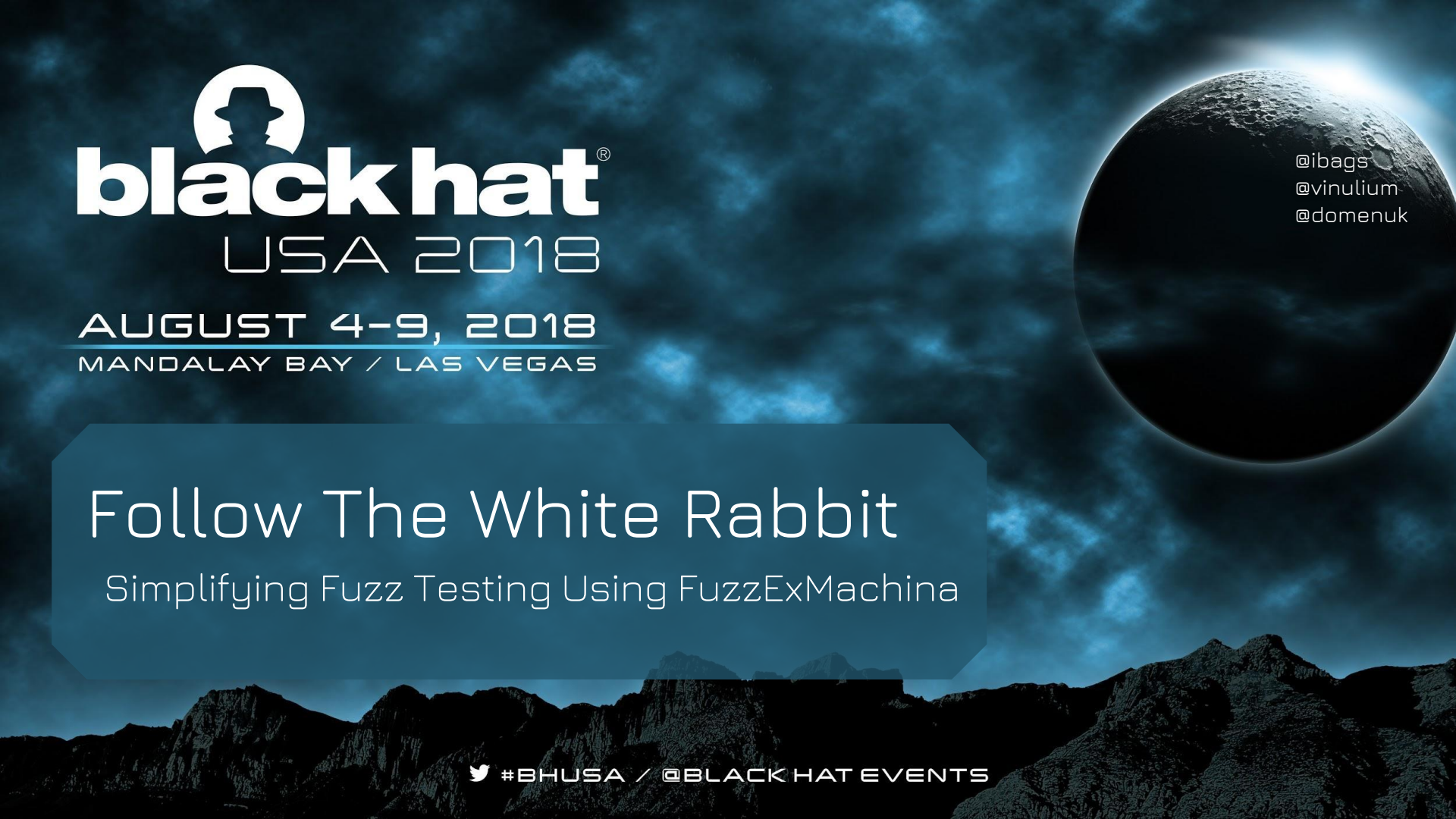

black hat®
USA 2018
AUGUST 4-9, 2018
MANDALAY BAY / LAS VEGAS
@ibags
@vinulium
@domenuk
Follow The White Rabbit
Simplifying Fuzz Testing Using FuzzExMachina
#BHUSA / @BLACK HAT EVENTS

Who Are We?

Vincent Ulitzsch

Grad Student @ TU Berlin

Intern @ Security Research Labs

Bhargava Shastry

PhD Student @ TU Berlin

Security Researcher

Dominik Maier

PhD Student @ TU Berlin

Program Manager Security @ AVM GmbH

TL;DR

FuzzExMachina (FExM)

- Automated fuzzing framework
- Clever tricks up its sleeve
- BYOB or fuzz distribution scale
- Found numerous bugs and crashes
- Free and open source

→ https://github.com/fgsect/fexm

Outline

- Introduction
  - Demo 1: Bring Your Own Binary
- FuzzExMachina
  - Demo 2: Bug Dashboard
- Findings
- TimeWarp
  - Demo 3: TimeWarp@BYOB
- Conclusion

Introduction
FuzzExMachina
Findings
TimeWarp
Conclusion

Introduction

TL;DR: Throw corner-case input at a program until it breaks

COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN-MADISON

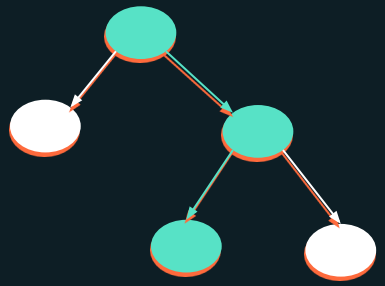
CS 736
Fall 1988

Bart Miller

Operating System Utility Program Reliability – *The Fuzz Generator:* The goal of this project is to evaluate the robustness of various UNIX utility programs, given an unpredictable input stream. This

A Brief History of Fuzzing

Feedback Guided



Input Spec. Guided

```
<DataModel name="Proto">
...
</DataModel>
```

Random

cat /dev/urandom | program

Early 2000's

Late 2000's

State of the Art

OSS-Fuzz - Continuous Fuzzing for Open Source Software

google / oss-fuzz

Watch 196 | Star 2,905 | Fork 492

Code | Issues 77 | Pull requests 12 | Projects 0 | Insights

OSS-Fuzz - continuous fuzzing of open source software

fuzzing | security | stability

4 trillion test cases per week!

What is Microsoft Security Risk Detection?

Security Risk Detection is Microsoft's unique fuzz testing service for finding security critical bugs in software. Security Risk Detection helps customers quickly adopt practices and technology battle-tested over the last 15 years at Microsoft.

READ SUCCESS STORIES >

yFuzz

yahoo / yfuzz

Watch 10 | Star 87 | Fork 5

Code | Issues 8 | Pull requests 0 | Projects 1 | Wiki | Insights

A project to run fuzzing jobs at scale with Kubernetes.

Trophies Abound

Google's Fuzz bot exposes over 1,000 open-source bugs

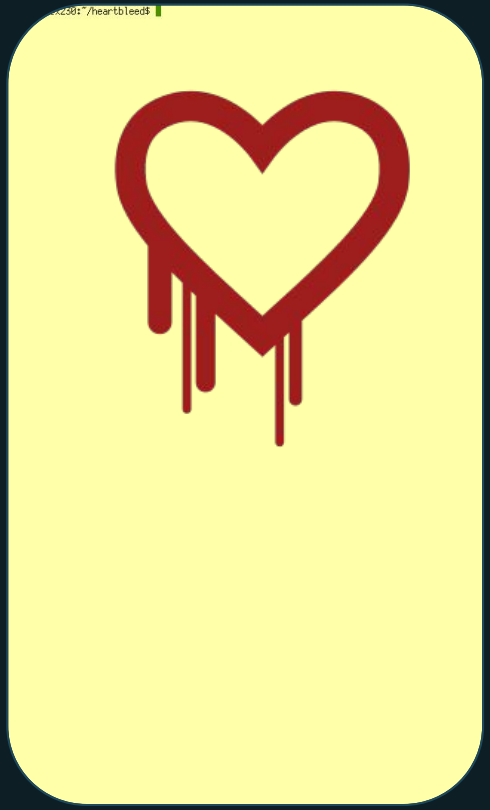
The OSS-Fuzz robot has uncovered vulnerabilities in a number of key open-source projects.

By Charlie Osborne for Zero Day | May 9, 2017 -- 07:50 GMT (08:50 BST) | Topic: Security

Linus Torvalds says targeted fuzzing is improving Linux security

Linux 4.14 release candidate five is out. "Go out and test," says Linus Torvalds.

By Liam Tung | October 17, 2017 -- 12:34 GMT (13:34 BST) | Topic: Security



*Aleph One. Smashing the Stack for Fun and Profit. Phrack 7, 1996

We still find buffer overflows like it's 1996*

So...

Why are dev/QA teams not fuzzing yet?

We Are Fuzzing the Sub 1%

- Google OSS-Fuzz
  - < 150 projects
  - < 2 years old
- Modern OS distribution
  - > 50K projects

So...

What about the 99%?

Case Study: Open vSwitch

- Here's what I did
  - Write a test program
  - Provide seed corpus
  - Write a build script and a Dockerfile

Googler Perspective

- Huge Inertia
- Lack of awareness
- Religious reasons
- No time

Expectation

Dev: Fuzz my software repo for me

Bot: Here you go, these are the bugs I found!

Reality

Dev: Fuzz my software repo for me

Bot: Give me test case, seeds, config and build script

Dev: kthxbye

Design Philosophy

Start automatic fall back to manual

It’s Demo time folks!

DEMO[FExMBYOB]

Introduction
FuzzExMachina
Findings
TimeWarp
Conclusion

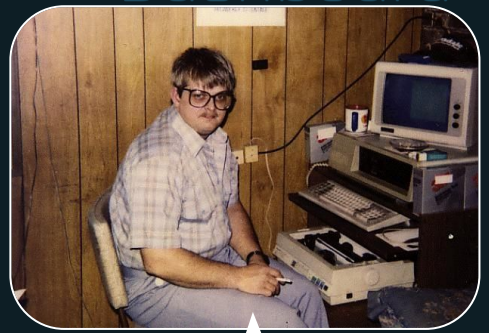
FuzzExMachina (FExM)

Distributed, large scale fuzz testing framework

- Simplifies the fuzzing process
- Almost entirely automated
- Built around battle-proof software
- Start fuzzing hundreds of packages immediately

The FExM Pipeline

Dashboard

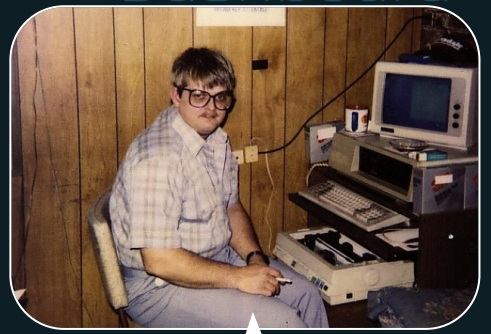


Crawl Binaries → Infer Inputs → Select Seeds → Fuzz! → Triage Crashes

The FExM Pipeline (1/6)

Crawl Binaries → Infer Inputs → Select Seeds → Fuzz! → Triage Crashes

Dashboard



Crawling for Binaries

Choose
Repository

Compile
Package

Compile
& instrument
(if possible)

Extract
Binaries

Instrumentation for Feedback

To receive feedback, we can use instrumentation at

Compile Time

- fast
- requires source
- hard to automate

Run Time

- blackbox
- slow (2-5x)

The FExM Pipeline (2/6)

Crawl Binaries → **Infer Inputs** → Select Seeds → Fuzz! → Triage Crashes

Dashboard

Input Vector Inference

tcpdump -nvr file.pcap

Identify this

Input Vector Inference

For each binary in the repo

Identify:

1. Parameters (when is input processed?)
2. Input channel (stdin/socket, file by filename?)

Examples:

- tcpdump -nvr <file>
- wget localhost 80 → via preeny desock

Parameter Inference

tcpdump
[-AbdDefhHJKLlnNOpqStuUvxX#]
[-B *buffer_size*]

Collect
Parameter
Candidates

parse help-flag
other heuristic

For each

Validate
Parameter
candidates

does param
lead to
input processing?

tcpdump -nvr <file>

Parameter Validation

Invoke with dummyfile

strace -yy -f .. -- binary <invocation>

STDIN || NETWORK:
desock & <dummy

FILE:
-f dummyfile

Processed?

grep for:
open(dummyfle)
read(dummyfile)

The FExM Pipeline (3/6)

Dashboard

Crawl Binaries → Infer Inputs → Select Seeds → Fuzz! → Triage Crashes

About Seeds

A good seed is a valid program input

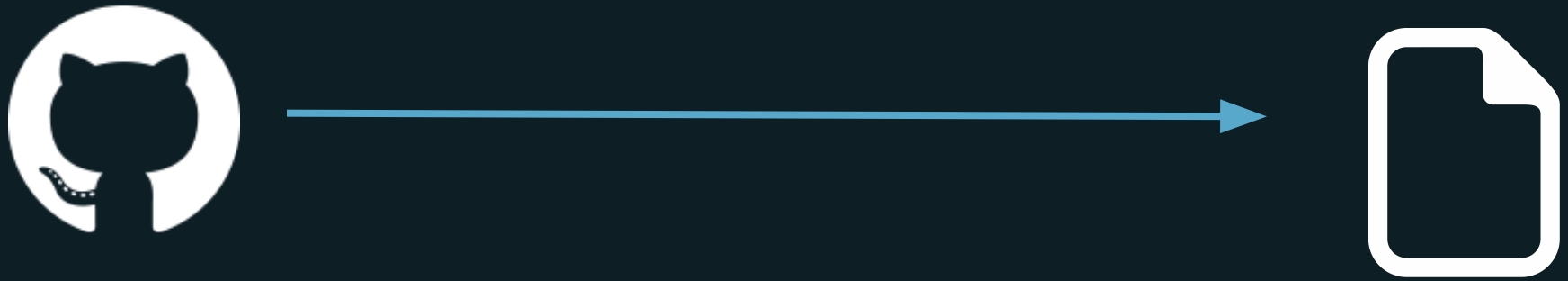
→ Identify {file type || protocol} the program parses

Inference: Assumption

File type and protocol inference are based on coverage

→ Correct input yields higher coverage

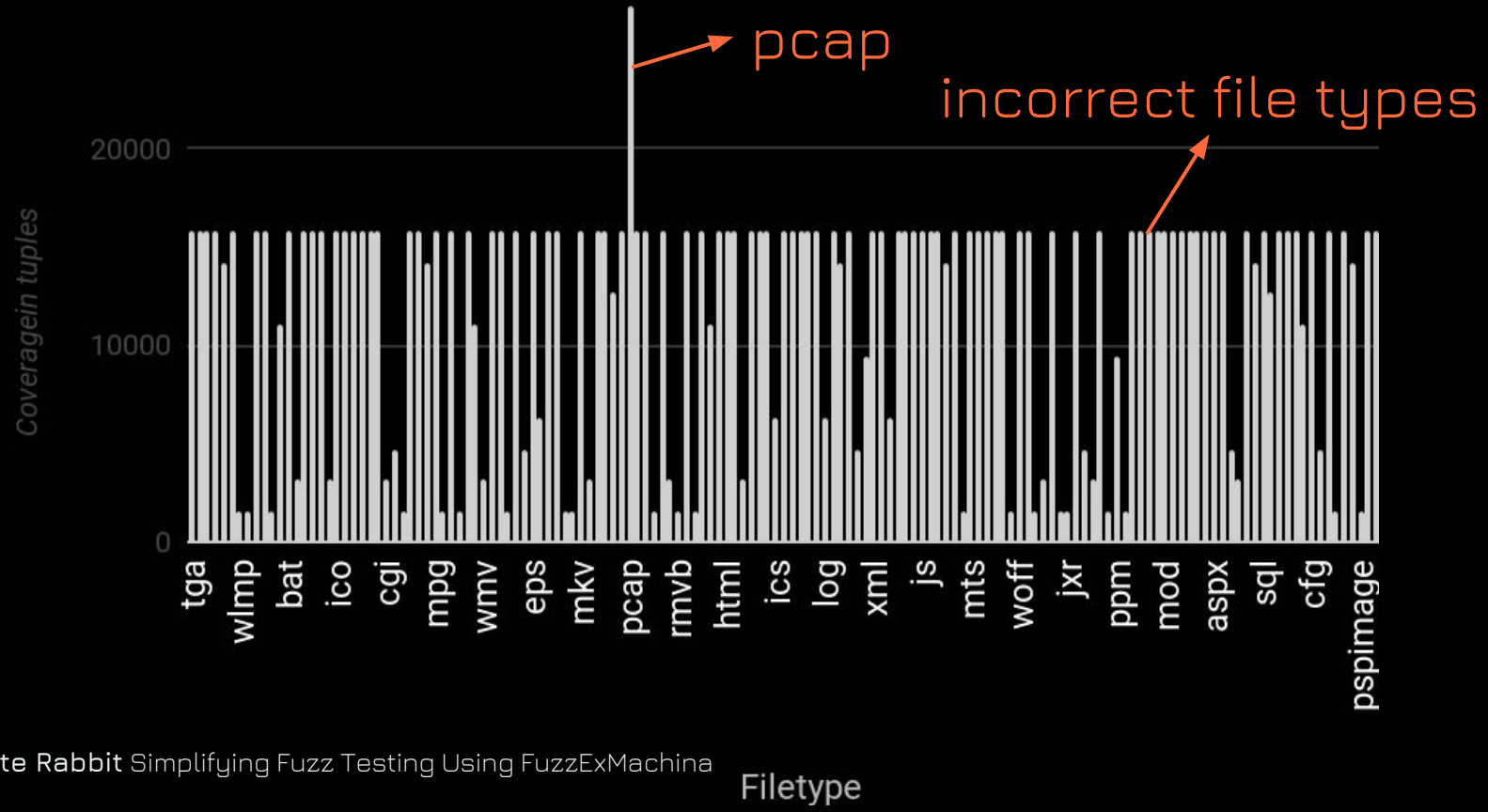
Seeds From...? Github!



Repos often include small, diverse test files that cover corner cases.

Coverage per File Type

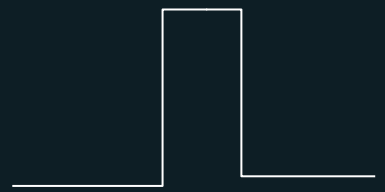
measured coverage for
tcpdump -nvr @@



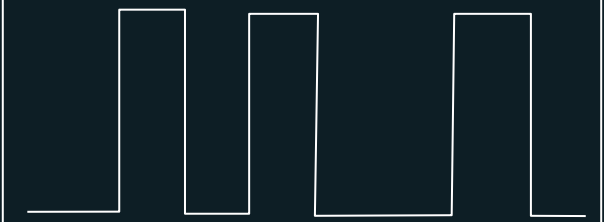
Coverage Distribution Patterns

Coverage Distribution for a binary follow patterns

Single File Type



Multiple File Types



Unclassifiable

The FExM Pipeline (4/6)

Dashboard

Crawl Binaries → Infer Inputs → Select Seeds → Fuzz! → Triage Crashes

FExM Fuzzing Stage

- Uses American Fuzzy Lop
- Using dictionaries where appropriate
- Use Sanitizers
- Network fuzzing via modified desock
- FExM schedules them round robin

The FExM Pipeline (5/6)

Dashboard

Crawl Binaries → Infer Inputs → Select Seeds → Fuzz! → Triage Crashes

What is triaging?

- We have way too many results!
- Categorize them!
- FExM leverages exploitable & afl-utils
- Classifying & Deduplicating

Triaging via ASAN

Security Criticality

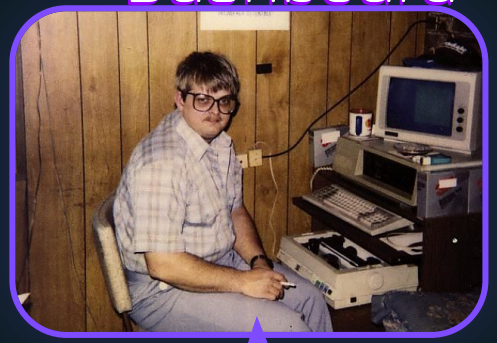
```
==3955==ERROR: AddressSanitizer:
heap-buffer-overflow on address
0x6100000001f5 at pc
0x5558ca920c3e bp 0x7ffd85b1b390
sp 0x7ffd85b1ab40
READ of size 16 at 0x6100000001f5
thread T0
    #0 0x5558ca920c3d
    #1 0x5558ca966533
    #2 0x5558ca96082a
```

3f6494e7343cb108505b2c8848e5c53d

MD5 Hash: Unique identifier

The FExM Pipeline (6/6)

Dashboard



Crawl Binaries → Infer Inputs → Select Seeds → Fuzz! → Triage Crashes

Dashboard

Show 10 entries

Search:

Package	Package Usage	Version	Package Worst Crash	Total number of crashing binaries	Total number of crashes	Status
bash	100.0	None	EXPLOITABLE	1	124	Now inferring invocation for /build/bash/repos/core-x86_64/src/bash-4.4/builtins/psize.aux
sqlite	100.0	None	EXPLOITABLE	1	12	Fuzzing /build/sqlite/repos/core-x86_64/src/sqlite-src-3240000/showjournal!
libidn	100.0	None	EXPLOITABLE	1	2	Fuzzing /build/libidn/repos/core-x86_64/src/libidn-1.34/examples/.libs/example5!
procps-ng	99.99	None	EXPLOITABLE	1	1	Fuzzing /build/procps-ng/repos/core-x86_64/src/procps-ng-3.3.15/.libs/pkill!
libcap-ng	100.0	None	None	0	0	Fuzzing /build/libcap-ng/repos/extra-x86_64/src/libcap-ng-0.7.9/utils/.libs/filecap!
gzip	100.0	None	None	0	0	Fuzzing /build/gzip/repos/core-x86_64/pkg/gzip/usr/bin/gzip!
mpfr	100.0	None	None	0	0	Fuzzable binaries detected:
linux-api-headers	100.0	None	None	0	0	Fuzzing /build/linux-api-headers/repos/core-any/src/linux-4.16/scripts/basic/fixdep!
libgpg-error	100.0	None	None	0	0	Now inferring invocation for /build/libgpg-error/repos/core-x86_64/src/libgpg-error-1.32/tests/t-lock
libmnl	99.99	None	None	0	0	Fuzzable binaries detected:

Showing 1 to 10 of 62 entries

Previous 1 2 3 4 5 6 7 Next

DEMO[FExMDashboard]

Introduction
FuzzExMachina
Findings
TimeWarp
Conclusion

FExM on a Large Scale

- Ran FExM against Arch Linux packages
  - Sorted by pkgstats ⇒ popularity index
  - Evaluated Top 500 packages
  - About 200 contained binaries
- Hardware
  - 32 CPU cores
  - 128 GB RAM
  - A few days of time

Day 0


```
(FExM) fuzz@sev$ fexm fuzz ./top500.json
```

Day 1

Day 2

- After 2 days of runtime
- 200 packages
- Crashes for 29 packages,
- 12 exploitable (automatically triaged)
- All of these packages have high popularity
  - sysctl (modifies kernel properties)
  - hyphen (does… hyphens? (Part of libreoffice))
  - gif2png (not popular) (but who doesn't like gifs?)
  - …

oops

```
=================================================================
==95802==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x621000001100 at pc
0x55902c3eb812 bp 0x7fff7ec84490 sp 0x7fff7ec83c40
READ of size 4082 at 0x621000001100 thread T0
    #0 0x55902c3eb811  (/build/procps-ng/repos/core-x86_64/src/procps-ng-3.3.15/.libs/sysctl+0x8d811)
    #1 0x55902c47e96a  (/build/procps-ng/repos/core-x86_64/src/procps-ng-3.3.15/.libs/sysctl+0x12096a)
    #2 0x55902c47cf45  (/build/procps-ng/repos/core-x86_64/src/procps-ng-3.3.15/.libs/sysctl+0x11ef45)
    #3 0x7f4a9a52b06a  (/usr/lib/libc.so.6+0x2306a)
    #4 0x55902c37ec79  (/build/procps-ng/repos/core-x86_64/src/procps-ng-3.3.15/.libs/sysctl+0x20c79)

0x621000001100 is located 0 bytes to the right of 4096-byte region [0x621000000100,0x621000001100)
allocated by thread T0 here:
    #0 0x7f4a9a471e1  (/build/procps-ng/repos/core-x86_64/src/procps-ng-3.3.15/.libs/sysctl+0xe91e1)
    #1 0x55902c37e85e  (/build/procps-ng/repos/core-x86_64/src/procps-ng-3.3.15/.libs/sysctl+0x11e85e)
    #2 0x7f4a9a52b06a  (/usr/lib/libc.so.6+0x2306a)

SUMMARY: AddressSanitizer: heap-buffer-overflow
Shadow bytes around the buffer:
  Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
```

What if FExM Inference Fails?

What if FExM Inference Fails?

- If we evaluate on a large scale:

Who cares, right? We have so many!

⇒ ¯_(ツ)_/¯

Not so fast!

Design Philosophy

Start automatic fall back to manual

What if FExM Inference Fails?

- If we evaluate on a large scale:

  ⇒ ¯_(ツ)_/¯ we can fall back to manual

- FExM Dashboard lists binaries that need attention
- Idea: easy for users to understand tools correctly

  Bonus: let user decide where to start fuzzing

  + Learn seeds on the way

Introduction
FuzzExMachina
Findings
TimeWarp
Conclusion

Unfuzzable


```
printf("Enter 8 Char Pwd> ");
fflush(stdout);
fgets(pw, (int) bufsize, stdin);
fgets(newline, sizeof(newline), stdin);
if (newline[0] != '\n') exit(printf("passwordsize incorrect"

printf("Reenter this Pwd> ");
fflush(stdout);
fgets(buf, (int) bufsize, stdin);
fgets(newline, sizeof(newline), stdin);
if (newline[0] != '\n') exit(printf("passwordsize incorrect"));

if (strlen(pw) != 8 || strncmp(buf, pw, bufsize) != 0) {
  fprintf(stderr, "Passwords needs to be at least 8 chars long and matching.\n");
  fflush(stderr);
  return 1;
}

strncat(complete, buf, PWSIZE);

printf("String to append to pwd> ");
fflush(stdout);
```

↓ crash below

pass1 == pass2
&& strlen() == 8
¡coverage does not help!
⇒ brute force 8 digits…

Fuzzer will run forever
A human can solve it in no time

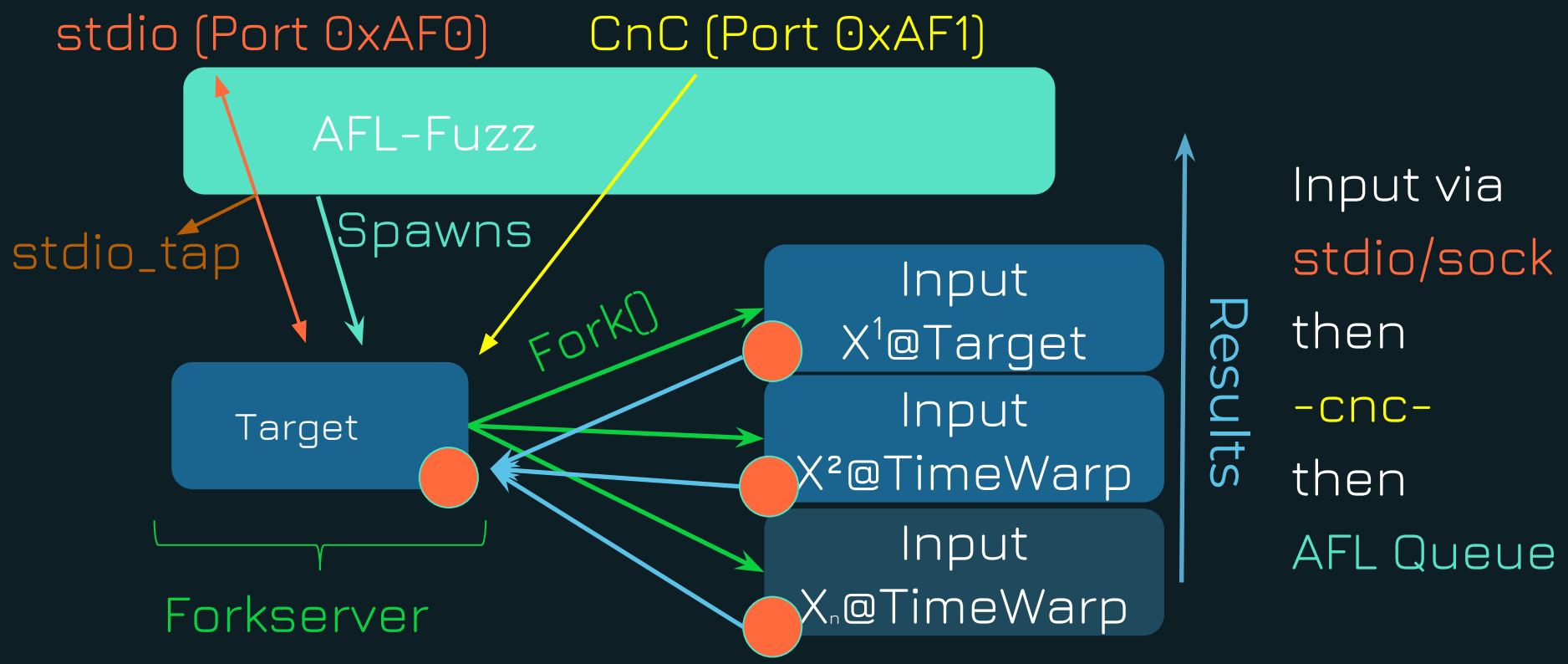
oops

```
=================================================================
==7221==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffde54f78a8 at pc 0x
bp 0x7ffde54f7650 sp 0x7ffde54f7648
WRITE of size 8 at 0x7ffde54f78a8 thread T0
    #0 0x55de61697f7f  (/build/gptfdisk/repos/extra-x86_64/pkg/gptfdisk/usr/bin/gdisk+0x15f57f)
    #1 0x55de61694f08  (/build/gptfdisk/repos/extra-x86_64/pkg/gptfdisk/usr/bin/gdisk+0x15cf08)
    #2 0x55de616bbddb  (/build/gptfdisk/repos/extra-x86_64/pkg/gptfdisk/usr/bin/gdisk+0x183ddb)
    #3 0x55de616b215b  (/build/gptfdisk/repos/extra-x86_64/pkg/gptfdisk/usr/bin/gdisk+0x17a15b)
    #4 0x55de616e5e98  (/build/gptfdisk/repos/extra-x86_64/pkg/gptfdisk/usr/bin/gdisk+0x1ade98)
    #5 0x7fca7be3b06a  (/usr/lib/libc.so.6+0x2306a)
    #6 0x55de6157a4f9  (/build/gptfdisk/repos/extra-x86_64/pkg/gptfdisk/usr/bin/gdisk+0x424f9)
```

TimeWarp

- Starts fuzzing at any given point in the execution
- Requires little technical knowledge
- Fully integrated in FExM Dashboard
  → Spawns Docker
  → Spawns TimeWarp with the target binary
  → Attaches Dashboard to stdio & cnc ports
- Allows easy generation of test cases manually

Details


stdio (Port 0xAF0)
CnC (Port 0xAF1)
AFL-Fuzz
stdio_tap
Spawns
Fork()
Target
Forkserver
Input
X¹@Target
Input
X²@TimeWarp
Input
Xₙ@TimeWarp
Results
Input via
stdio/sock
then
-cnc-
then
AFL Queue

OOPS

```
==5339==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60800000bff7 at p
0x7fca8b622676 bp 0x7ffc865155a0 sp 0x7ffc86514d48
READ of size 29 at 0x60800000bff7 thread T0
    #0 0x7fca8b622675 in memcmp (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x77675)
    #1 0x403cca in main /home/vincent/tmp/jpegoptim-1.4.4/jpegoptim.c:579
    #2 0x7fca8ac9f82f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
    #3 0x408768 in _start (/home/vincent/tmp/jpegoptim-1.4.4/jpegoptim+0x408768)

0x60800000bff7 is located 0 bytes to the right of 87-byte region [0x60800000bfa0,0x60800000bff7)
ated by thread T0 here:
ca8b643602 in malloc (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x98602)
079633  (/usr/lib/x86_64-linux-gnu/libjpeg.so.8+0x30633)
f4f  (<unknown module>)

r: heap-buffer-overflow ??:0 memcmp
u address:
fa fa fa fa fa fa fa fa
fa fa fa fa fa fa fa
fa fa fa fa fa
fa fa fa
fa
0[07]fa
```

AFL Timewarp

Sorry for all those gifs

DEMO[TimeWarp]

CnC: Start fuzzing

```
fuzzwarp/fuzzwarp on ᚠ master [?]
[I] → nc localhost 2801
Welcome to AFL Timewarp.
Start learning with "L"
reset to L and accept current input as Fuzzer input using "R" (repeat this multiple times),
then start Fuzzing with "F",
exit with "E".
F
```

stdio

```
fuzzwarp/fuzzwarp on ᚠ master [?]
[I] → nc localhost 2800
Enter 8 Char Pwd> TESTTEST
Reenter this Pwd> TESTTEST
String to append to pwd>
```

AFL-TW Output

```
american fuzzy lop

+- process timing ------------------------------------+
|        run time : 0 days, 0 hrs, 0 min, 13 sec       |  cycles done : 0       |
|   last new path : none seen yet                      |  total paths : 1       |
| last uniq crash : 0 days, 0 hrs, 0 min, 0 sec        | uniq crashes : 2       |
|  last uniq hang : none seen yet                      |   uniq hangs : 0       |
+- cycle progress -------------------+- map coverage -+-----------------------+
|  now processing : 0 (0.00%)         |    map density : 0.02% / 0.02%         |
| paths timed out : 0 (0.00%)         | count coverage : 1.00 bits/tuple       |
+- stage progress -------------------+- findings in depth -------------------+
|  now trying : havoc                 | favored paths : 1 (100.00%)            |
| stage execs : 36/1024 (3.52%)       |  new edges on : 1 (100.00%)            |
| total execs : 522                   | total crashes : 2 (2 unique)           |
|  exec speed : 225.6/sec             |  total tmouts : 0 (0 unique)           |
+- fuzzing strategy yields -----------+---------------+- path geometry --------+
|   bit flips : 0/32, 0/31, 0/29                      |    levels : 1          |
|  byte flips : 0/4, 0/3, 0/1                         |   pending : 1          |
| arithmetics : 0/221, 0/0, 0/0                       |  pend fav : 1          |
|  known ints : 0/28, 0/84, 0/44                      | own finds : 0          |
|  dictionary : 0/0, 0/0, 0/0                         |  imported : n/a        |
|       havoc : 0/0, 0/0                              | stability : 100.00%    |
|        trim : n/a, 0.00%                            +------------------------+
^C----------------------------------------------------+          [cpu000: 82%]
```

TimeWarp mode lets the user enter the correct passwords.

→ AFL then finds the bug in seconds!

Oops

```
==7276==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000034 at p
0x5563fa67a7dd bp 0x7ffc8bc12670 sp 0x7ffc8bc12668
READ of size 1 at 0x602000000034 thread T0
    #0 0x5563fa67a7dc  (/build/jhead/repos/community-x86_64/pkg/jhead/usr/bin/jhead+0x1377dc)
    #1 0x5563fa67aa88  (/build/jhead/repos/community-x86_64/pkg/jhead/usr/bin/jhead+0x137a88)
    #2 0x5563fa672fb3  (/build/jhead/repos/community-x86_64/pkg/jhead/usr/bin/jhead+0x12ffb3)
    #3 0x7f7aa19ec06a  (/usr/lib/libc.so.6+0x2306a)
    #4 0x5563fa572bc9  (/build/jhead/repos/community-x86_64/pkg/jhead/usr/bin/jhead+0x2fbc9)

SUMMARY: AddressSanitizer: heap-buffer-overflow
/build/jhead/repos/community-x86_64/pkg/jhead/usr/bin/jhead+0x1377dc)
```

Introduction

FuzzExMachina

Findings

TimeWarp

Conclusion

oops

```
=================================================================
==102039==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x63200001713e at pc
0x55898bb98da4 bp 0x7fffe449b030 sp 0x7fffe449b028
READ of size 1 at 0x63200001713e thread T0
    #0 0x55898bb98da3  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x14dda3)
    #1 0x55898bb7be6a  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x130e6a)
    #2 0x55898bb72706  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x127706)
    #3 0x55898bb714e6  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x1264e6)
    #4 0x7faf0db8906a  (/usr/lib/libc.so.6+0x2306a)
    #5 0x55898ba74719  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x29719)

0x63200001713e is located 0 bytes to the right of 92478-byte region [0x632000000800,0x63200001713e)
allocated by thread T0 here:
    #0 0x7faf0dc1f3a1  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0xf1c81)
    #1 0x55898bb98d9d  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x126dfd)
    #2 0x55898bb7bc8a  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x1261f0)
    #3 0x55898bb98da3  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x14dda3)
    #4 0x7faf0db8906a  (/usr/lib/libc.so.6+0x2306a)

SUMMARY: AddressSanitizer: heap-buffer-overflow
Shadow bytes around the buffer:
  0x63200001700a: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x63200001700b: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

OOPS

```
=================================================================
==17910==ERROR: AddressSanitizer: global-buffer-overflow on address 0x55b4d19e0910 at pc
0x55b4d17a0d8e bp 0x7ffe57681c90 sp 0x7ffe57681c88
WRITE of size 4 at 0x55b4d19e0910 thread T0
    #0 0x55b4d17a0d8d  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x14dd8d)
    #1 0x55b4d1783e6a  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x130e6a)
    #2 0x55b4d177a706  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x127706)
    #3 0x55b4d17794e6  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x1264e6)
    #4 0x7f93a799806a  (/usr/lib/libc.so.6+0x2306a)
    #5 0x55b4d167c719  (/build/bash/repos/core-x86_64/src/bash-4.4/support/man2html+0x29719)

0910 is located 48 bytes to the left of global variable 'maxtstop' defined in 'man2html.c:438:12'
0) of size 4
cated 0 bytes to the right of global variable 'tabstops' defined in 'man2html.c:437:12'
80
global-buffer-overflow
rc/bash-4.4/support/man2html+0x14dd8d)
ess:
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
f9 f9 f9
f9
00
```

Man2html Crashing Input


```
000000000000 -e "\n\n\n\n\nYOU REALLY SHOULD BE USING ys.auto or
better yet -sploit
BUT IF YOU MUST USE $0 at least use /S instead of just nc.
Packrat now has an option to do just that:
packrat -n /0000000000000000S
"
sleep 4
usage ()
{
echo "Usage: ${0}
```

[->Goes on like this for a bit, including CSS and more… (?)]

“This program is rather buggy, but in spite of that it often works.”

— *man2html.c:11*

Oops

```
=================================================================
==1596==ERROR: AddressSanitizer: stack-buffer-overflow on address
0x7ffdc30cdfe0 at pc 0x5620ecae914e bp 0x7ffdc30cddc0 sp 0x7ffdc30cd570
WRITE of size 259 at 0x7ffdc30cdfe0 thread T0
    #0 0x5620ecae914d in vsprintf (/usr/bin/mkafmmap+0xa514d)
    #1 0x5620ecae9487 in __interceptor_sprintf (/usr/bin/mkafmmap+0xa5487)
    #2 0x5620ecbc74f5  (/usr/bin/mkafmmap+0x1834f5)
    #3 0x5620ecbbefd3  (/usr/bin/mkafmmap+0x17afd3)
    #4 0x5620ecbbb00d  (/usr/bin/mkafmmap+0x177000d)
    #5 0x5620ecbb881a  (/usr/bin/mkafmmap+0x17481a)
    #6 0x55bfe9cf49 in __libc_start_main (/usr/lib/libc.so.6+0x20f49)
    #7 0x7fabb739 in pthread_getattr_np (/usr/bin/mkafmmap+0x77739)

Address 0x7ffdc30cdfe0 is located in stack of thread T0 at offset 288 in
frame
    #0 0x5620ecbb7f0 in mkafmmap+0x18322f)

  This frame has 1 object(s):
    [32, 288) '<unknown_module>' (line 0) <= Memory access at offset 288 overflows this
variable
HINT: this may be a false positive or your shadow memory is corrupted.
    (1) Your program uses some custom stack
    (2) Your program uses some custom stack
    #0 0x5620ecae914d in vsprintf (/usr/bin/mkafmmap+0xa514d)
```

Whatever Happened to the Bugs

Package	Binary invocation	Version	Crash Type	Status	GitHub Stars
enscript	mkafmmap @@		heap-buffer-overflow	no response	N/A
aircrack-ng	wpclean -nvr @@	1.2rc4	segmentation fault	fixed	131
catimg	catimg @@	2.4.0	global-buffer-overflow	“no proper repo”(?)	468
jpegoptim	jpegoptim @@	1.4.4	heap-buffer-overflow	was missing in Arch	671
jhead	jhead @@	3.00	heap-buffer-overflow	-> see next slide	N/A
libpng	pnm2png	1.6.34	stack-buffer-overflow	“send non-binary file”	267
CFITSIO	funpack @@	3.430	segmentation fault	fixed	N/A

oops

```
=================================================================
==7199==ERROR: AddressSanitizer: global-buffer-overflow on address 0x7f033ecd5f00 at pc 0
bp 0x7ffd175b0520 sp 0x7ffd175b0518
READ of size 4 at 0x7f033ecd5f00 thread T0
    #0 0x7f033eaae640  (/build/fribidi/repos/extra-x86_64/src/build/bin/../lib/libfribidi.so.0+0x15640)
    #1 0x7f033eaadaf4  (/build/fribidi/repos/extra-x86_64/src/build/bin/../lib/libfribidi.so.0+0x14af4)
    #2 0x55892c317357  (/build/fribidi/repos/extra-x86_64/src/build/bin/fribidi-caprtl2utf8+0x11a357)
    #3 0x7f033ecfa06a  (/usr/lib/libc.so.6+0x2306a)
    #4 0x55892c21a9c9  (/build/fribidi/repos/extra-x86_64/src/build/bin/fribidi-caprtl2utf8+0x1d9c9)

0x7f033ecd5f00 is a wild pointer.
ddressSanitizer: global-buffer-overflow
os/extra-x86_64/src/build/bin/../lib/libfribidi.so.0+0x15640)
```

Bug Reports

Date: Wed, 28 Mar 2018 20:51:01 +0000
To: Vincent Ulitzsch

Unfortunately, I just don't have time to work on it these days.

------ Original Message ------
From: "Vincent Ulitzsch"
Sent: 2018-03-28 2:50:06 PM
Subject: Command Line Heap Bufferoverflow
> During my research, I have found a heap-buffer-overflow [...]

Bugs

```
=================================================================
==7228==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000033 at pc
0x55ac99925f1a bp 0x7ffc283a9b70 sp 0x7ffc283a9b68
WRITE of size 1 at 0x602000000033 thread T0
    #0 0x55ac99925f19  (/build/hyphen/repos/extra-x86_64/src/hyphen-2.8.8/substrings+0x11df19)
    #1 0x7f215af5d06a  (/usr/lib/libc.so.6+0x2306a)
    #2 0x55ac99825c69  (/build/hyphen/repos/extra-x86_64/src/hyphen-2.8.8/substrings+0x1dc69)

SUMMARY: AddressSanitizer: heap-buffer-overflow
/build/hyphen/repos/extra-x86_64/src/hyphen-2.8.8/substrings+0x11df19)
```

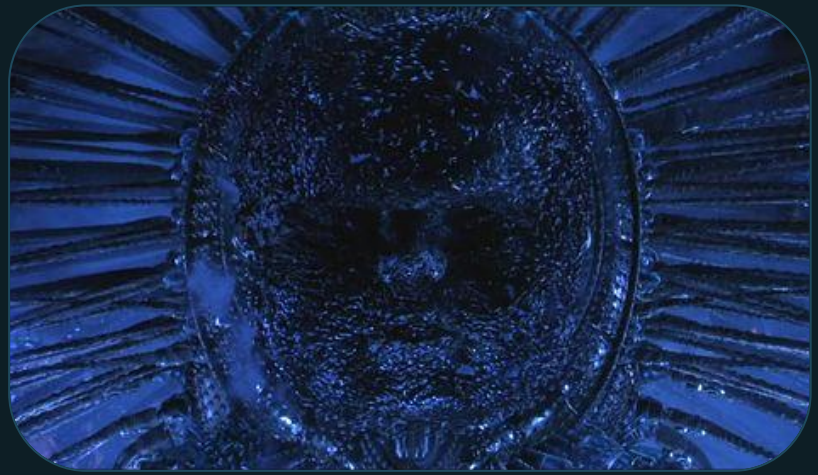
Future Work

- Obviously, a trained human can still do a lot better.

→ Let the machines take over!

- Add more repo backends
- Scale to GitHub?
- Fuzzing is still shallow.

  "We need to go deeper"



OOPS

```
==13817==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x621000006110 at p
0x7fe05d327ae2 bp 0x7fffa20956d0 sp 0x7fffa20956c8
READ of size 16 at 0x621000006110 thread T0
    #0 0x7fe05d327ae1 in soundtouch::TDStretchSSE::calcCrossCorr(float const*, float const*,
    double&) (/usr/lib/libSoundTouch.so.1+0x2bae1)
      #1 0x7fe05d316e04 in soundtouch::TDStretch::seekBestOverlapPositionFull(float const*)
    (/usr/lib/libSoundTouch.so.1+0x1ae04)
      #2 0x7fe05d316c5c in soundtouch::TDStretch::seekBestOverlapPosition(float const*)
    /usr/lib/libSoundTouch.so.1+0x1ac5c)
    0x7fe05d31854e in soundtouch::TDStretch::processSamples()
    libSoundTouch.so.1+0x1c54e)
    d31208f in soundtouch::SoundTouch::putSamples(float const*, unsigned int)
    Touch.so.1+0x1608f)
    (/usr/bin/soundstretch+0x12241e)
    libc_start_main (/usr/lib/libc.so.6+0x2306a)
    r/bin/soundstretch+0x21419)

    e right of 4112-byte region [0x621000005100,0x621000006110)

    long) (/usr/bin/soundstretch+0x11dee1)
    ffer::ensureCapacity(unsigned int)

    ndTouch.so.1+0x2bae1) in
    uble&)
```

TL;DR

FuzzExMachina (FExM)

- Automated fuzzing framework
- Clever tricks up its sleeve
- BYOB or fuzz distribution scale
- Found numerous bugs and crashes
- Free and open source

→ https://github.com/fgsect/fexm

Oops

```
=================================================================
==35788==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x607000000060 at pc
0x5642f134346c bp 0x7ffd2eb17c70 sp 0x7ffd2eb17c68
READ of size 4 at 0x607000000060 thread T0
    #0 0x5642f134346b  (/build/sqlite/repos/core-x86_64/pkg/sqlite/usr/bin/showwal+0x15c46b)
    #1 0x7fad883db06a  (/usr/lib/libc.so.6+0x2306a)
    #2 0x5642f123ff29  (/build/sqlite/repos/core-x86_64/pkg/sqlite/usr/bin/showwal+0x58f29)

0x607000000061 is located 0 bytes to the right of 65-byte region
[0x607000000020,0x607000000061)
allocated by thread T0 here:
    #0 0x5642f1308491  (/build/sqlite/repos/core-x86_64/pkg/sqlite/usr/bin/showwal+0x121491)
    #1 0x7fad883db084  (/usr/lib/libc.so.6+0x2306a)
    #2 0x5642f123d84b  (/build/sqlite/repos/core-x86_64/pkg/sqlite/usr/bin/showwal+0x155d84)

SUMMARY: AddressSanitizer: heap-buffer-overflow
Shadow bytes around the buggy address:
  0x0c0e80000000: 00 00 00 00 00 00 00 00
  0x0c0e80000010: 00 00 00 00 00 00 00 00
  0x0c0e80000020: 00 00 00 00 00 00 00 00
  0x0c0e80000030: 00 00 00 00 00 00 00 00
  0x0c0e80000040: 00 00 00 00 00 00 00 00
```

Try out FExM Today

- Available today
- Would not have been possible without
  - All the projects used in the repo
  - Ben Stock, Tommi Unruh, rc0r, jfoote, zardus, lcamtuf
  - hack-the-beach.com
- Help is always appreciated. :)

→ https://github.com/fgsect/fexm

0101
0100
0111
0010
0111
1001
0010
0000
0110
0110
0110
0101
0111
1000
0110
1101

Sound Bytes

"Simple memory corruptions are still way too widespread, even in popular software."

"FExM automates, facilitates and scales the fuzzing pipeline."

"There is no excuse not to fuzz-test software projects right from the start."

→ https://github.com/fgsect/fexm

@ibags
@vinulium
@domenuk

```
while (questions());

char buf[16];
strncpy(buf, ""
    "Thank you for your attention."
    "\n", sizeof(buf));
printf("%s", buf);
```

→ https://github.com/fgsect/fexm