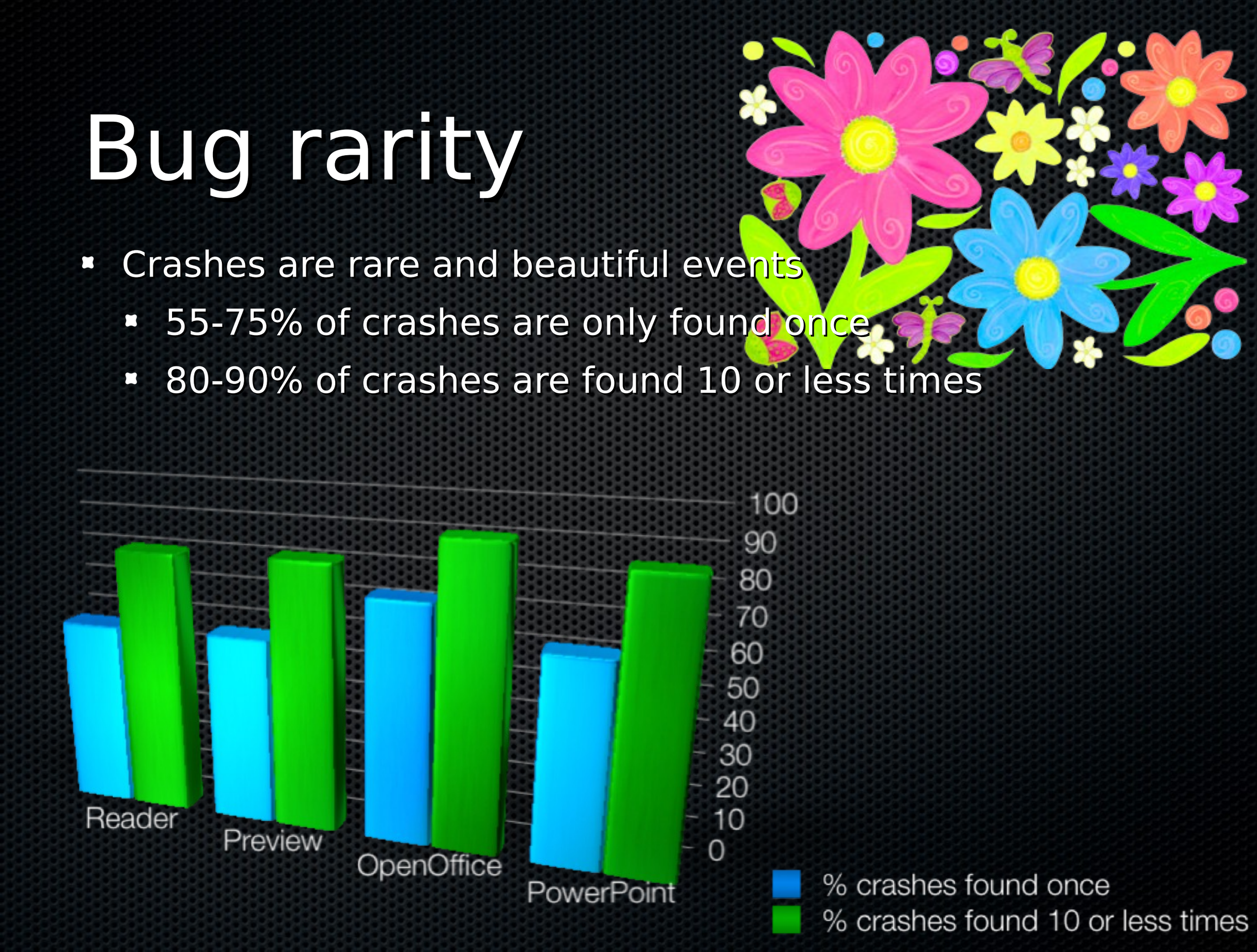
Choice of initial files

- Over 25% of files found 2 or fewer different crashes
- Except OpenOffice, >10% of files found no crashes
- These files represent less than 2% of Internet files



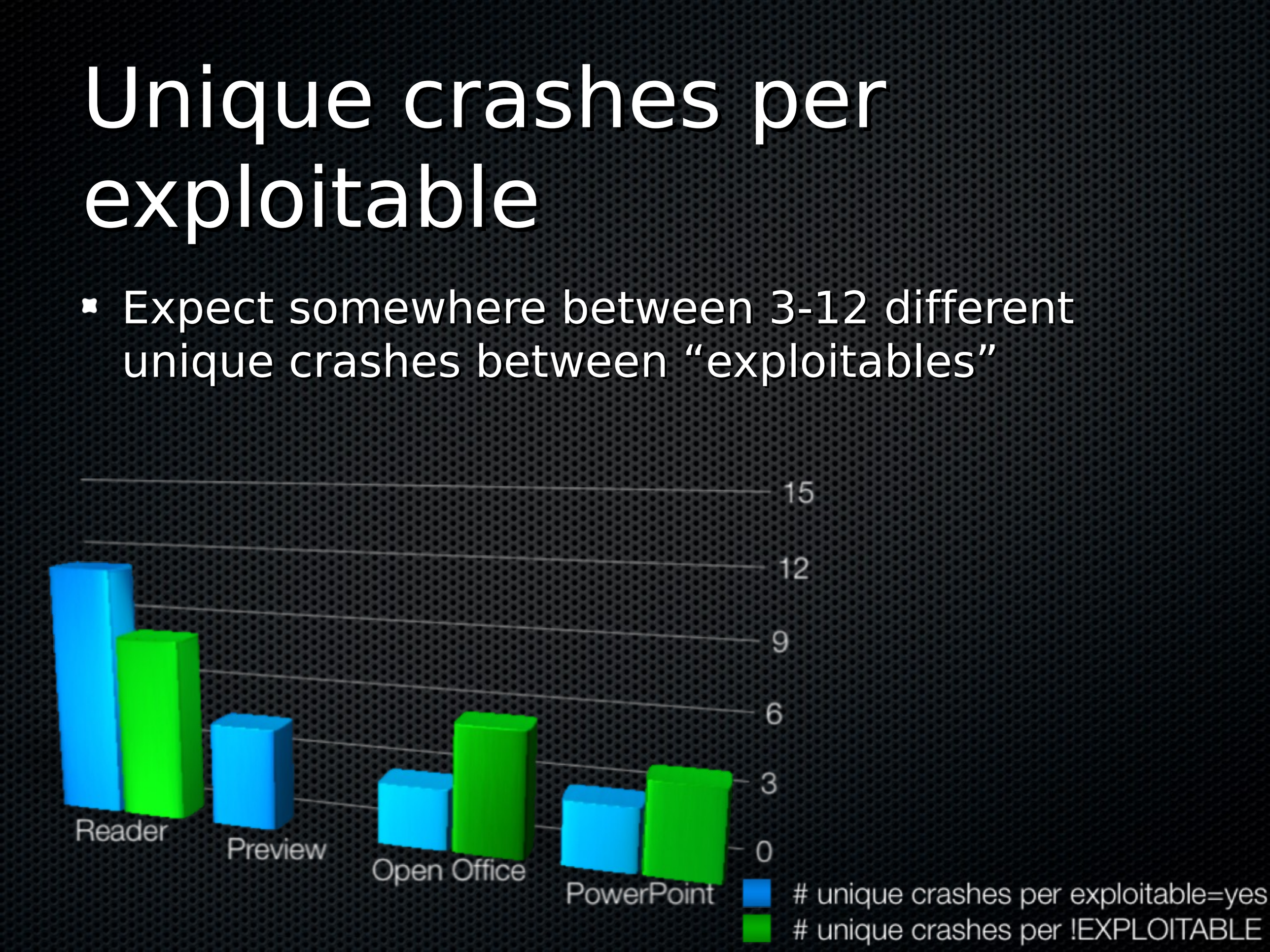
Bug rarity

- Crashes are rare and beautiful events
  - 55-75% of crashes are only found once
  - 80-90% of crashes are found 10 or less times



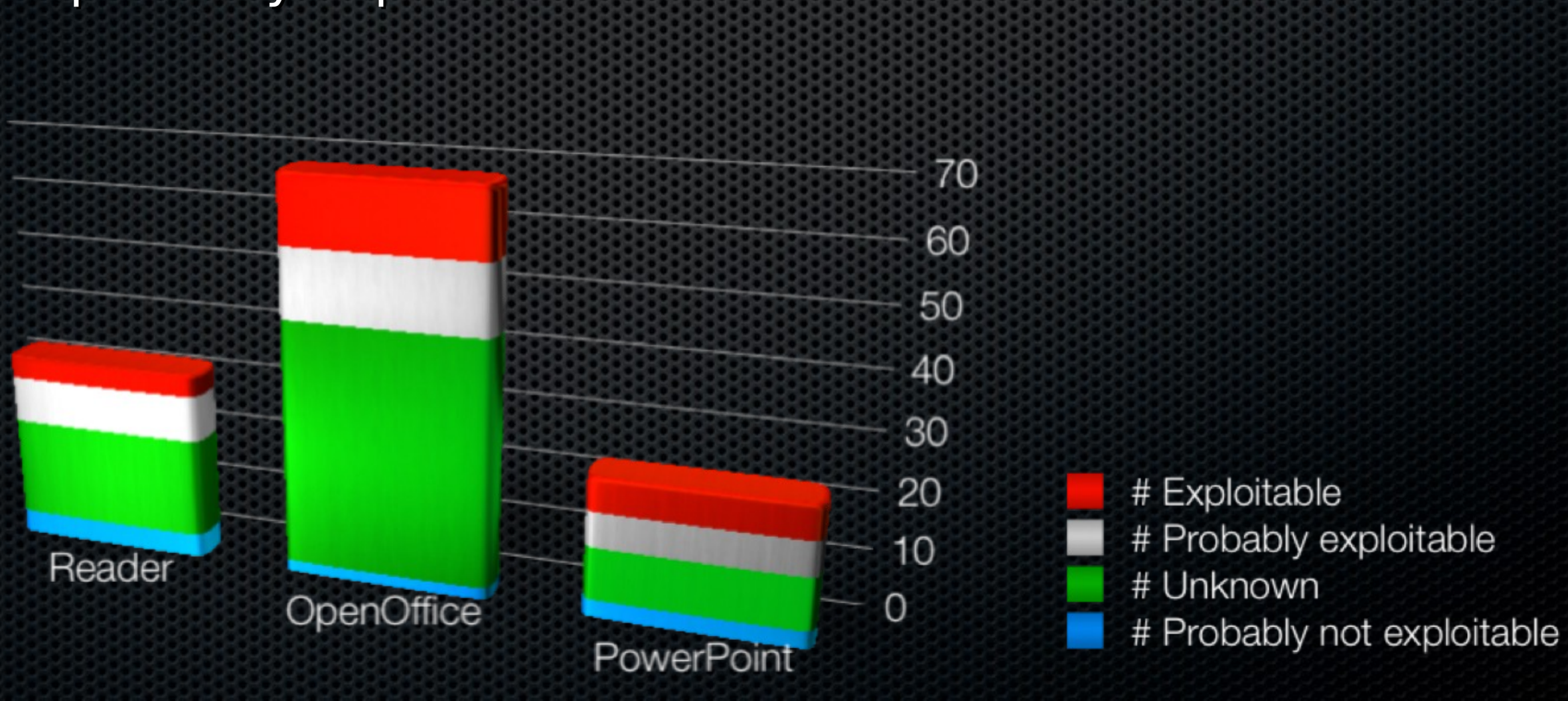
Unique crashes per exploitable

- Expect somewhere between 3-12 different unique crashes between “exploitables”



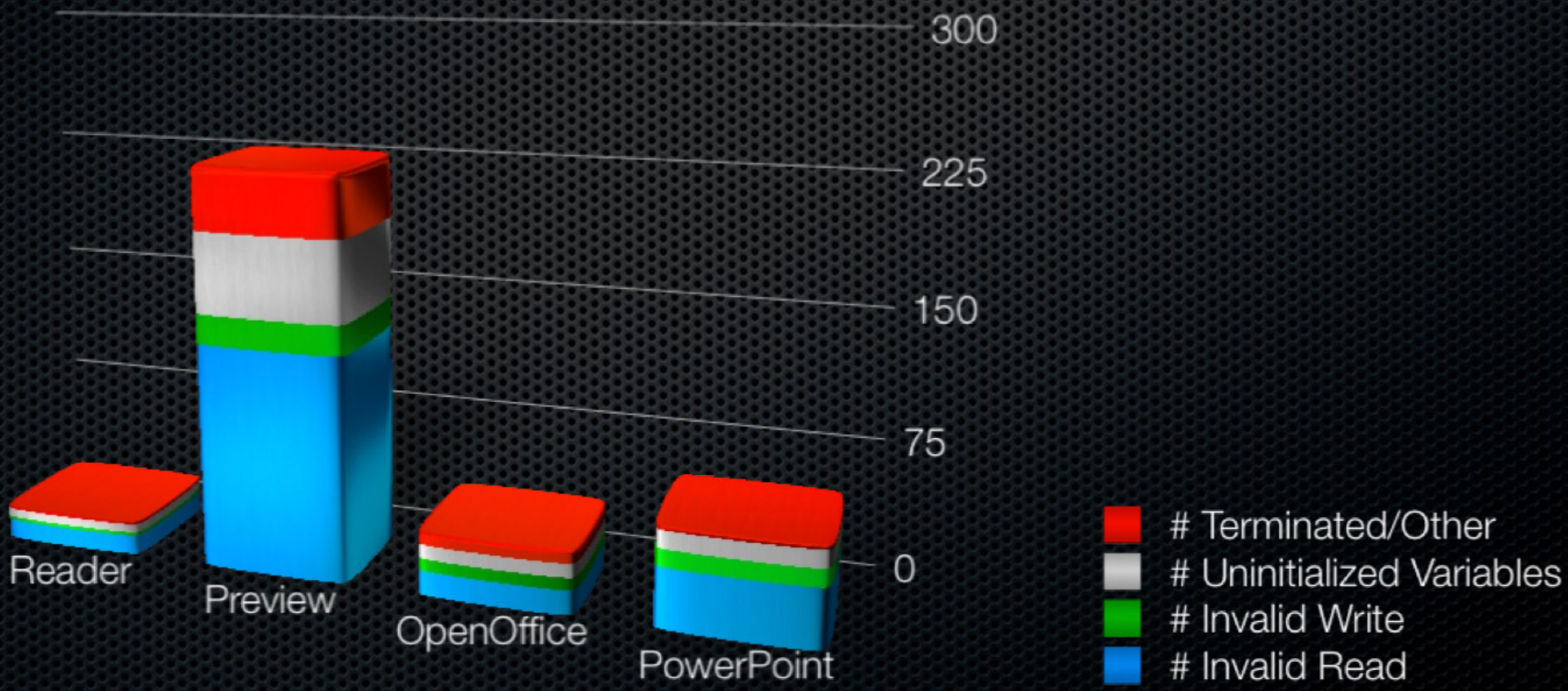
!exploitable bug classifications

- Expect roughly 12-25% of crashes to be exploitable
- Expect roughly 35-50% of crashes to be at least probably exploitable



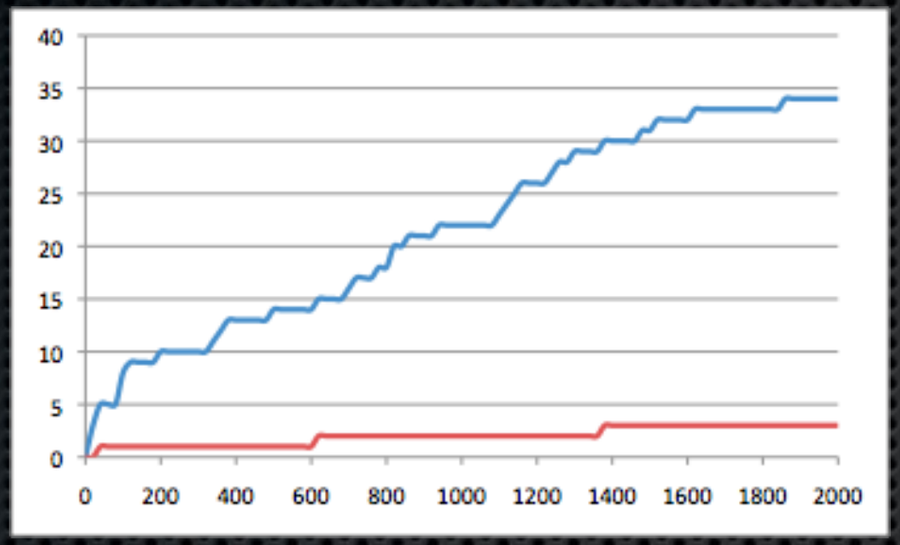
Valgrind bug types

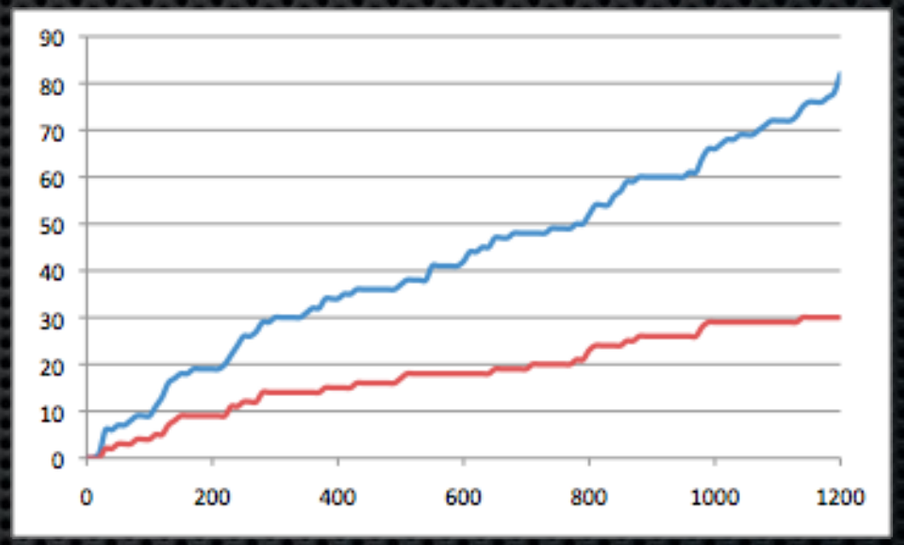
- Expect a rough split of 40/20/20/20 for Read/Write/Uninitialized/Terminated

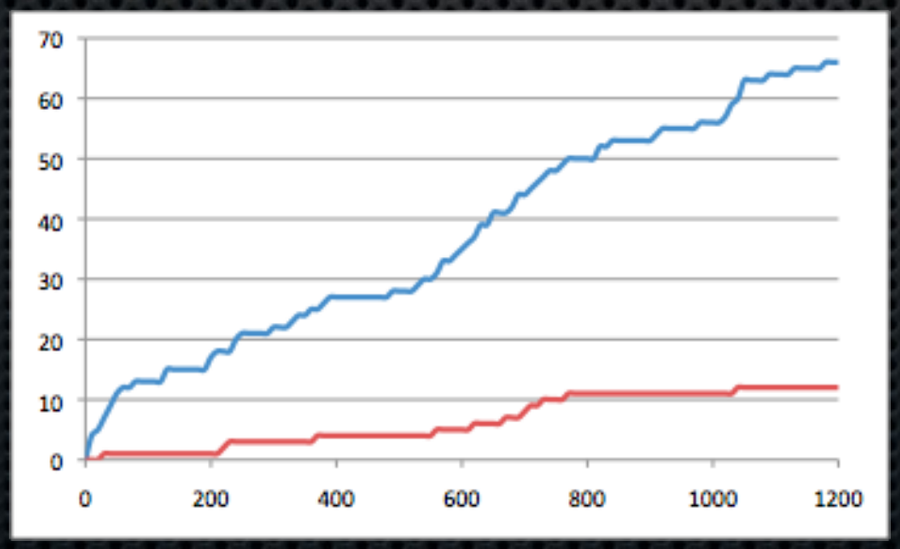


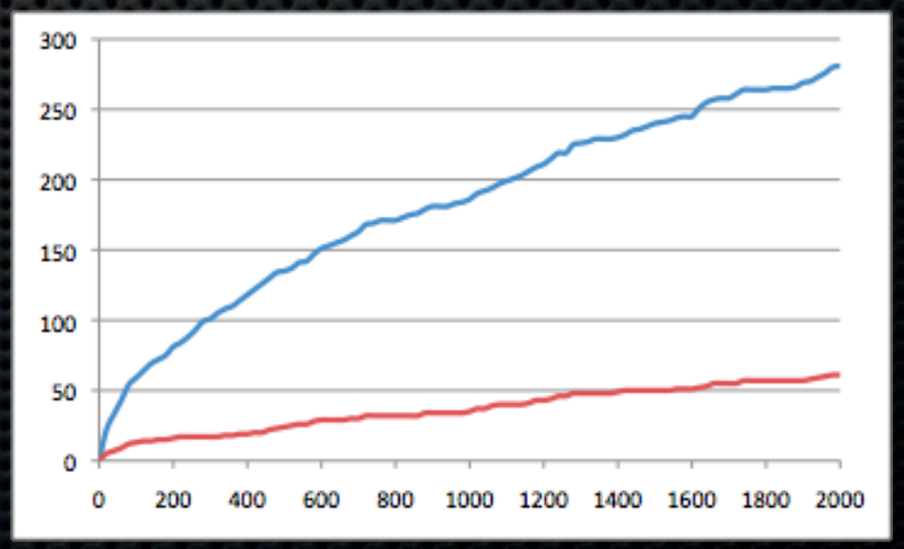
iterations

- Expect to fuzz more than 2000 iterations per file









Vendors

- Despite the fun I had, please fuzz your products
  - You’re not doing a good enough job at this
  - Especially some of you!
- Fix the bugs you find, eventually someone else will find them
- This talk isn’t designed to embarrass you, just to present my findings
  - If you’re embarrassed, good, do something about it

Questions?

- E-mail me: cmiller@securityevaluators.com
- Follow me: @0xcharlie