Fuzzing with AFL++ - A Rookie's Guide

Let's fuzz a 'real' world C based target - *theory + practical*

by **Ajmal Moochingal**

null/OWASP Bangalore Chapter Meet
14 October 2023

Null - An open security community

Before we start

- I'm a Rookie too, learning & sharing things I learnt along the way. Pardon the mistakes 🙏.

  Would be a great help if you could point out mistakes/feedback if any.

- Please note any questions/feedback you have - we will open for discussion in the end.

Zooming Out - the bigger picture

- What is fuzzing ?
  - Theoretically → brute-forcing
  - Industry-lingo → brute-forcing done in a ‘smarter’ way
  - For finding bugs in an automated way

Example

- Fuzzing a Web App
    - Tools like ffuf

https://youtu.be/9Hik0xy9qd0?feature=shared&t=358

Warm-up - fuzzing with AFL++

- Fuzzing with AFL++
  - Mutational Fuzzing (Coverage Guided)
- AFL++ can fuzz?
  - Network services
  - C/C++ Targets **with** Source code
  - Some GUI programs
  - Binary targets **without** Source code

Ref :
https://github.com/AFLplusplus/AFLplusplus/blob/stable/docs/best_practices.md#targets-1
https://github.com/AFLplusplus/AFLplusplus/blob/stable/docs/ideas.md

Algorithm - *Coverage Guided Mutational Fuzzing*

1. Load the next input from the queue
2. Minimize the test case
3. Mutate the test case. If any mutant results in additional code coverage, add it to the queue. If the mutant results in a crash or hang, save it to disk for later inspection.
4. Go to step 1

Is this a technique good enough to find bugs? 🤔

What does the researchers in the industry say ?

"Fuzzing is King" - quoting Zardus (Yan)

(Coverage guided) Fuzzing is *undeniably* the best automated program analysis technique we have. (binary targets)

Since the inception of AFL in 2013 & AFL++ in 2017 the no. of CVEs every year in binary targets have taken a big jump

Ref:
https://youtu.be/K_2DAo5pPQQ?feature=shared&t=1094
https://lcamtuf.coredump.cx/afl/

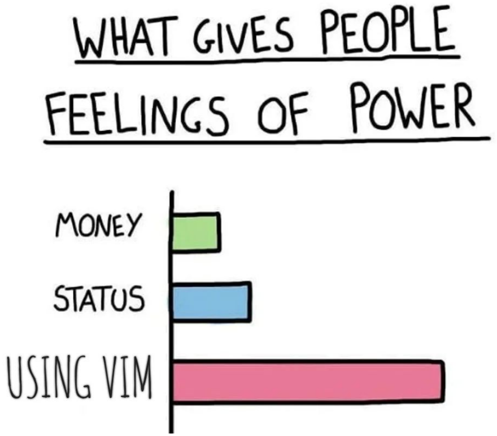
The kind of bugs we're looking for 🧐

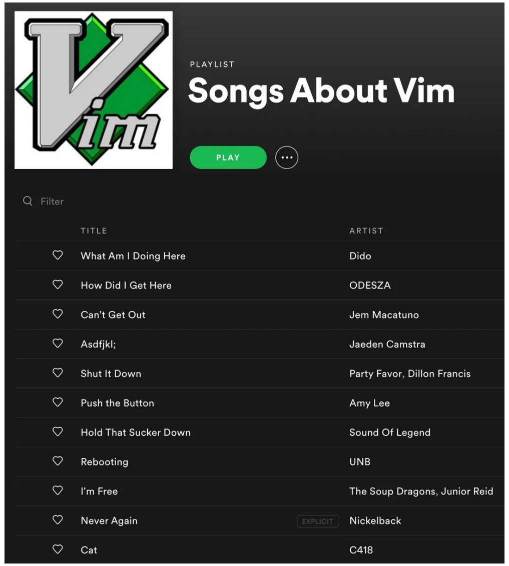
- Buffer Overflow
  - Stack
  - Heap
- Integer Overflow / Integer Underflow
- Out of bounds Read / Write
- User After Free / Double Free

Which may lead to ..

- DoS
- Information leak
- RCE

We install Vim on the door so that it was impossible to exit during quarantine





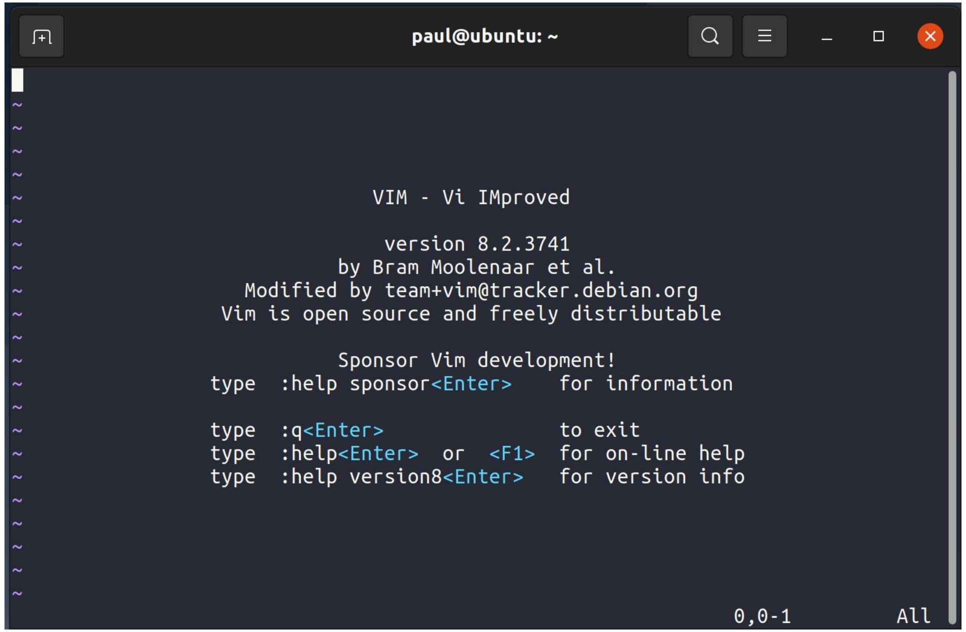
Songs About Vim

TITLE		ARTIST
What Am I Doing Here		Dido
How Did I Get Here		ODESZA
Can't Get Out		Jem Macatuno
Asdfjkl;		Jaeden Camstra
Shut It Down		Party Favor, Dillon Francis
Push the Button		Amy Lee
Hold That Sucker Down		Sound Of Legend
Rebooting		UNB
I'm Free		The Soup Dragons, Junior Reid
Never Again	EXPLICIT	Nickelback
Cat		C418

Fuzzing target - Vim

- Tons of complicated features
  ⇒ Lotsa bugs
- Very popular
- Open source - faster fuzzing
- 26 CVEs in 2023 alone

🙏 Taking a moment to remember the creator of Vim - Bram Moolenaar - who died 2 months ago. 😥
He was very active in triaging & fixing bugs until 2 months ago.



```
                      VIM - Vi IMproved

                       version 8.2.3741
                   by Bram Moolenaar et al.
           Modified by team+vim@tracker.debian.org
         Vim is open source and freely distributable

                  Sponsor Vim development!
        type  :help sponsor<Enter>    for information

        type  :q<Enter>               to exit
        type  :help<Enter>  or  <F1>  for on-line help
        type  :help version8<Enter>   for version info

                                          0,0-1         All
```

Major Steps

1. Compiling Vim - with AFL toolchain
2. Refining input
3. Fuzzing

0. Compiling AFL

→ Compile AFL++ from the source repo to have the latest version - https://aflplus.plus/building/

```
git clone https://github.com/AFLplusplus/AFLplusplus

make source-only # for fuzzing only targets with source code

make install
```

https://youtu.be/pqK7Kk4Z4YM

1. Compiling Vim with AFL's toolchain

1. Selecting compiler - use latest clang/llvm if possible
2. Use sanitizers
   a. ASAN
   b. MSAN
3. Compile the target !

```
wget https://github.com/vim/vim/archive/refs/tags/v9.0.2018.zip && unzip v9.0.2018.zip && cd vim-9.0.2018

CC=afl-clang-lto CXX=afl-clang-lto++ ./configure || CC=afl-clang-fast CXX=afl-clang-fast++ ./configure

export AFL_USE_ASAN=1

make
```

https://youtu.be/kHArA3V00AA Demo #2 →

2. Refining Input corpus

1. Collecting 'interesting' inputs
2. Making the input corpus unique - using *afl-cmin*
3. Minimizing all corpus files - using *afl-tmin*

```
mkdir -p ./input ./input_uniq ./output

afl-cmin -T all -i ./vim_input_crude -o ./input_uniq -- ./vim-9.0.2018/src/vim -u NONE -i NONE -n
-m -X -Z -e -s -S @@ -c :qa!

cd input_uniq; export AFL_MAP_SIZE=10000000; for i in *; do afl-tmin -i "$i" -o "../input/$i" --
../vim-9.0.2018/src/vim -u NONE -i NONE -n -m -X -Z -e -s -S @@ -c :qa! ; done
```

https://youtu.be/FAOL-idJpeU

Demo #3 →

directory structure

```
.
├── input
├── input_uniq
├── output
├── v9.0.2018.zip
├── vim-9.0.2018
└── vim_input_crude

5 directories, 1 file
```

3. Fuzzing

1. Run *afl-fuzz*
2. Keep an eye on the coverage
   a. How long to fuzz a target?

      "Basically, if no new path is found for a long time (e.g., for a day or a week), then you can expect that your fuzzing won't be fruitful anymore."

3. Use multiple core - run more instances parallely

```
afl-fuzz -D -i ./input -o ./output -- ./vim-9.0.2018/src/vim -u NONE -i NONE -n -m -X -Z -e -s -S @@ -c :qa!
screen -S main
afl-fuzz -M main -D -i ./input -o ./output -- ./vim-9.0.2018/src/vim -u NONE -i NONE -n -m -X -Z -e -s -S @@ -c :qa!
screen -S sub1
afl-fuzz -S sub1 -D -i ./input -o ./output -- ./vim-9.0.2018/src/vim -u NONE -i NONE -n -m -X -Z -e -s -S @@ -c :qa!
```

https://youtu.be/zpcJY4Ab9Hw

Demo #4 →

3+. Analysing/Triaging Crashes

- Minimize the size of crash input - *afl-tmin*
- Figuring out the root cause by analysing the crash - stacktrace
- -C flag – exploitability

https://youtu.be/LRkTfKifvfQ (Demo)

Strategies to do fuzzing better

- Persistent mode - 2x - 20x faster
- In memory ramdisk - avoid heavy disk I/O
- Running a combination of multiple fuzzers compatible with AFL++
- Being more creative with the input payloads
    - Aim for better coverage - interesting code paths.
        - Varied: it should represent a good coverage of the functionality of the program
        - Weird: it should trigger enough uncommon behavior to give the fuzzer a good starting point to trigger really rare behavior. Also have some normal stuff!
        - Small: AFL will eventually try to mutate every bit. Don't put in useless bits.

Ref: https://aflplus.plus/docs/fuzzing_in_depth/

What’s not covered

- Fuzzing binary-only black box target
  - QEMU, Frida modes
- Persistence mode fuzzing
- Using dictionaries
- *afl-fuzz* with -*C* flag - insights on exploitability.

Shout out 🙏 to these amazing resources out there.

https://pwn.college - Yan, Kanak & Team
> For learning system software security From scratch
> Covers fuzzing from scratch towards the final chapters.
> Youtube channel

https://fuzzing.in - Hardik
> Defcon '22 hands-on fuzzing workshop for free
> Youtube channel

Image credits :
https://www.youtube.com/@MrHardik05
https://www.youtube.com/@pwncollege
https://fuzzing.in

Questions/Feedback 🎤😀

Keep in touch - for QnA | Research