Advanced fuzzing workshop
2nd edition

Antonio Morales

// WHO AM I

#define speaker Antonio Morales

#define job Security Researcher at GitHub

#define twitter @nosoynadiemas

using namespace Hackfest;

int main(int argc, char argv[]){*


Security Lab

September 24, 2020

GHSL-2020-113: Command injection vulnerability in limdu - CVE-2020-4066

The `trainBatch` function has a command injection vulnerability. Clients of the Limdu library are unlikely to be aware of this, so they might unwittingly write code that contains a vulnerability

Kevin Backhouse

September 22, 2020

GHSL-2020-097: Missing hostname validation in twitter-stream - CVE-2020-24392

Missing hostname validation allows an attacker to perform a monster in the middle attack against users of the library.

Agustin Gianni

September 22, 2020

GHSL-2020-096: Missing hostname validation in tweetstream - CVE-2020-24393

Missing hostname validation allows an attacker to perform a monster in the middle attack

Security Lab

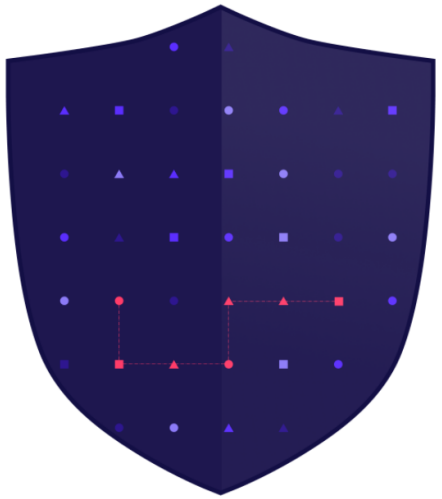
Bounties CodeQL Research Advisories Get Involved Events

GitHub Security Lab

Securing the world's software, together

GitHub Security Lab's mission is to inspire and enable the community to secure the open source software we all depend on.

Follow @GHSecurityLab



https://securitylab.github.com/

Security Lab

Bounties CodeQL Research Advisories Get Involved Events

August 27, 2020

C, Javascript, Python, Perl

Now you C me, now you don't: An introduction to the hidden attack surface of interpreted languages

Aimed at developers, in this series we introduce and explore the memory unsafe attack surface of interpreted languages.

Bas Alberts

August 11, 2020

Fuzzing, FreeRDP, AFL, CVE

Fuzzing sockets, part 2: FreeRDP

In this second installment, I'll delve into the research conducted on FreeRDP (http://www.freerdp.com/).

Antonio Morales

August 6, 2020

SSTI, CVE, RCE, Security

Room for Escape: Scribbling Outside the Lines of Template Security

in this Q&A with Alvaro Muñoz, dive in a recent research that uncovered more than 30 CVEs across 20 different CMS

@GHSecurityLab

Motivation

CVE-2019-20176	CVE-2019-14438	CVE-2019-14777	CVE-2020-4030	CVE-2020-9273
CVE-2020-9274	CVE-2019-14498	CVE-2019-14970	CVE-2020-11096	CVE-2019-14778
CVE-2020-9365	CVE-2019-14535	CVE-2020-13396	CVE-2020-11095	CVE-2020-11097
CVE-2020-6162	CVE-2019-14534	CVE-2020-13397	CVE-2020-4032	CVE-2019-14437
CVE-2020-6835	CVE-2019-14533	CVE-2020-13398	CVE-2020-4033	CVE-2019-14779
CVE-2020-9272	CVE-2019-14776	CVE-2020-11099	CVE-2020-4031	CVE-2020-11098

The aim of this workshop

Dumb Fuzzing **Smart Fuzzing**

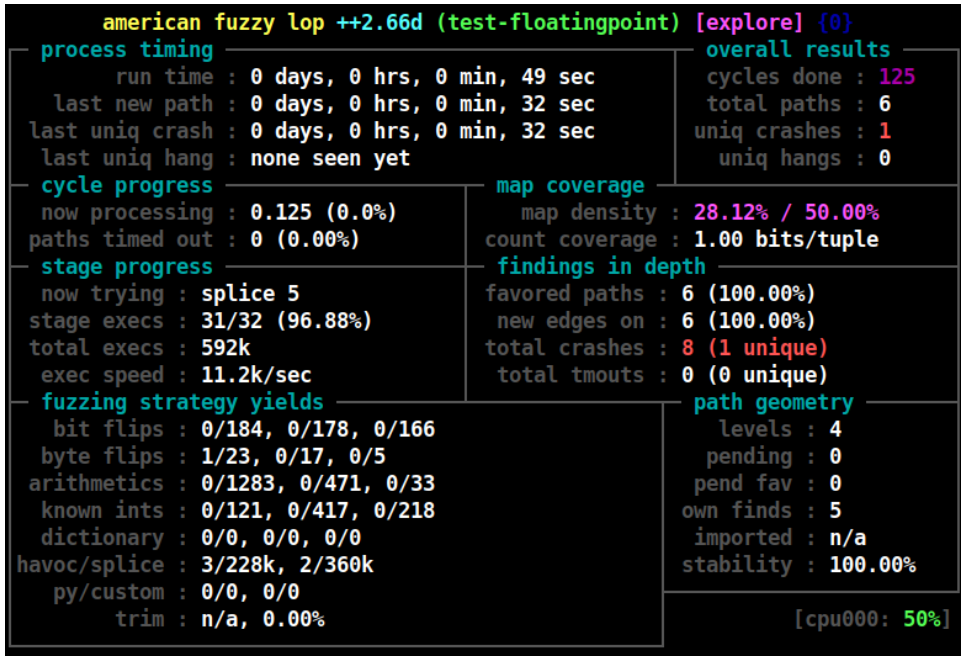
Workshop Format

- It’s a hands-on CTF-style workshop (learning-by-doing method).
- You will learn while facing the challenges. I’m here to **guide your learning**.

Tools - AFL

- All you need for the workshop is **AFL++ tool** running on a Linux system. Please, if you haven't download yet, do it now: https://github.com/AFLplusplus/AFLplusplus/releases

- Installing AFL++ -> https://github.com/AFLplusplus/AFLplusplus#building-and-installing-afl



RULES

Workshop repository

Find all you need for the workshop at
https://github.com/antonio-morales/Hackfest_Advanced_Fuzzing_Workshop

antonio-morales / Hackfest_Advanced_Fuzzing_Workshop

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 1 branch | 0 tags | Go to file | Add file | Code

antonio-morales Update README.md f05bf67 yesterday 7 commits

README.md Update README.md yesterday

README.md

Hackfest - Advanced Fuzzing Workshop

Previous editions

- EkoParty => https://github.com/antonio-morales/EkoParty_Advanced_Fuzzing_Workshop

Requirements

All you need for the workshop is:

- A **Telegram account**. You will need it to use it to send me your **questions/solutions**
- A running Linux system with an internet connection
- Latest version of **AFL++** installed on the system (https://github.com/AFLplusplus/AFLplusplus#building-and-installing-afl). You can download AFL++ source code at https://github.com/AFLplusplus/AFLplusplus/releases.

Rule 1

- Challenges are intended to be solved by fuzzing.

  … but you can use whatever method you want (good luck xD)

- There may be more than one correct solution.

- After each challenge, I will show my solution and I will explain it to you.

Rule 2

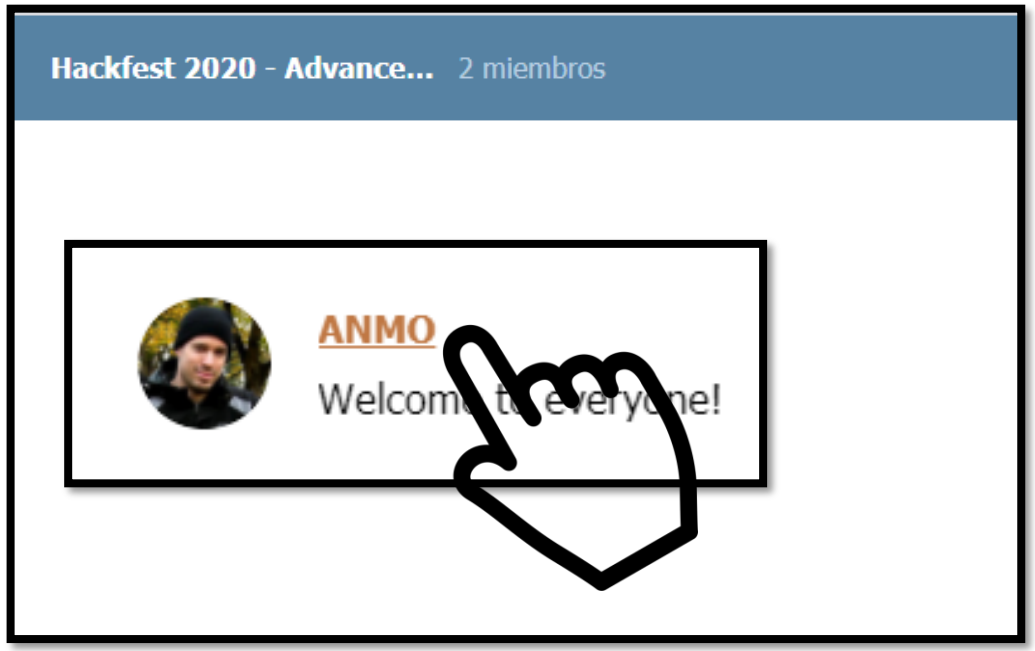
- There will be **3 different challenges**. The goal is to **find a reproducible bug** on each of them.

- We're looking for exploitable vulnerabilities. "Theoretical bugs" or code warnings are not welcome, sorry. Memory leaks are not valid solutions.

- In order to be the winner of a challenge, **you must provide a valid crash/PoC**.

Rule 3

- Please, don’t disclose your solutions
- **Send me a private message via Telegram**



Rule 4

- I will give you some hints and tips before and during the challenge
- I'll release a **new hint every 5-10 minutes** (approx)
- So, don't despair and keep trying!

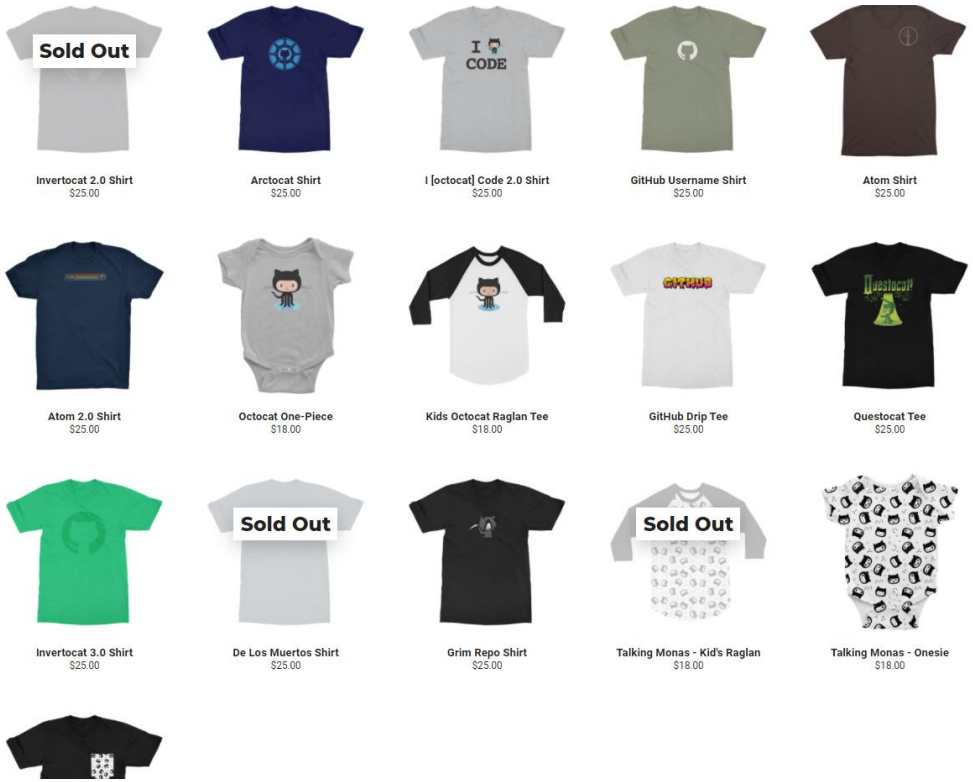
Awards

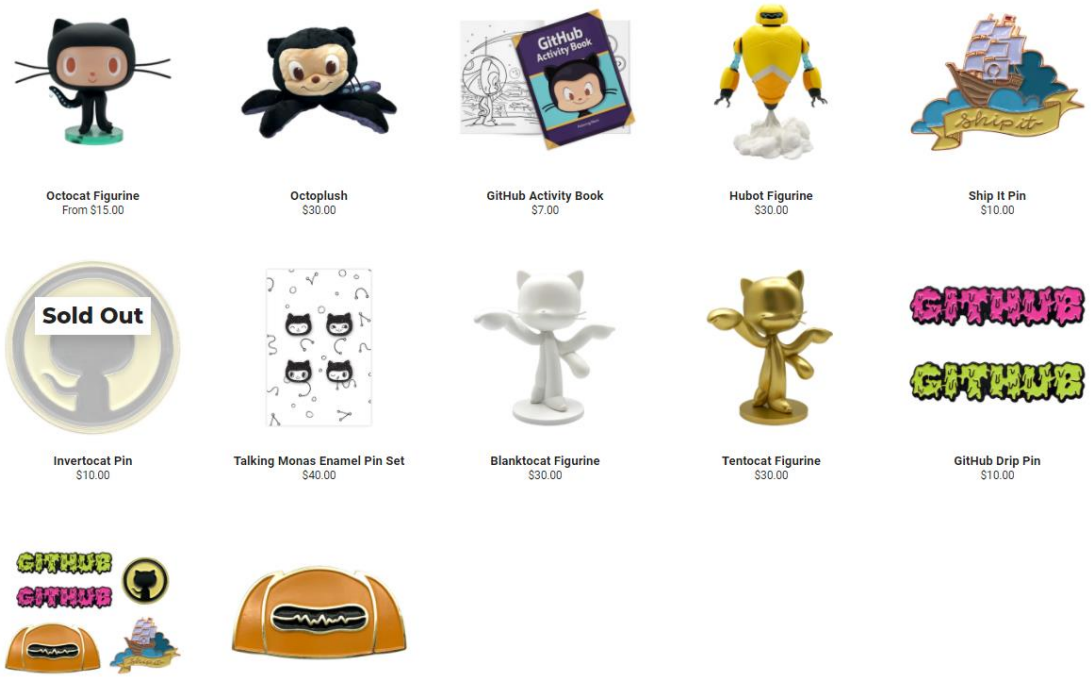
- There will be **2 winners for each challenge** (6 total winners).

- The winners will be the fastest ones in solving the challenge (find the vulnerability).

Prizes

GitHub SHOP





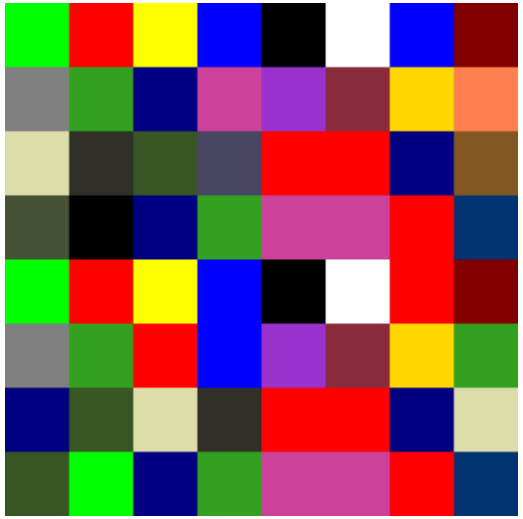
https://github.myshopify.com/

QUESTIONS / PREGUNTAS

Challenge 1 - ESIF (Extremely Stupid Image Format)

RELOAD (V2.0)

Get the code at: https://github.com/antonio-morales/Hackfest_Advanced_Fuzzing_Workshop/



Convert ESIF format to PPM format

Build:

```
> gcc HackFest1.c -o HackFest1 -w -lcrypto -lssl
```

Run:

```
> ./ HackFest1 example.ESIF output.ppm
```

You can find "Example.ESIF" in the same folder

Challenge 1 - ESIF (Extremely Stupid Image Format)

Ask me any doubt
via PM

Reminder

45 minutes

LET’S GO!!!

PASSWORD: **hello3487hello**

Challenge 1 – TIP

- That’s all you need to start fuzzing with AFL:

[COMPILE] **afl-gcc** src/HackFest1.c -o HackFest1 -w -lcrypto –lssl

[FUZZING] **afl-fuzz** -i ./AFL/afl_in/ -o './AFL/afl_out' -- ./HackFest1 @@ output

- If you have any problem, first try with:

> sudo apt-get install libssl-dev

Challenge 1 – Hint 1

- I strongly advise you to link your binary with **ASan (AddressSanitizer)** and **UBSan (Undefined Behavior Sanitizer)**
- To do this, add **-fsanitize=address,undefined** to your compile line
- Don’t forget to add **-m none** to your AFL command line

afl-gcc src/HackFest1.c -o HackFest1 -w -lcrypto -lssl **-fsanitize=address,undefined**

afl-fuzz **-m none** -i ./AFL/afl_in/ -o './AFL/afl_out' -- ./HackFest1 @@ output

Challenge 1 – Hint 2

- **Code coverage** can be really useful here. You can enable it adding **--coverage** to your compile line

```
> sudo apt install lcov
```

- You can find a **Code Coverage script** in the folder: **lcov.sh**

- You can collect code coverage, as follows:

```
>  chmod +x run_files
>  chmod +x lcov.sh
> ./lcov.sh
```

Then, open **./html_coverage/index.html** to view generated LCOV code coverage report

Challenge 1 – Hint 2

- Sometimes checksums can be a pain in the ass.
- Take a look at: https://securitylab.github.com/research/fuzzing-challenges-solutions-1

Challenge 1 – Hint 3

Looks like there are some obstacles in the code…

```
ch.Data = malloc(length);
memcpy(ch.Data, addr, length);

//CRC check
uint32_t crc = to_uint32(&ch.Header[4]);
if(crc != crc32(addr, length))
    goto error;

if(chunk_type(ch.Header, ch.Data, length) < 0)
    goto error;

return length+8;
```

```
data += 2;

if(glob.p == 0 || glob.d == 0)
    goto error;

MD5_Update(&context, svd, svdn-24);
MD5_Final(md5, &context);
if(memcmp(md5, data, 16))
    goto error;

data += 16;

if(memcmp(data, "\x20\x21", 2))
    goto error;
```

Challenge 1 – Hint 4

A little bit easier…

```
ch.Data = malloc(length);
memcpy(ch.Data, addr, length);

//CRC check
uint32_t crc = to_uint32(&ch.Header[4]);
//if(crc != crc32(addr, length))
    //goto error;

if(chunk_type(ch.Header, ch.Data, length) < 0)
    goto error;
```

```
if(glob.p == 0 || glob.d == 0)
    goto error;

MD5_Update(&context, svd, svdn-24);
MD5_Final(md5, &context);
//if(memcmp(md5, data, 16))
    //goto error;

data += 16;
```

Challenge 1 - Hint 5

- A **dictionary** can be useful… **sometimes**

afl-fuzz -t 500 -m none -i ../AFL/afl_in/ -o ../AFL/afl_out
-x ../AFL/mydict.txt -- ./hackfest1 @@

If you need more help, take a look at:
https://securitylab.github.com/research/fuzzing-challenges-solutions-1 *("Providing a custom dictionary")*

Challenge 1 - Hint 6

- Which items should be included in my dict?
- It’s hard to guess… XD

```
static const uint8_t DICTIONARY[5][4] = {
        { 0x2C, 0x61, 0x73, 0x75 },
        { 0x2D, 0x41, 0x63, 0x85 },
        { 0x74, 0x54, 0xDF, 0xDC },
        { 0x84, 0x83, 0xDF, 0xDC },
        { 0x98, 0x32, 0x67, 0x54 }
};
```

Challenge 1 – My Solution

```
1 "\x2C\x61\x73\x75"
2 "\x2D\x41\x63\x85"
3 "\x74\x54\xDF\xDC"
4 "\x84\x83\xDF\xDC"
5 "\x98\x32\x67\x54"
```

```
=================================================================
==78765==ERROR: AddressSanitizer: FPE on unknown address 0x5639c2756b4a (pc 0x5639c2756b4a bp 0x7ffed36de300
sp 0x7ffed36de2d0 T0)
    #0 0x5639c2756b49 in SYSCALL_CANCEL (/home/antonio/eclipse-workspace/HackFest1/hackfest1+0xbb49)
    #1 0x5639c2756f2e in last (/home/antonio/eclipse-workspace/HackFest1/hackfest1+0xbf2e)
    #2 0x5639c2758882 in print_output (/home/antonio/eclipse-workspace/HackFest1/hackfest1+0xd882)
    #3 0x5639c2758e0f in main (/home/antonio/eclipse-workspace/HackFest1/hackfest1+0xde0f)
    #4 0x7f1a400661e2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x271e2)
    #5 0x5639c275462d in _start (/home/antonio/eclipse-workspace/HackFest1/hackfest1+0x962d)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: FPE (/home/antonio/eclipse-workspace/HackFest1/hackfest1+0xbb49) in SYSCALL_CANCEL
==78765==ABORTING
```

Challenge 2 – QSSLANG (Quite Stupid Structured Language)

Get the code at: https://github.com/antonio-morales/Hackfest_Advanced_Fuzzing_Workshop/

```
#A humble tribute to IOCCC contests XD
#QSSLANG example
<QSS>
    <main> #Put here your cool comment
        <b>config</b> #Let's go
        <if><b true><n 12><SUM><n 87><endif><d 1605722137>
        <SUM>
        <AV><n 123456><n 7.8><text>hide</text><when><d 1605722111
        <MULT><n 123.21><n 7888.2><body><n 12></body>
        <t John><SIZE><n 321321>
        <MULT>
        <text>helloworld</text>
        ><b><comment><t Alex>
        0><SUB><n 122></comment><n 123.12>
        dif>[IF]<s cool challenge><MULT><n 500><n
        0><if><b!=text><t Hack><CONCAT><t Fest>
```

The most stupid structured-language never created

Build:

```
> gcc ./HackFest2.c  -w -o hackfest2
```

Run:

```
> ./hackfest2 Example.xml
```

Challenge 2 – QSSLANG (Quite Stupid Structured Language)

Reminder

50 minutes

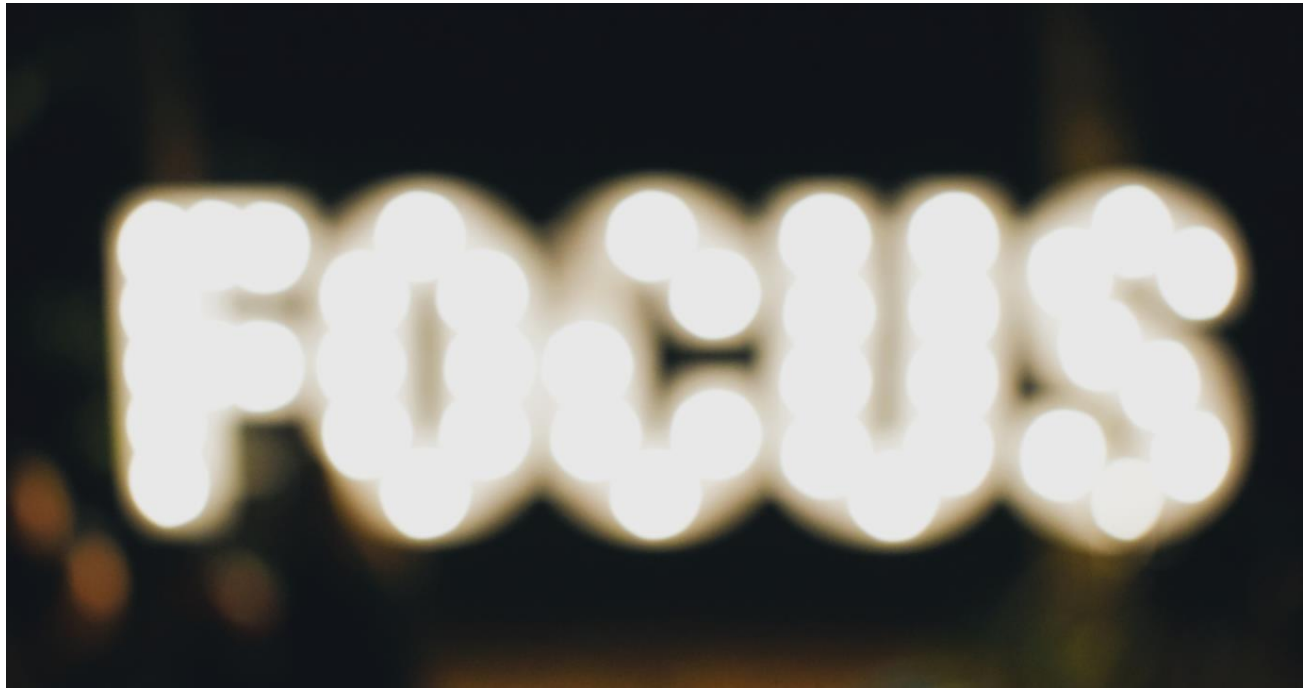
Ask me any doubt
via PM

LET’S GO!!!

PASSWORD: **fuzz9283fuzz**

Challenge 2 – Hint 1

- Pay attention to the GUI,
- Is there anything wrong?



Challenge 2 – Hint 2

```
            american fuzzy lop ++2.66c (hackfest2) [explore] {0}
┌─ process timing ──────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 1 min, 47 sec        │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec         │  total paths : 268    │
│ last uniq crash : none seen yet                       │ uniq crashes : 0      │
│  last uniq hang : none seen yet                       │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 0.1 (0.0%)        │    map density : 4.95% / 12.90%        │
│ paths timed out : 0 (0.00%)         │ count coverage : 2.58 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : bitflip 4/1           │ favored paths : 1 (0.37%)              │
│ stage execs : 484/5229 (9.26%)      │  new edges on : 157 (58.58%)           │
│ total execs : 22.0k                 │ total crashes : 0 (0 unique)           │
│  exec speed : 217.5/sec             │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 221/5232, 42/5231, 0/0                │    levels : 2          │
│  byte flips : 0/0, 0/0, 0/0                         │   pending : 268        │
│ arithmetics : 0/0, 0/0, 0/0                         │  pend fav : 1          │
│  known ints : 0/0, 0/0, 0/0                         │ own finds : 267        │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│havoc/splice : 0/0, 0/0                              │ stability : 24.47%     │
│   py/custom : 0/0, 0/0                              ├────────────────────────┘
│        trim : 0.00%/314, n/a                        │          [cpu000: 50%]
└─────────────────────────────────────────────────────┘                  ^C
```

Challenge 2 – Hint 3

- **Stability**: *it measures the consistency of observed traces. If a program always behaves the same for the same input data, it will earn a score of 100%. When the value is lower but still shown in purple, the fuzzing process is unlikely to be negatively affected. If it goes into red, you may be in trouble, since AFL will have difficulty discerning between meaningful and "phantom" effects of tweaking the input file*.

Challenge 2 – Hint 4

- **Life can be random…**

Challenge 2 – Hint 5

- Ok… That's what you need:

```
void vnsi(){

    struct timespec start;

    clock_gettime( CLOCK_REALTIME, &start);

    //srandom(start.tv_nsec); //RESTORE
    srandom(1);
}
```

- We should avoid high drops in stability

Challenge 2 – Hint 6

Can you find the differences?

Binary formats

Vs
Bit/bytes mutations

Text-based file formats

Vs
Bit/bytes mutations

Challenge 2 – Hint 7

- Maybe binary mutations are not the best for text-based file formats

- Take a look at: https://securitylab.github.com/research/fuzzing-software-2 *("Custom mutators")*

Challenge 2 – Hint 8

- Have you ever heard of **RADAMSA**?
- https://github.com/AFLplusplus/AFLplusplus/tree/stable/custom_mutators/radamsa

Challenge 2 – Hint 9

Too slow? You can try disabling ASAN and UBSAN

Challenge 2 – Hint 10

“MOPT is an excellent mutator…”

Challenge 2 – My Solution

- Using **MOPT mutator** you can find the bug **in about 10 minutes**

```
            american fuzzy lop ++2.66c (hackfest2) [explore] {3}
┌─ process timing ──────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 11 min, 39 sec       │  cycles done : 4      │
│   last new path : 0 days, 0 hrs, 0 min, 1 sec         │  total paths : 1500   │
│ last uniq crash : 0 days, 0 hrs, 0 min, 42 sec        │ uniq crashes : 1      │
│  last uniq hang : none seen yet                       │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 501.0 (33.4%)     │    map density : 1.31% / 2.33%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 5.09 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : MOpt-havoc            │ favored paths : 115 (7.67%)            │
│ stage execs : 3166/4096 (77.29%)    │  new edges on : 192 (12.80%)           │
│ total execs : 1.12M                 │ total crashes : 1 (1 unique)           │
│  exec speed : 1149/sec              │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 0/0, 0/0, 0/0                         │    levels : 8          │
│  byte flips : 0/0, 0/0, 0/0                         │   pending : 1500       │
│ arithmetics : 0/0, 0/0, 0/0                         │  pend fav : 115        │
│  known ints : 0/0, 0/0, 0/0                         │ own finds : 1499       │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│havoc/splice : 1271/544k, 0/0                        │ stability : 100.00%    │
│   py/custom : 0/0, 0/0                              ├────────────────────────┘
│        trim : 1.94%/63.1k, n/a                      │          [cpu003: 87%]
└─────────────────────────────────────────────────────┘          ^C
```

Challenge 2 – My Solution

```
=================================================================
==77178==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffff9a0b830 at pc 0x55e62
47118f4 bp 0x7ffff9a0b530 sp 0x7ffff9a0b520
WRITE of size 1 at 0x7ffff9a0b830 thread T0
    #0 0x55e6247118f3 in setall src/HackFest2.h:502
    #1 0x55e624717926 in function4 src/HackFest2.c:2474
    #2 0x55e6247b7d2f in check_time src/HackFest2.h:570
    #3 0x55e6247b7d2f in Process src/HackFest2.c:2618
    #4 0x55e6246e27fa in main src/HackFest2.c:2682
    #5 0x7ff4dcf021e2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x271e2)
    #6 0x55e6246e2d2d in _start (/home/antonio/eclipse-workspace/HackFest2/hackfest2+0x53d2d)

Address 0x7ffff9a0b830 is located in stack of thread T0 at offset 160 in frame
    #0 0x55e6247b784f in Process src/HackFest2.c:2602

  This frame has 1 object(s):
    [32, 160) 'ascii' (line 2605) <== Memory access at offset 160 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swa
pcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow src/HackFest2.h:502 in setall
Shadow bytes around the buggy address:
  0x10007f3396b0: 00 00 00 00 f1 f1 f1 f1 00 f3 f3 f3 00 00 00 00
  0x10007f3396c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007f3396d0: 00 00 00 00 f1 f1 f1 f1 00 f2 f2 f2 00 f3 f3 f3
  0x10007f3396e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007f3396f0: 00 00 f1 f1 f1 f1 00 00 00 00 00 00 00 00 00 00
```

Challenge 2 – My Solution

A **Stack buffer overflow** vulnerability

```
#QSSLANG example
<QSS>
    <main> #Put here your cool comment
        <b>config</b> #Let's go
        <if><b true><n 12><SUM><n 87><endif><d 1605722137>
        <SUM>
        <AV><n 123456><n 7.8><text>hide</text><when><d 1605722111>
        <MULT><n 123.21><n 7888.2><body><n 12></body>
        <t John><SIZE><n 321321>
        <MULT>
        <text>helloworld</text>
        [IF]<n 123><SIZE><b><comment><t Alex>
            [THEN]<n 1000><SUB><n 122></comment><n 123.12>
        <SIZE><n 144><endif>[IF]<s cool challenge><MULT><n 500><n 100>[ELSE]
        <n 75.5><SUM><n
        122.212345678901234567890123456789012345678901234567890123456789012345
        678901234567890123456789012345678901234567890123456789012345678901234567
        678901234567890123456789012345678901234567890123456789012345678901234567
        678901234567890123456789012345678901234567890>
        <i 100><SUM><i 50><if><b!=text><t Hack><CONCAT><t Fest>
    </main>
</QSS>
```

Challenge 2 – My Solution

You can trigger with:

SIZE tag && 3rd argument

&& == float size(3rd argument) > 128

```
#QSSLANG example
<QSS>
    <main> #Put here your cool comment
        <b>config</b> #Let's go
        <if><b true><n 12><SUM><n 87><endif><d 1605722137>
        <SUM>
        <AV><n 123456><n 7.8><text>hide</text><when><d 1605722111>
        <MULT><n 123.21><n 7888.2><body><n 12></body>
        <t John><SIZE><n 321321>
        <MULT>
        <text>helloworld</text>
        [IF]<n 123><SIZE><b><comment><t Alex>
            [THEN]<n 1000><SUB><n 122></comment><n 123.12>
        <SIZE><n 144><endif>[IF]<s cool challenge><MULT><n 500><n 100>[ELSE]
        <n 75.5><SUM><n
        122.212345678901234567890123456789012345678901234567890123456789012345
        678901234567890123456789012345678901234567890123456789012345678901234
        678901234567890123456789012345678901234567890123456789012345678901234
        678901234567890123456789012345678901234567890>
        <i 100><SUM><i 50><if><b!=text><t Hack><CONCAT><t Fest>
    </main>
</QSS>
```

Challenge 2 – Based on

https://hackerone.com/reports/480883

```
1  <?xml version="1.0" encoding="0123456789012345678901234567890123456789012345678901234567890
2  <NotepadPlus>
3      <!-- The key words of the supported languages, don't touch them! -->
4      <Languages>
5          <Language name="normal" ext="txt" />
6          <Language name="actionscript" ext="as mx" commentLine="//" commentStart="/*" commen
7              <Keywords name="instre1">add for lt tellTarget and function ne this break ge ne
8              <Keywords name="type1">arguments constructor class dynamic false extends implem
9          </Language>
10         <Language name="
11             <Keywords na
12         </Language>
13         <Language name="
14             <Keywords na
15             <Keywords na
16             <Keywords na
17             <Keywords na
18             <Keywords na
19             <Keywords na
20         </Language>
21         <Language name="
22             <Keywords na
23         </Language>
24         <Language name="
```

Antonio (ammm)

188	-	5.60	92nd	15.00	83rd
Reputation	Rank	Signal	Percentile	Impact	Percentile

96

#480883 **Stack overflow in XML Parsing**

Share:

State	Resolved (Closed)	Severity	High (8.1)
Disclosed	August 25, 2019 2:50pm +0200	Participants	
Reported To	Notepad++	Visibility	Disclosed (Full)
Weakness	Stack Overflow		
Bounty	$2,862.23		

Challenge 3 - My sweet parser

- I will publish it next Monday at: https://github.com/antonio-morales/Hackfest_Advanced_Fuzzing_Workshop/ and it **will last one week**.

- I will announce Challenge 3 winners next week ☺

- If you have any doubt on it, send me a pm via Twitter @nosoynadiemas

CONCLUSION

Conclusion

Don’t waste fuzzing iterations. Use your brain first

THE END

THANK YOU!
GRACIAS!

ASK ME ANYTHING

Antonio Morales Maldonado

Twitter: @nosoynadiemas
Email: antoniomoralesmaldonado@gmail.com