Corpus distillation & fuzzing

Jaanus Kääp
Clarified Security

Who is this guy

- Jaanus Kääp
- Working at Clarified Security
  - Vulnerability testing, research, trainings, cyber excercises
- Developer background (web and native)
- Multiple bug bounties:
  - Facebook, Adobe, Google, MS etc

Why this topic

- CVE-2015-6696 Adobe Reader
- CVE-2015-6698 Adobe Reader
- CVE-2015-6978 Apple Safari
- CVE-2016-0936 Adobe Reader
- CVE-2016-0938 Adobe Reader
- CVE-2016-0939 Adobe Reader
- CVE-2016-0046 Microsoft Reader
- CVE-2016-0118 Microsoft Edge
- CVE-2016-1009 Adobe Reader
- CVE-2016-1088 Adobe Reader
- CVE-2016-1093 Adobe Reader
- CVE-2016-1094 Adobe Reader
- ZDI-15-525 Foxit Reader
- ZDI-15-524 Foxit Reader
- ZDI-15-641 Foxit Reader
- ZDI-16-029 Foxit Reader
- ZDI-16-221 Foxit Reader

Topic itself

- Fuzzing basics
- Corpus distillation
- How I fuzz
- Tools I developed

Fuzzing

- Automated bug/vuln finding
- Functionality
  - Generates input
  - Runs target with input
  - Detects crash/special condition
  - Rinse and repeat

Detection

- Detects crashes
- Detects special exceptions (access violation)
- Detects special behaviour (DOS, error message, output)
- Target dependent

Input generation

- Dumb fuzzing
  - Total/Stupid random
  - Mutations based
- Smart fuzzing
  - Generation based

Total/Stupid random

- Examples
  - **dd if=/dev/urandom bs=100000 count=1 > file.doc**
  - **dd if=/dev/urandom bs=1024 count=1 | telnet target 443**
- Pros
  - Stupid easy to create
- Cons
  - Relatively useless in most case

Mutations based

- Mutates/changes initial set
  - Bit flipping
  - Adding long strings
- Pros
  - Easy to create
  - Better code coverage
- Cons
  - Code coverage depends on initial set

Generation based/smart fuzzing

- Fuzzer creates new valid input from scratch
- Pros
  - Best code coverage if well implemented
  - Better control
- Cons
  - Time consuming to implement
  - Closed protocol == reverse engineering

What I did (pdf)

- I'm lazy
- Wanted to find vulnerabilities
- Have couple of computers laying around

  ==

- Dumb fuzzing

How to improve initial set

- As much functionality as possible
- Unknown protocol (for me)
- Very common filetype
- Lot of readers

Corpus distillation

- What you need
  - Huge number of initial files
  - Application that can read them
  - Time and computing power
- What you do
  - Code coverage with every input
  - Analyse the coverage of all the files
  - Minimize the set

Base logic

- Initial files (functionality):
  - Input1 (A, B, F)
  - Input2 (A, C, D, E)
  - Input3 (A, B, D)
  - Input4 (A, E)
  - Input5 (A, F)
- Final set (covers same functionality):
  - Input1 (A, B, F)
  - Input2 (A, C, D, E)

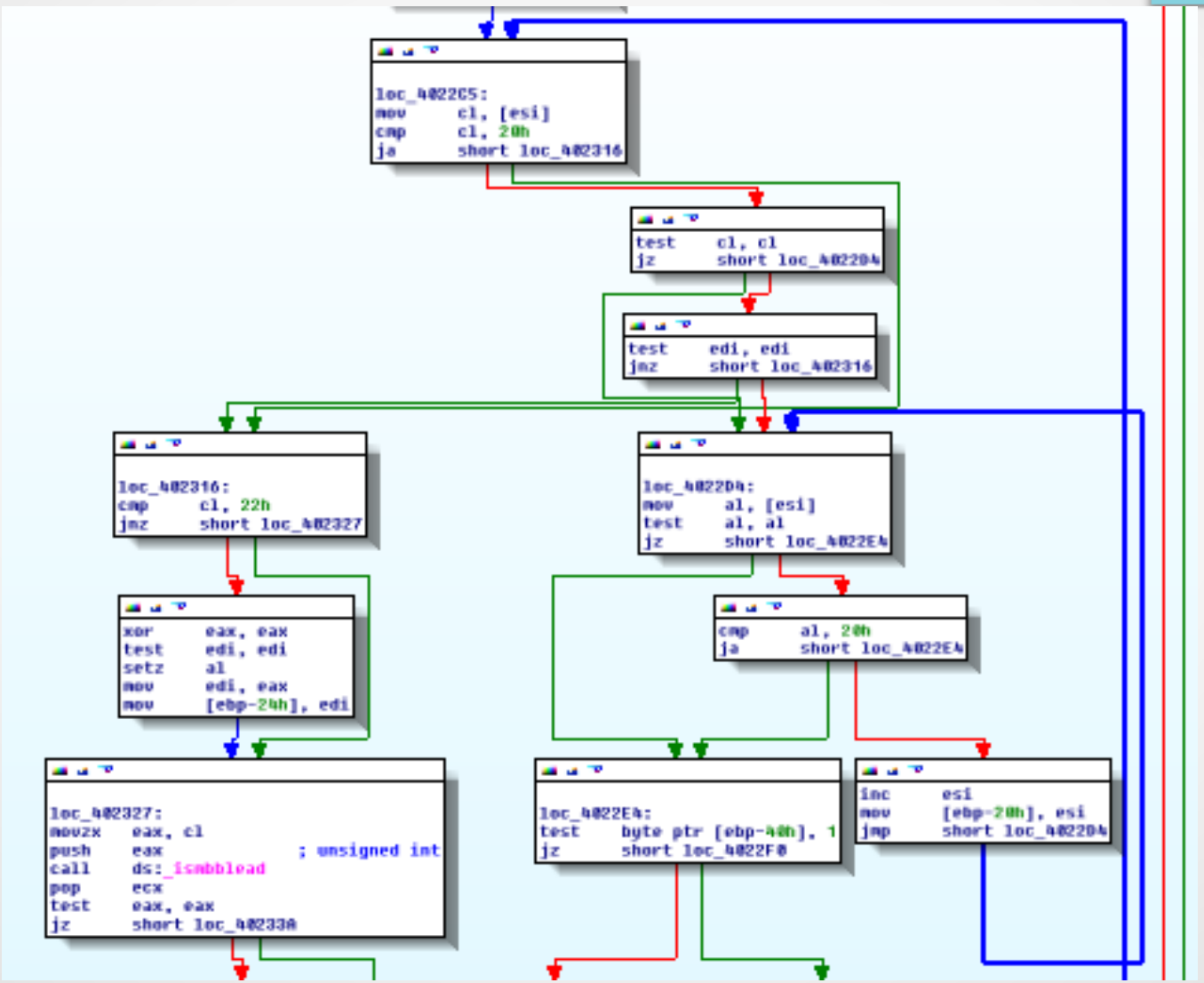
Code coverage

- Open source – simple (special flags)
- Closed source
  - Trace the code (dead slow)
  - Some tools/libs: Pin, DynamoRIO
  - Write coverage tool yourself

Code coverage

- Not every asm instruction
- Basic blocks are enough
- First idea:
  - Breakpoint to every basic block
- First implementation
  - Set breakpoints
  - Write down each bp-event
  - Continue execution

How to get basic blocks



How to get basic blocks

- IDA pro + IDApython
- Each basic block
  - RVA from base address

First run

- Foxit software
  - 611 927 breakpoint
  - 8 sec wait
  - 180 seconds on VM for setup
  - 30 seconds for execution
  - TOTAL: ~210s/execution == 411 runs per day

- **TOO SLOW**

How to speed up?

- Most time was spent on setting breakpoints
- What is breakpoint
  - 0xCC
- Why not set them in executable?

How to get basicblocks

- IDA pro + IDApython
- Each basic block
  - RVA from base address
  - RVA/Offset in the file
  - Original value

New process

- Prep
  - IDA analysis
  - Basic blocks file generation
  - Modification of the exe/dll files
- Execution
  - Catch 0xCC exceptions
  - If in the basic block list
    - Record location
    - Replace 0xCC with original value
    - EIP = EIP – 1

Second run

- Foxit software
  - 611 927 breakpoint
  - 8 sec wait
  - 30 seconds for execution
  - TOTAL: ~30s/execution

- **MUCH (~7x) BETTER**

Additional optimization

- Reducing basic blocks count
  - Analyse 100 files
  - Take file/files with most coverage
  - Add them to final set
  - Remove basicblocks covered by them

Third run

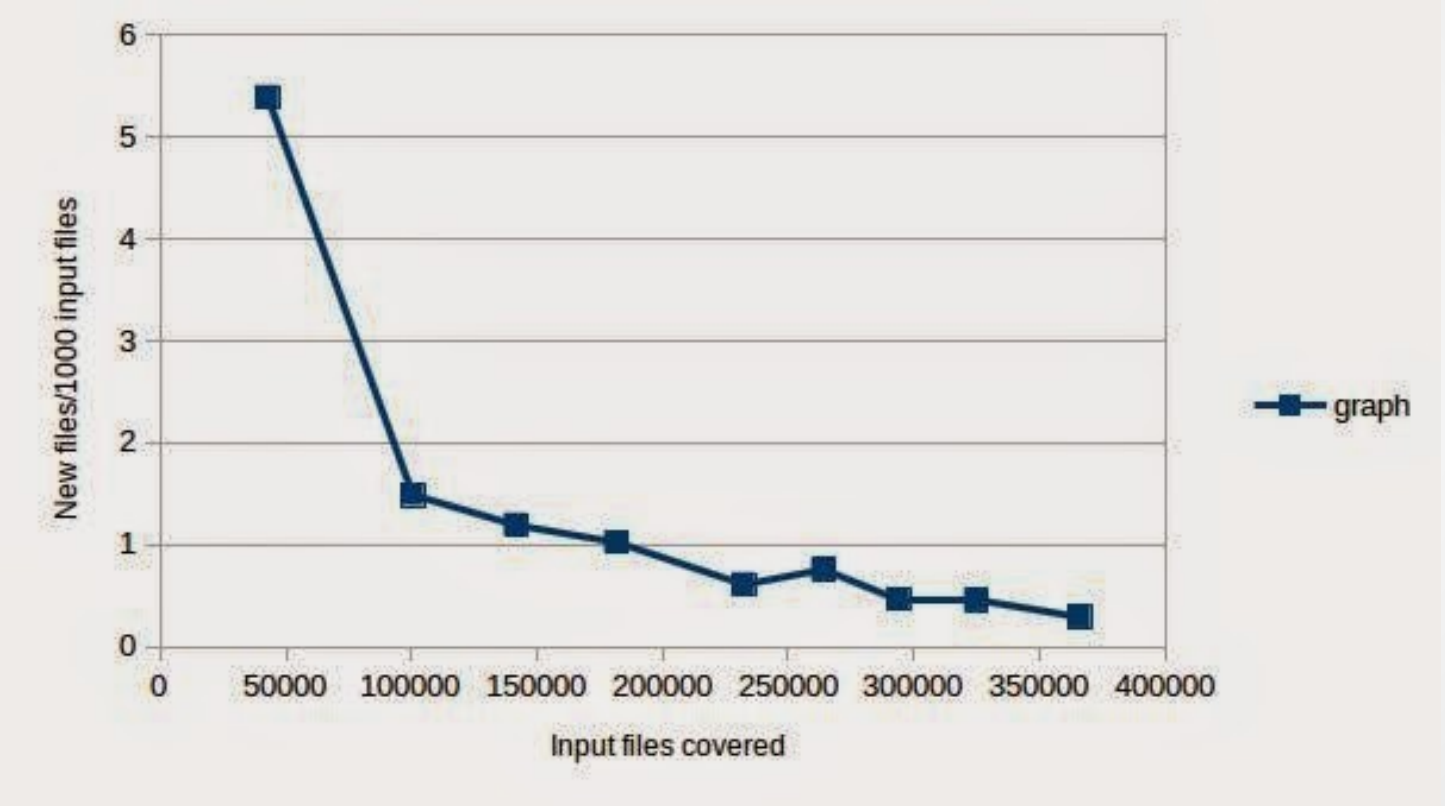
- Foxit software
  - <600 000 breakpoint
  - 8 sec wait
  - 13 seconds for execution
  - TOTAL: ~13s/execution

- **EVEN MORE (~16x) BETTER THEN FIRST**

Benefit

- I had 366027 pdf files
- Final set was 726 files (removed 1)
- Code coverage was 21.8%
- It took ~2 weeks

How large initial set you need?



How large initial set you need?

What does corpus distillation look like at Google scale? Turns out we have a large index of the web, so we cranked through 20 terabytes of SWF file downloads followed by 1 week of run time on 2,000 CPU cores to calculate the minimal set of about 20,000 files. Finally, those same 2,000 cores plus 3 more weeks of runtime

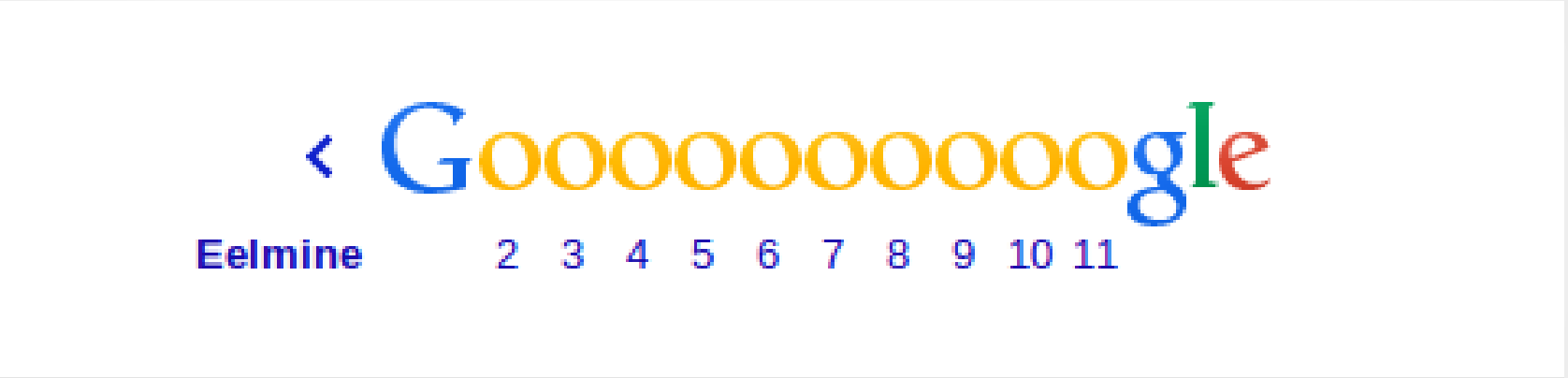
Benefit (latest results)

Filetype	Software	Initial set	Final set
pdf	Adobe Reader	400 000	1217 (0.30%)
doc	MS Word	400 000	1319 (0.33%)
docx	MS Word	400 000	2222 (0.56%)

How to get these files?

- Google „filetype: pdf“ ()

How to get these files?



Additional problems

- Not real pdf files
- DDOS protection

Solution

- Searches
  - filetype:pdf aaa
  - filetype:pdf aab
  - filetype:pdf aac
- Not real pdf files
  - Magic value - %PDF
- DDOS protection
  - It's all about timing

Fuzzing itself

- Mutate input file
- Run executable with debuging
- Wait for crash or exception
  - Report and primary filtering
- Repeat

What mutations

- Random bit flipping
- Adding stuff (long strings for example)
- Special values like x00, xFF, xFFFF, xFFFFFFFF etc

Detection

- Debuger
  - Exceptions (access violations, DEP, etc)
- **Full page heap is your friend**

Keep in mind

- ASLR calculations
- Random crashes
- More instances the better
- Filter the issues automatically!!!

About filtering

- You need it unless you are **VERY** bored!
- My simple filtering:
  - Near NULL/Not near NULL/Both
    - Type of exception
      - Location of exception
        - [STACK trace]

Analysis of results

- First info
  - Code in crash location
  - Stack trace
  - Registry values
- If promising then additional analysis

What can you do with results

- Inform vendor
- ZDI for money
- Full disclosure for ~fame and hate
- Writing exploits for more money (ZERODIUM)
- Writing exploits for own usage

About my tools

- Fuzzer „Vanapagan” (Windows, Linux, Android over ADB)
  - https://github.com/JaanusFuzzing/Vanapagan
- Code coverage tool „KavalAnts” (Windows only)
  - https://github.com/JaanusFuzzing/KavalAnts

- NB1: Both have no real documentation yet and both are developed as my own needs dictate :(
- NB2: KavalAnts needs IDA Pro for initial analysis – not for coverage execution.

About my tools

- If you use them and find something
  - Let me know (it's cool to know)
  - If you sell some, then donate 10%
    - Fuzzing needs luck
    - Donation increases karma
    - Karma increases luck (at least it should)

Q&A

Thank you

Jaanus.kaap@gmail.com

Jaanus@clarifiedsecurity.com