

Modern Era of Reverse Engineering

A Demonstration Collection by Melih Tolga Sahin

\$ whoami

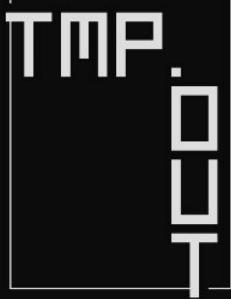
- Currently: DevSecOps Engineer at



- Author @ devilinside.me, tmp0ut.sh
- Exec. @ TTMO
- Worked as :

Penetration Tester, Reverse Engineer,
Threat Intelligence Analyst, Vulnerability
Researcher, Application Security
Engineer, Software Developer

Public CVEs: CVE-2024-26343, CVE-2023-1605



MALWARE ARENA

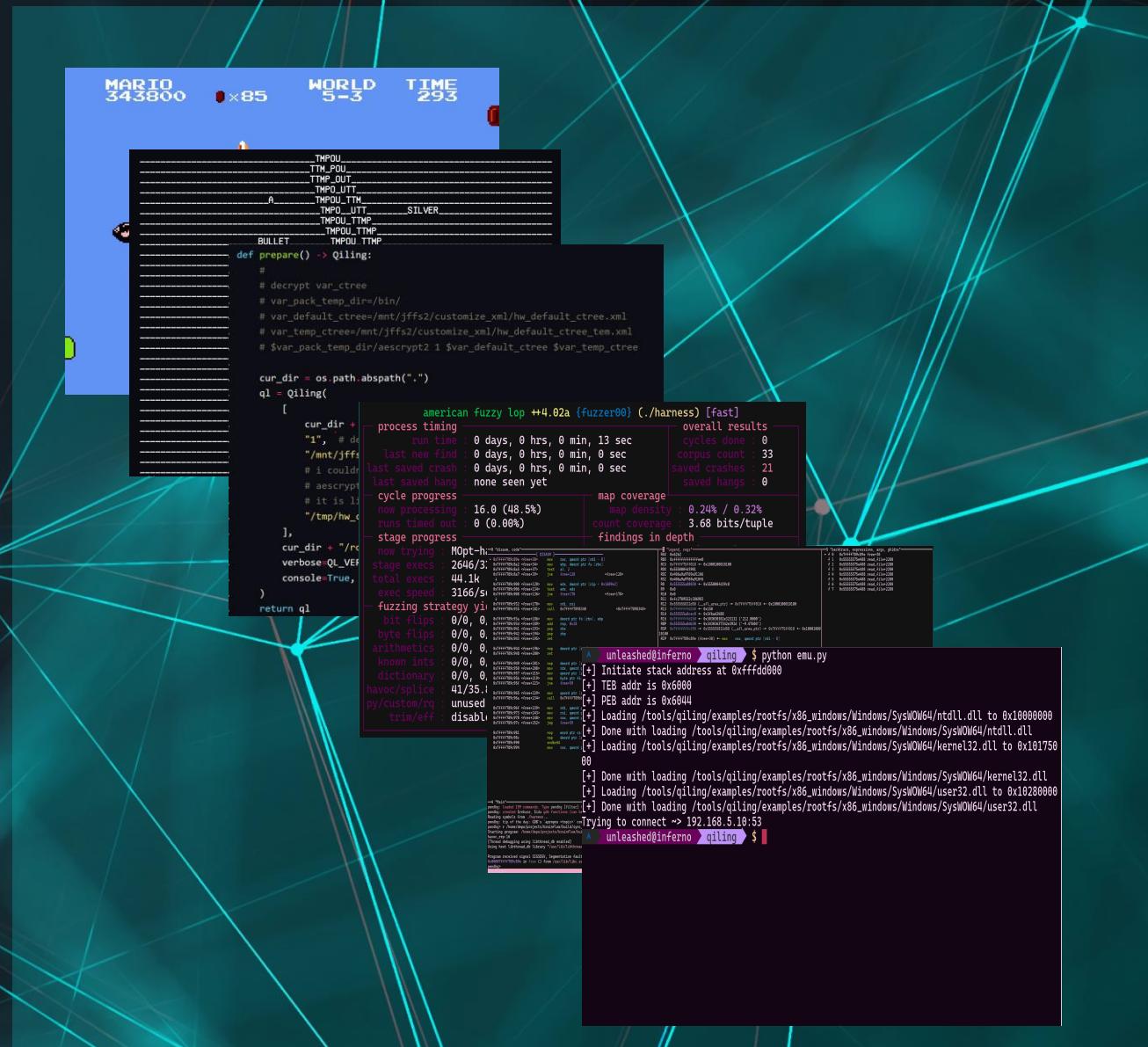
Mentors



malwarearena.com

\$ Contents

- Quick glimpse of daily routines of RE
- Tool-Assisted Superplay (TAS)
- TAS Example 1:
 - CVE-2015-2797 experience without an Airties Modem
- TAS Example 2:
 - Emulating Huawei OptiXstar's Encryption binary without a real Huawei ONT
- TAS Example 3:
 - A dynamic analysis of a msfvenom payload
- Fuzzing
- Fuzzing Campaign Example 1:
 - A simple harness for a EEG, EMG, ECG biosensors library.
- Fuzzing Campaign Example 2:
 - Fuzzing ELF Parsers with a custom mutator
- Fuzzing Campaign Example 3:
 - Fuzzing complex Instant Messaging Applications' data structures via Structure-Aware Fuzzing

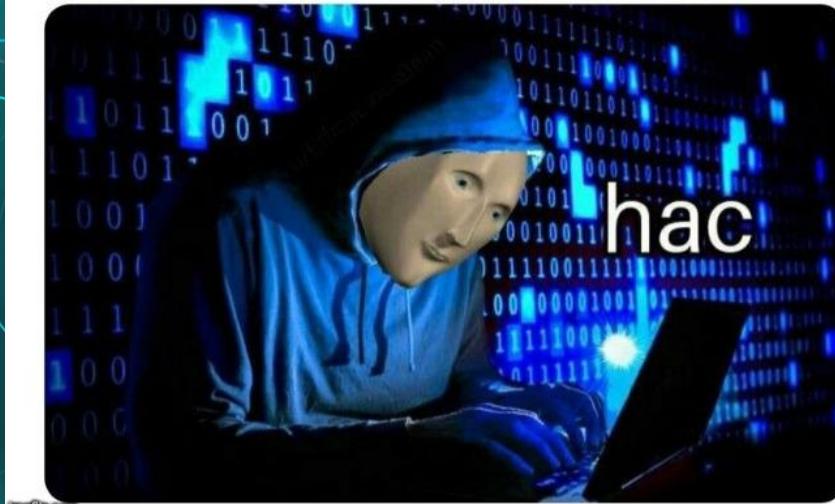




Daily RE Tasks (Almost)

- Reading Assembly Codes
- Reading Source Codes (If it is available)
- Control-flow unflattening
- Decrypt actual payloads
- Finding Interesting Functions
- Dealing with VMs (and other protections)
- Debugging the program to find out Input Flow
- Finding out the accepted format of user input
- Fuzzing
- Writing an actual exploit for the target

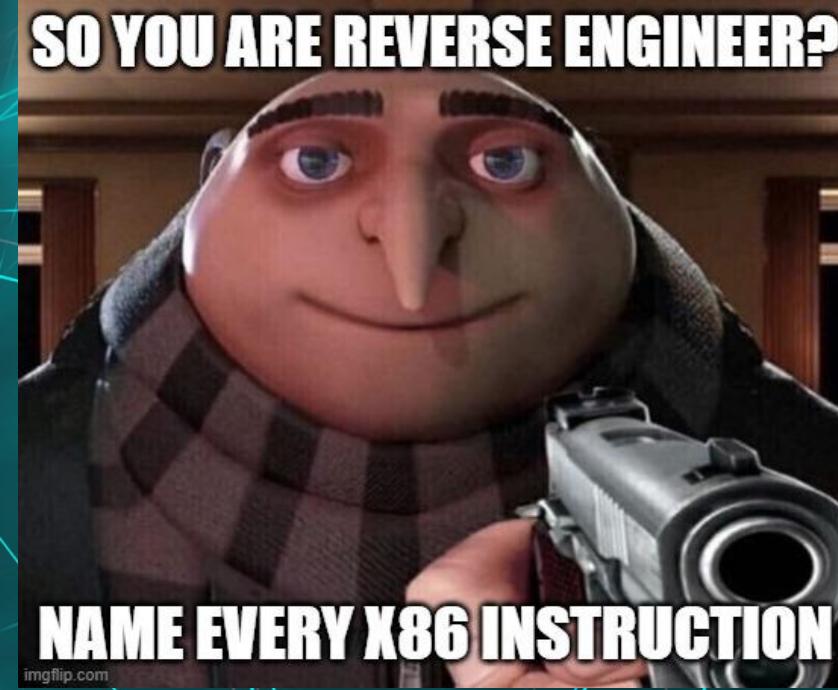
**When you use gdb to see
the assembly code of a program:**





Daily RE Tasks (Mentioned)

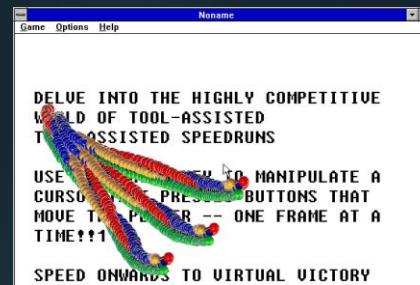
- Reading Assembly Codes
- Reading Source Codes (If it is available)
- Control flow unflattening
- Decrypt actual payloads
- Finding Interesting Functions
- Dealing with VMs (and other protections)
- Debugging the program to find out Input Flow
- Finding out the accepted format of user input
- Fuzzing
- Writing an actual exploit for the target





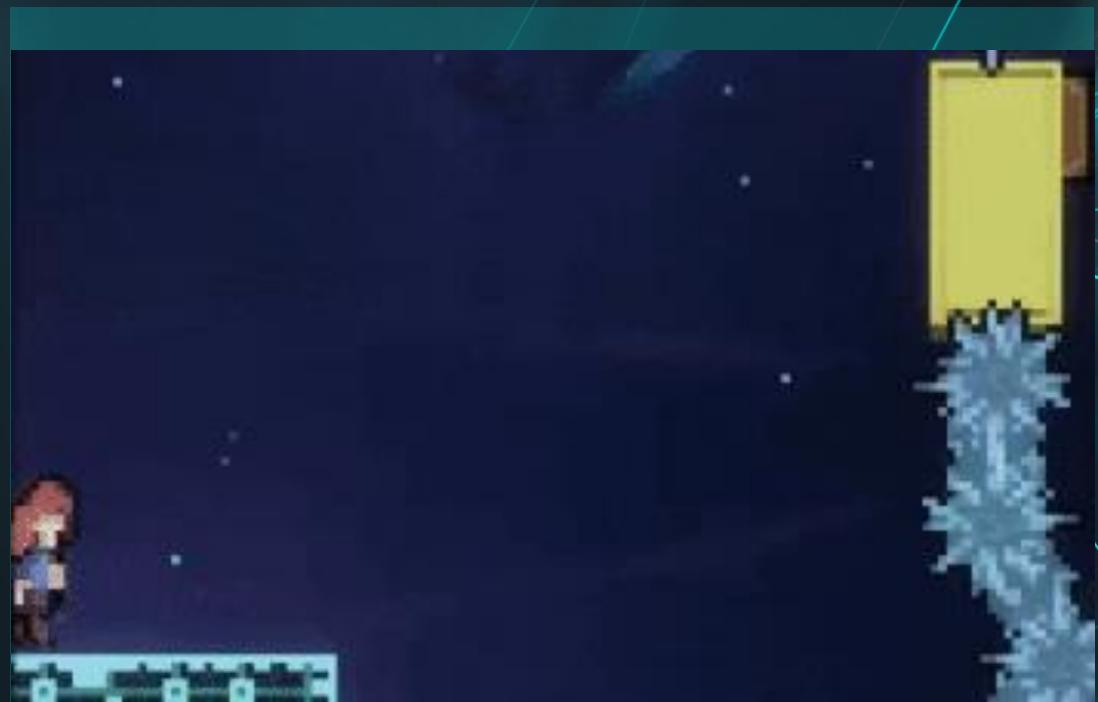
Reverse Engineer Issues

- Psychologically exhausting (optional)
- Time consuming
- If there is no purchased tool already, requires more manual work
- Not being able to automate routines



Tool-Assisted Superplay (TAS)

CELESTE (2018)



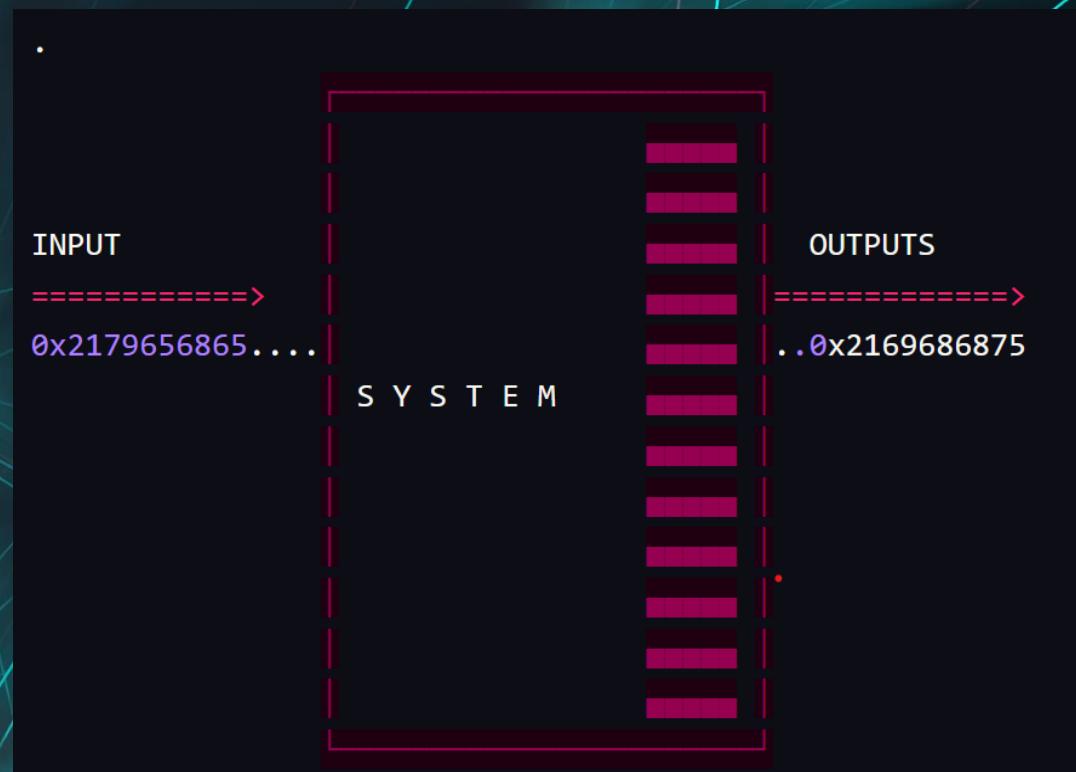
Mario Bros. (1983)





Super Simplification

Common ground between TAS Runners and Vulnerability Researchers

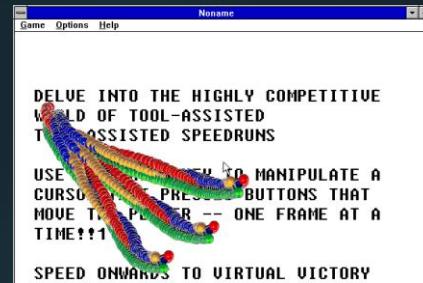




TAS Example: CVE-2015-2797

Stack-based buffer overflow in AirTies Air 6372, 5760, 5750, 5650TT, 5453, 5444TT, 5443, 5442, 5343, 5342, 5341, and 5021 DSL modems with firmware 1.0.2.0 and earlier allows remote attackers to execute arbitrary code via a long string in the redirect parameter to cgi-bin/login.

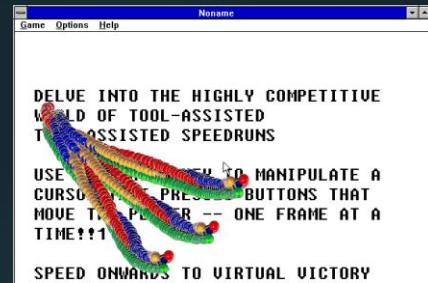




TAS Example: CVE-2015-2797



- This is an Airties Air 5750



TAS Example: CVE-2015-2797

```
unleashed@inferno ~airties5630> ! dev x + $ qltool run -f /home/qiling_projects/Airties/5650v3TT/squashfs-root --strace
[=] [mman.py:200] mmap(0x0, 0x14, 0x3, 0x802, -1, 0) = 0x774bf000
[=] [fcntl.py:44] open('/lib/libui.so', 0x0, 0o0) = 3
[=] [stat.py:322] fstat(3, 0x7ff3c130) = 0
[=] [mman.py:200] mmap(0x0, 0x1000, 0x3, 0x802, -1, 0) = 0x774c0000
[=] [unistd.py:267] read(3, 0x774c0000, 0x1000) = 4096
[=] [mman.py:200] mmap(0x0, 0x14000, 0x0, 0x802, -1, 0) = 0x774c1000
[=] [mman.py:200] mmap(0x774c1000, 0x2e04, 0x5, 0x12, 3, 0) = 0x774c1000
[=] [mman.py:200] mmap(0x774d4000, 0x13c, 0x3, 0x12, 3, 12288) = 0x774d4000
[=] [unistd.py:234] close(3) = 0
[=] [mman.py:42] munmap(0x774c0000, 0x1000) = 0
[=] [fcntl.py:44] open('/lib/libc.so.0', 0x0, 0o0) = 3
[=] [stat.py:322] fstat(3, 0x7ff3c120) = 0
[=] [mman.py:200] mmap(0x0, 0x1000, 0x3, 0x802, -1, 0) = 0x774d5000
[=] [unistd.py:267] read(3, 0x774d5000, 0x1000) = 4096
[=] [mman.py:200] mmap(0x0, 0x9f000, 0x0, 0x802, -1, 0) = 0x774d6000
[=] [mman.py:200] mmap(0x774d6000, 0x88728, 0x5, 0x12, 3, 0) = 0x774d6000
[=] [mman.py:200] mmap(0x7756f000, 0xde0, 0x3, 0x12, 3, 561152) = 0x7756f000
[=] [mman.py:200] mmap(0x77570000, 0x4af0, 0x3, 0x812, -1, 0) = 0x77570000
[=] [unistd.py:234] close(3) = 0
[=] [mman.py:42] munmap(0x774d5000, 0x1000) = 0
[=] [fcntl.py:44] open('/lib/libc.so.0', 0x0, 0o0) = 3
[=] [stat.py:322] fstat(3, 0x7ff3c110) = 0
[=] [unistd.py:234] close(3) = 0
[=] [stat.py:456] stat('/lib/ld-uClibc.so.0', 0x7ff3cb7c) = 0
[=] [ioctl.py:105] ioctl(0x0, 0x540d, 0x7ff3ca68) = -1
[=] [ioctl.py:105] ioctl(0x1, 0x540d, 0x7ff3ca68) = -1
[=] [unistd.py:286] write(1, 0x77571dc8, 209) = 0
Content-type: text/html; Charset=UTF-8
Pragma: no-cache
Cache-Control: no-cache
Expires: -1

<html>
<head>
    <meta http-equiv="Refresh" content="0; url=/login.html?ErrorCode=1">
</head>
<body>
</body>
</html>
[=] [unistd.py:20]     exit(1) = 1
unleashed@inferno ~airties5630> ! dev x + $
```

qltool is in tool-chain in Qiling Binary Framework



TAS Example: CVE-2015-2797

- CVE-2015-2797 Metasploit Module snippet

```
def execute_command(cmd, opts)
    shellcode = prepare_shellcode(cmd)
    begin
        res = send_request_cgi({
            'method' => 'POST',
            'uri'      => '/cgi-bin/login',
            'encode_params' => false,
            'vars_post' => {
                'redirect' => shellcode,
                'user'      => rand_text_alpha(5),
                'password' => rand_text_alpha(8)
            }
        })
        return res
    rescue ::Rex::ConnectionError
        fail_with(Failure::Unreachable, "#{peer} - Failed to connect to the web server")
    end
end
end
```



TAS Example: CVE-2015-2797

cgi-bin/login binary analysis with Cutter

```
[0x00400c68]
addiu    v0, fp, 0x50
move     a0, v0
move     a1, zero
addiu    a2, zero, 8
lw       v0, -sym.imp.memset(gp)
move     t9, v0
jalr    t9
nop
lw       gp, (var_20h)
addiu    v0, fp, 0x58
move     a0, v0
move     a1, zero
addiu    a2, zero, 8
lw       v0, -sym.imp.memset(gp)
move     t9, v0
jalr    t9
nop
lw       gp, (var_20h)
lw       v0, -segment.LOAD0(gp)
addiu    a0, v0, 0x2810
lw       v0, -sym.imp.getenv(gp)
move     t9, v0
jalr    t9
nop
lw       gp, (var_20h)
sw       v0, (var_38h)
lw       v0, (var_38h)
bnez   v0, 0x400ce8
nop
```

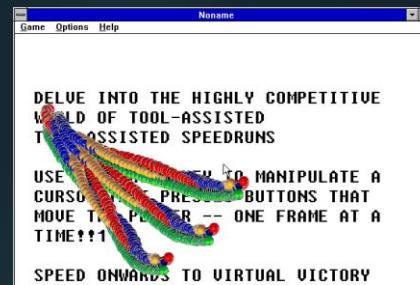
; envp
; 0x412bb8

; envp
; 0x412bb8

; 0x412b64
; 0x402810 ; "REMOTE_ADDR" ; argc ; str.RE...

; 0x412c10

1. lw v0, -sym.imp.getenv(gp)



TAS Example: CVE-2015-2797

Writing our custom emulation script

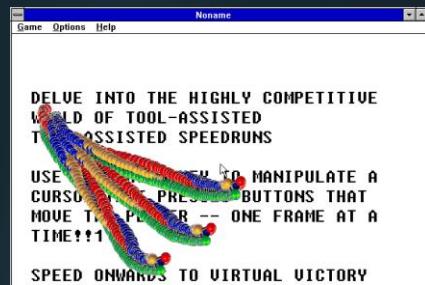
```
env_vars = {  
    "REQUEST_METHOD": "GET",  
    "REQUEST_URI": "/cgi-bin/login",  
    "CONTENT-TYPE": "application/x-www-form-urlencoded",  
    "REMOTE_ADDR": "0.0.0.0",  
    "REMOTE_PORT": "80",  
    "QUERY_STRING": "user=echelon&password=ILOVEQILING&redirect=" + "vulnerabilityhere'  
}
```



TAS Example: CVE-2015-2797

Writing our custom emulation script

```
1. def main(input_content: bytes, enable_trace: bool, hook: bool):
2.     env_vars = {
3.         "REQUEST_METHOD": "GET",
4.         "REQUEST_URI": "/cgi-bin/login",
5.         "CONTENT-TYPE": "application/x-www-form-urlencoded",
6.         "REMOTE_ADDR": "0.0.0.0",
7.         "REMOTE_PORT": "80",
8.         "QUERY_STRING": "user=echelon&password=ILOVEQILING&redirect=" + "vulnerab:
9.     }
10.    ql = Qiling(
11.        ["/home/qiling_projects/Airties/5650v3TT/squashfs-root/webs/cgi-bin/login",
12.         rootfs="/home/qiling_projects/Airties/5650v3TT/squashfs-root/",
13.         output="default",
14.         env=env_vars,
15.         console=True if enable_trace else False,
16.     )
```



TAS Example: CVE-2015-2797

Writing our custom emulation script

```
.. def hook_callback(ql, address, size):
..     global last_register
..     global only_main
..     # read current instruction bytes
..     data = ql.mem.read(address, size)
..     # initialize Capstone
..     md = Cs(CS_ARCH_MIPS, CS_MODE_MIPS32 + CS_MODE_BIG_ENDIAN)
..     # disassemble current instruction
..     for i in md.disasm(data, address):
..         addr = f"0x{i.address:08x}"
..         if only_main:
..             if int(addr,16) > 0x00400b80 and int(addr,16) < 0x004012ac:
..                 last_register = [hex(ql.reg.arch_pc), hex(ql.reg.arch_sp), hex(ql
..                 print("[*] 0x{:08x}: {} {}".format(i.address, i.mnemonic, i.op_st
..             else:
..                 last_register = [hex(ql.reg.arch_pc), hex(ql.reg.arch_sp), hex(ql
..                 print("[*] 0x{:08x}: {} {}".format(i.address, i.mnemonic, i.op_st
```



TAS Example: CVE-2015-2797

Writing our custom emulation script

```
[*] 0x004011f4: move $a0, $v0
[*] 0x004011f8: lw $v0, -0x7fdc($gp)
[*] 0x004011fc: addiu $v0, $v0, 0x12b4
[*] 0x00401200: move $t9, $v0
[*] 0x00401204: jalr $t9
[*] 0x00401208: nop
[*] 0x0040120c: lw $gp, 0x20($fp)
[*] 0x00401210: b 0x401264
[*] 0x00401214: nop
[*] 0x00401264: lw $v0, -0x7f6c($gp)
[*] 0x00401268: move $t9, $v0
[*] 0x0040126c: jalr $t9
[*] 0x00401270: nop
[*] 0x00401274: lw $gp, 0x20($fp)
[*] 0x00401278: lw $v0, -0x7f24($gp)
[*] 0x0040127c: lw $v0, ($v0)
[*] 0x00401280: move $a0, $v0
[*] 0x00401284: lw $v0, -0x7f4c($gp)
[*] 0x00401288: move $t9, $v0
[*] 0x0040128c: jalr $t9
[*] 0x00401290: nop
Content-type: text/html; Charset=UTF-8
Pragma: no-cache
Cache-Control: no-cache
Expires: -1

<html>
<head>
    <meta http-equiv="Refresh" content="0; url=/login.html?ErrorCode=4&redirect=vulnhere">
</head>
<body>
</body>
</html>
[*] 0x00401294: lw $gp, 0x20($fp)
[*] 0x00401298: lw $v0, 0x28($fp)
[*] 0x0040129c: move $sp, $fp
[*] 0x004012a0: lw $ra, 0x1f4($sp)
[*] 0x004012a4: lw $fp, 0x1f0($sp)
[*] 0x004012a8: addiu $sp, $sp, 0x1f8
```



TAS Example: CVE-2015-2797

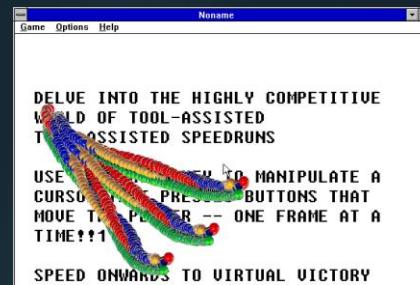
Writing our custom emulation script

```
try:
    ql.run()
    # If I understood correctly,
    # unicorn will throw a signal to inform us something bad happened
except:
    print("[*] It is crashed")
    print("[*] Last Registers:")
    print(f"[*] $pc : {hex(ql.reg.arch_pc)}")
    # $sp == stack pointer
    print(f"[*] $sp : {hex(ql.reg.arch_sp)}")
    print("[*] $sp:data ")
    # Print it out what is in the address
    print(ql.mem.read(ql.reg.arch_sp,16))
    sys.exit()
```



TAS Example: CVE-2015-2797

Writing our custom emulation script



TAS Example: CVE-2015-2797

The big question: Is fuzzing possible?

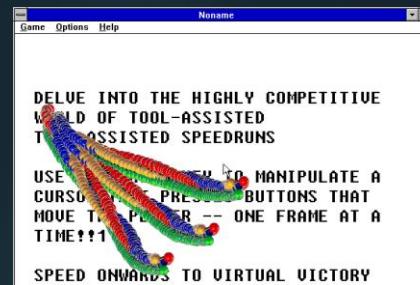
```
american fuzzy lop ++3.01a (default) [fast] {0}
process timing
  run time : 0 days, 0 hrs, 14 min, 28 sec
  last new path : 0 days, 0 hrs, 4 min, 14 sec
  last uniq crash : 0 days, 0 hrs, 4 min, 22 sec
  last uniq hang : none seen yet
cycle progress
  now processing : 2.1 (66.7%)
  paths timed out : 1 (33.33%)
stage progress
  now trying : trim 8/8
  stage execs : 457/520 (87.88%)
  total execs : 107k
  exec speed : 163.5/sec
fuzzing strategy yields
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
  arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
  havoc/splice : 11/106k, 0/0
  py/custom : 0/0, 0/0
  trim : 0.00%/968, n/a
overall results
  cycles done : 98
  total paths : 3
  uniq crashes : 9
  uniq hangs : 0
map coverage
  map density : 0.69% / 1.26%
  count coverage : 1.32 bits/tuple
findings in depth
  favored paths : 2 (66.67%)
  new edges on : 3 (100.00%)
  total crashes : 304 (9 unique)
  total timeouts : 565 (10 unique)
path geometry
  levels : 2
  pending : 2
  pend fav : 1
  own finds : 2
  imported : 0
  stability : 100.00%
[cpu000: 75%]
^C
```



TAS Example 2: Emulating aescrypt2

Huawei HG8245X6 Optical Network Terminal





TAS Example 2: Emulating aescrypt2

Huawei HG8245X6 ONT's configuration XML files are stored encrypted.

```
$ grep -R aescrypt2
.
.
.
bin/restorehwmode.sh:      $var_pack_temp_dir/aesCrypt2 0 $2 $2"_tmp"
bin/restorehwmode.sh:      $var_pack_temp_dir/aesCrypt2 1 $1 $1"_tmp"
.
.
```

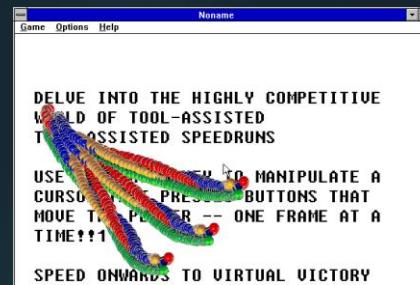


TAS Example 2: Emulating aescrypt2

Emulating aescrypt2 with some adjustments

```
def prepare() -> Qiling:
    #
    # decrypt var_ctree
    # var_pack_temp_dir=/bin/
    # var_default_ctree=/mnt/jffs2/customize_xml/hw_default_ctree.xml
    # var_temp_ctree=/mnt/jffs2/customize_xml/hw_default_ctree_temp.xml
    # $var_pack_temp_dir/aescrypt2 1 $var_default_ctree $var_temp_ctree

    cur_dir = os.path.abspath(".")
    ql = Qiling(
        [
            cur_dir + "/rootfs/bin/aescrypt2",
            "1", # decrypt flag for encryption set to 0
            "/mnt/jffs2/hw_ctree.xml", # encrypted file
            # i couldnt understand what second arg's purpose because
            # aescrypt2 renames this field to original file
            # it is like a dummy file imo.
            "/tmp/hw_ctree.gz",
        ],
        cur_dir + "/rootfs/",
        verbose=QL_VERBOSE.DEBUG,
        console=True,
    )
    return ql
```



TAS Example 2: Emulating aescrypt2

Emulating aescrypt2 with some adjustments

```
5\x9dX\x93\x15N$\xa2\x96\xa2q\x90\x80~u"\xbcy\xf4\x17`\xab\x9f.\x93\x05\x03\x00')
[+] bytearray(b'')
[+] 0x04814400: writev(fd = 0x6, vec = 0x7ff3be5c, vien = 0x2) = 0x19d
[+] [+] Received Interrupt: 2 Hooked Interrupt: 2
[+] 0x048140cc: close(fd = 0x6) = 0x0
[+] [+] Received Interrupt: 2 Hooked Interrupt: 2
[+] unlink(/mnt/jffs2/hw_ctree.xml) = 0
[+] 0x04818364: unlink(pathname = 0x7ff3cb74) = 0x0
[+] [+] Received Interrupt: 2 Hooked Interrupt: 2
[+] rename() path: /tmp/hw_ctree.gz → /mnt/jffs2/hw_ctree.xml
[+] 0x0481838c: rename(oldname_buf = 0x7ff3cbf4, newname_buf = 0x7ff3cb74) = 0x0
[+] [+] Received Interrupt: 2 Hooked Interrupt: 2
[+] 0x047d804c: exit_group(exit_code = 0x0)
```



TAS Example 2: Emulating aescrypt2

Emulating aescrypt2 with some adjustments



TAS Example 3: msfvenom payload

Instrumented Dynamic Analysis of a msfvenom payload

```
def prepare() -> Qiling:

    shellcode = open("./bin.sc", "rb").read()
    rootfs = "/tools/qiling/examples/rootfs/x86_windows"
    ostype = "windows"
    archetype = "x86"
    output = "default"

    ql = Qiling(
        shellcoder=shellcode,
        rootfs=rootfs,
        ostype=ostype,
        archetype=archetype,
        output=output,
        console=True,
    )

    ql.filter = ["connect"]
    ql.set_api("connect", _connect, QL_INTERCEPT.ENTER)
    return ql
```



TAS Example 3: msfvenom payload

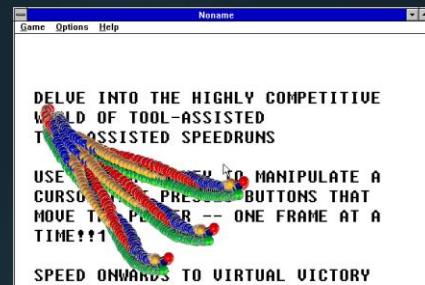
Instrumented Dynamic Analysis of a msfvenom payload

```
def human_readable_ip_str(name):
    ipp_addr_version = hexlify(name[0:2])
    ip_addr_port = int(hexlify(name[2:4]), 16)
    ip_addr = hexlify(name[4:8])

    ip = ip_builder(ip_addr)

    print(f"Trying to connect ~> {ip}:{ip_addr_port}")

def _connect(ql, address, params):
    value = ql.mem.read(params["name"], params["namelen"])
    human_readable_ip_str(value)
```



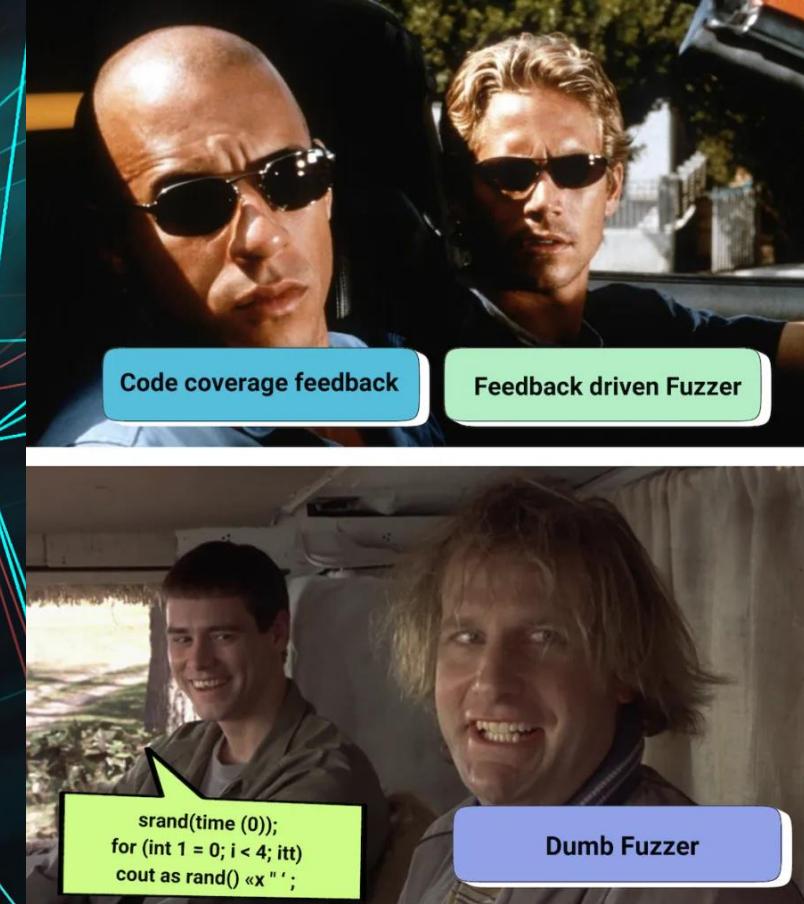
TAS Example 3: msfvenom payload

Instrumented Dynamic Analysis of a msfvenom payload

```
unleashed@inferno ➤ qiling ➤ $ python emu.py
[+] Initiate stack address at 0xffffdd000
[+] TEB addr is 0x6000
[+] PEB addr is 0x6044
[+] Loading /tools/qiling/examples/rootfs/x86_windows/Windows/SysWOW64/ntdll.dll to 0x10000000
[+] Done with loading /tools/qiling/examples/rootfs/x86_windows/Windows/SysWOW64/ntdll.dll
[+] Loading /tools/qiling/examples/rootfs/x86_windows/Windows/SysWOW64/kernel32.dll to 0x101750
00
[+] Done with loading /tools/qiling/examples/rootfs/x86_windows/Windows/SysWOW64/kernel32.dll
[+] Loading /tools/qiling/examples/rootfs/x86_windows/Windows/SysWOW64/user32.dll to 0x10280000
[+] Done with loading /tools/qiling/examples/rootfs/x86_windows/Windows/SysWOW64/user32.dll
Trying to connect ~> 192.168.5.10:53
unleashed@inferno ➤ qiling ➤ $
```

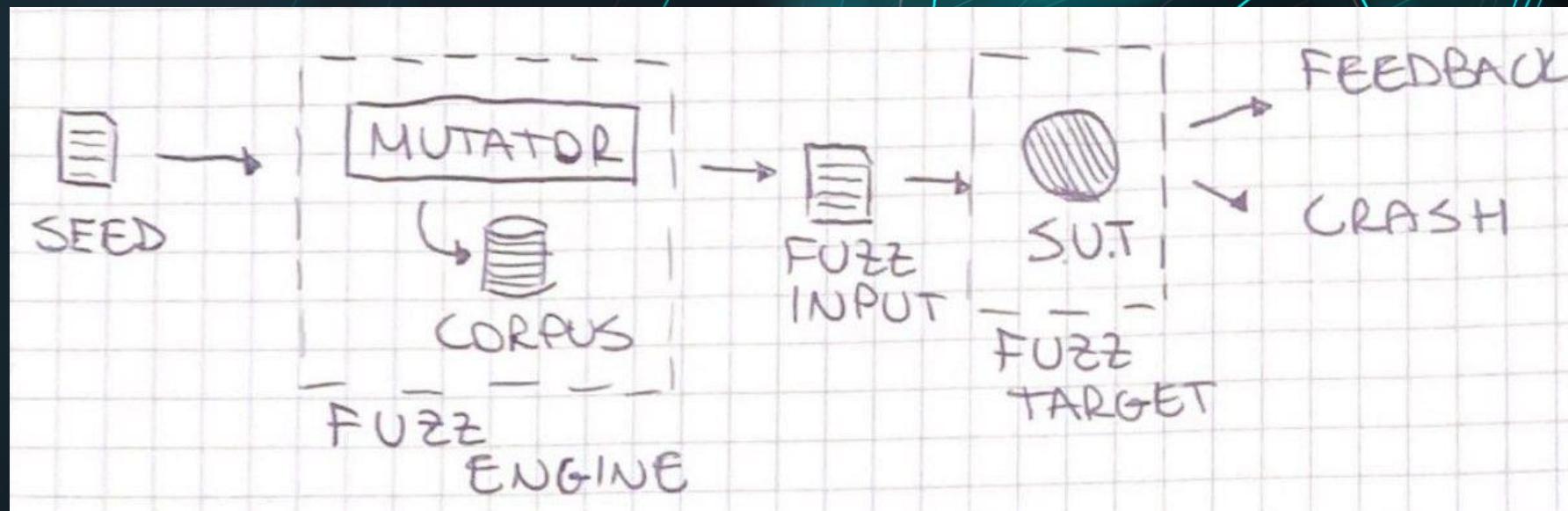


Fuzzing





Fuzzing





Fuzzing Example:



BrainFlow is a library intended to obtain, parse and analyze EEG, EMG, ECG, and other kinds of data from biosensors.

```
int main (int argc, char *argv[])
{
    int res = 0;
    try
    {
        // traditional file input
        BrainFlowArray<double, 2> data;
        BrainFlowArray<double, 2> restored_data = DataFilter::read_file(argv[1])
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
    }

    return res;
}
```

Transfer into this version

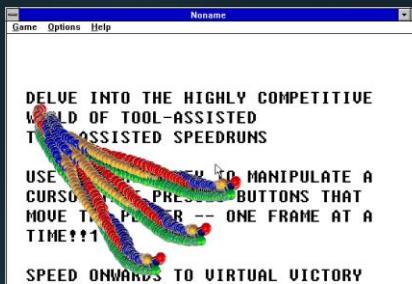
```
int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int res = 0;
    int board_id = (int)BoardIds::SYNTHETIC_BOARD;
    // use synthetic board for demo
    BoardShim *board = new BoardShim (board_id, params);

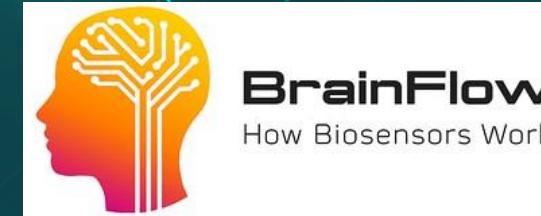
    try
    {
        ...
        DataFilter::write_file (data, "test.csv", "w");
        BrainFlowArray<double, 2> restored_data = DataFilter::read_file ("test.csv");
        std::cout << "Restored data:" << std::endl << restored_data << std::endl;
    }
    catch (const BrainFlowException &err)
    {
        ...
    }

    delete board;

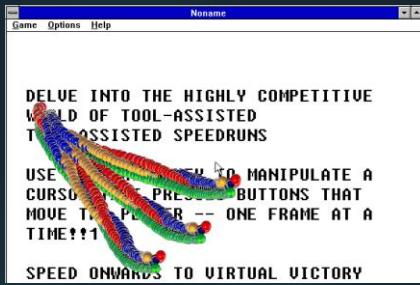
    return res;
}
```



Fuzzing Example:



```
american fuzzy lop ++4.02a {fuzzer00} ./harness [fast]
process timing
  run time : 0 days, 0 hrs, 0 min, 13 sec
  last new find : 0 days, 0 hrs, 0 min, 0 sec
  last saved crash : 0 days, 0 hrs, 0 min, 0 sec
  last saved hang : none seen yet
cycle progress
  now processing : 16.0 (48.5%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : MOpt-havoc
  stage execs : 2646/32.8k (8.07%)
  total execs : 44.1k
  exec speed : 3166/sec
fuzzing strategy yields
  bit flips : 0/0, 0/0, 0/0
  byte flips : 0/0, 0/0, 0/0
  arithmetics : 0/0, 0/0, 0/0
  known ints : 0/0, 0/0, 0/0
  dictionary : 0/0, 0/0, 0/0, 0/0
  havoc/splice : 41/35.8k, 5/5376
  py/custom/rq : unused, unused, unused, unused
  trim/eff : disabled, n/a
overall results
  cycles done : 0
  corpus count : 33
  saved crashes : 21
  saved hangs : 0
map coverage
  map density : 0.24% / 0.32%
  count coverage : 3.68 bits/tuple
findings in depth
  favored items : 6 (18.18%)
  new edges on : 10 (30.30%)
  total crashes : 35.0k (21 saved)
  total tmouts : 0 (0 saved)
item geometry
  levels : 3
  pending : 33
  pend fav : 6
  own finds : 32
  imported : 0
  stability : 100.00%
[cpu000: 33%]
```



Fuzzing Example:



```

—0 "disasm, code"
[ DISASM ]
0x7ffff789c89e <free+30>    mov    rax, qword ptr [rdi - 8]
0x7ffff789c8a2 <free+34>    mov    ebp, dword ptr fs:[rbx]
0x7ffff789c8a5 <free+37>    test   al, 2
0x7ffff789c8a7 <free+39>    jne    free+128           <free+128>
↓
0x7ffff789c900 <free+128>   mov    edx, dword ptr [rip + 0x1609e2]
0x7ffff789c906 <free+134>   test   edx, edx
0x7ffff789c908 <free+136>   jne    free+178           <free+178>
↓
0x7ffff789c932 <free+178>   mov    rdi, rsi
0x7ffff789c935 <free+181>   call   0xfffff7898340       <0x7ffff7898340>

0x7ffff789c93a <free+186>   mov    dword ptr fs:[rbx], ebp
0x7ffff789c93d <free+189>   add    rsp, 0x18
0x7ffff789c941 <free+193>   pop    rbx
0x7ffff789c942 <free+194>   pop    rbp
0x7ffff789c943 <free+195>   ret
0x7ffff789c944 <free+196>   nop
0x7ffff789c948 <free+200>   ret
0x7ffff789c949 <free+201>   nop
0x7ffff789c950 <free+208>   mov    rdx, qword ptr [rip + 0x1603a9]
0x7ffff789c957 <free+215>   mov    qword ptr [rsp], rdi
0x7ffff789c95b <free+219>   cmp    byte ptr fs:[rdx], 0
0x7ffff789c95f <free+223>   jne    free+59            <free+59>
0x7ffff789c965 <free+229>   mov    qword ptr [rsp + 8], rsi
0x7ffff789c96a <free+234>   call   0xfffff789bbb0       <0x7ffff789bbb0>
0x7ffff789c96f <free+239>   mov    rdi, qword ptr [rsp]
0x7ffff789c973 <free+243>   mov    rsi, qword ptr [rsp + 8]
0x7ffff789c978 <free+248>   mov    rax, qword ptr [rdi - 8]
0x7ffff789c97c <free+252>   jmp    free+59            <free+59>
0x7ffff789c981
0x7ffff789c98c
0x7ffff789c990
0x7ffff789c994    nop
0x7ffff789c991    word ptr cs:[rax + rax]
0x7ffff789c992    nop
0x7ffff789c993    dword ptr [rax]
endbr64
0x7ffff789c994    mov    rax, qword ptr [rip + 0x16035d]

—4 "Main"
pwndbg: loaded 199 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from ./harness...
pwndbg: tip of the day: GDB's 'apropos <topic>' command displays all registered commands that are related to the given <topic>
pwndbg> r /home/depo/projects/brainflow/build/sync_dir/fuzzer00/crashes/id:000009,sig:11,src:000000,time:104,execs:220,op:MOpt_havoc,rep:16
Starting program: /home/depo/projects/brainflow/build/harness /home/depo/projects/brainflow/build/sync_dir/fuzzer00/crashes/id:000009,sig:11,src:000000,time:104,execs:220,op:MOpt_havoc,rep:16
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff789c89e in free () from /usr/lib/libc.so.6
pwndbg>
```

```

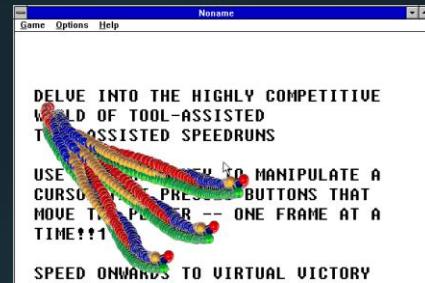
—1 "legend, regs"
RAX 0x62b2
RBX 0xfffffff7fffffe0
RCX 0x7ffff75ff010 ← 0x1000100010100
RDX 0x555000fd3901
RDI 0x06a9a9769a91106
RSI 0x06a9a9769a910f6
R8 0x555555a88030 ← 0x555000fd39c8
R9 0x0
R10 0x0
R11 0xfc2789552c386983
R12 0x555555a8d258 (_afl_area_ptr) → 0x7ffff75ff010 ← 0x1000100010100
R13 0x7ffff7ffffd240 ← 0x150
R14 0x555555a8ce0 ← 0x5fbad2488
R15 0x7ffff7ffffd250 ← 0x303030302e323132 ('212.0000')
RBP 0x555555a8db30 ← 0x34303637342e392d ('-9.47604')
RSP 0x7ffff7fffffc90 → 0x555555a8db30 (_afl_area_ptr) → 0x7ffff75ff010 ← 0x1000100010100
RIP 0x7ffff789c89e (free+30) ← mov rax, qword ptr [rdi - 8]

—2 "stack"
000100010100
01:0008| 0x7ffff7fffffc98 ← 0x3e0
02:0010| 0x7ffff7fffffcfa0 ← 0x14b
03:0018| 0x7ffff7fffffcfa8 → 0x555555a8d740 ← '212.00000'
04:0020| 0x7ffff7fffffcfb0 → 0x555555a8db30 ← 0x34303637342e392d ('-9.47604')
05:0028| 0x7ffff7fffffcfb8 → 0x55555575e488 (read_file+2280) ← add rbx, 0x10
06:0030| 0x7ffff7fffffcfc0 ← 0xb00000009 /* '\t' */
07:0038| 0x7ffff7fffffcfc8 ← 0xffffffff000000009 /* '\t' */

—3 "Input / Output"
/dev/pts/8
warning: GDB: Failed to set controlling terminal: Operation not permitted

—6 "inferno"
Python 3.10.5 (main, Jun  6 2022, 18:49:26) [GCC 12.1.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```



Fuzzing Example: ELF Parsers

Custom mutators help researchers to reach out more edges and code coverage. If your target is expecting more structured input, randomly mutated inputs will not help you to go through into code base.

- <https://tmpout.sh/3/09.html>
- <https://github.com/echel0nn/golfuzz>

- Scope -

I tested eight projects which are;

- 1) [] binutils/readelf
- 2) [x] elfmaster/libelfmaster
- 3) [x] radareorg/radare2 (CVE-2023-1605) [1]
- 4) [x] jacob-baines/elfparser
- 5) [x] finixbit/elf-parser
- 6) [] eliben/pyelftools (unhandled EI_VERSION)
- 7) [o] lief-project/LIEF
- 8) [] cea-sec/miasm

- [] no crash
- [x] crash
- [o] Infinite Loop

```
size_t afl_custom_fuzz(my_mutator_t *data, uint8_t *in_buf, size_t buf_size,
                       u8 **out_buf, uint8_t *add_buf,
                       size_t add_buf_size, // add_buf can be NULL
                       size_t max_size) {

    RawBinary elf_obj;
    RawBinary *elf = &elf_obj;
    elf->isa = 62;
    Elf64_Ehdr *ehdr;
    Elf64_Phdr *phdr;
    copy_text_segment(elf, buf, sizeof(buf));
    ehdr = populate_ehdr(elf);
    phdr = populate_phdr(elf);
    set_entry_point(elf);

    size_t mutated_size = ehdr_size + phdr_size + elf->text.text_size;

    // let's put the mutated bytes into dead bytes.

    ehdr->e_ident[EI_CLASS] = (uint8_t *) (in_buf + pos);
    pos = pos + 1;
    ehdr->e_ident[EI_DATA] = (uint8_t *) (in_buf + pos);
    pos = pos + 1;
    ehdr->e_ident[EI_VERSION] = (uint8_t *) (in_buf + pos);
    pos = pos + 1;
    ehdr->e_ident[EI_OSABI] = (uint8_t *) (in_buf + pos);
    pos = pos + 1;
    for (int i = 0x8; i < 0x10; ++i) {
        (ehdr->e_ident)[i] = (uint8_t *) (in_buf + pos);
        pos = pos + 1;
    }

    ehdr->e_version = (uint32_t *) (in_buf + pos);
    pos = pos + 4;
    // sections headers
    ehdr->e_shoff = (uint64_t *) (in_buf + pos);
    pos = pos + 8;
    ehdr->e_shentsize = (uint16_t *) (in_buf + pos);
    pos = pos + 2;
    ehdr->e_shnum = (uint16_t *) (in_buf + pos);
    pos = pos + 2;
    ehdr->e_shstrndx = (uint16_t *) (in_buf + pos);
    pos = pos + 2;
    ehdr->e_flags = (uint32_t *) (in_buf + pos);
    pos = pos + 4;
    // physical addr
    phdr->p_paddr = (uint64_t *) (in_buf + pos);
    pos = pos + 8;
    phdr->p_align = (uint64_t *) (in_buf + pos);
    pos = pos + 8.
```

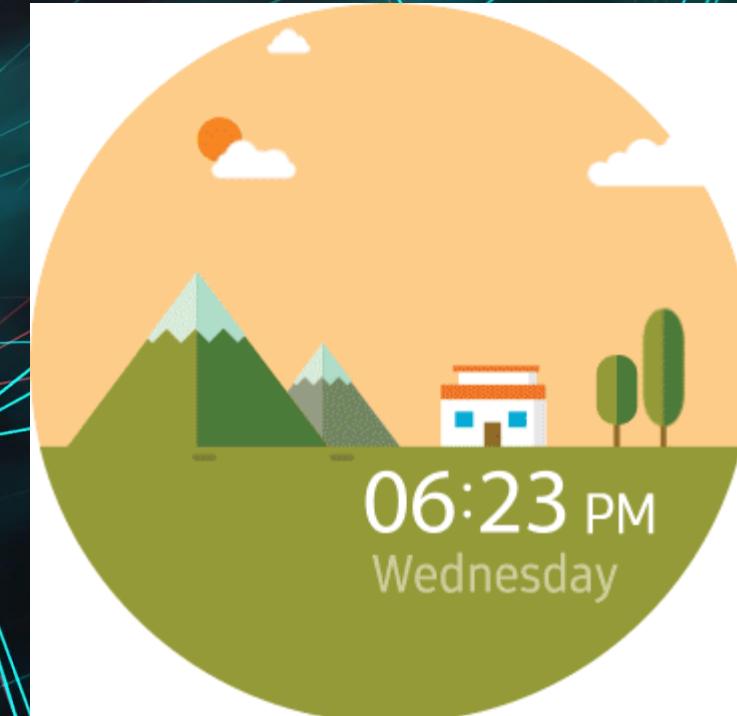


Fuzzing Example: Telegram



This example is a vector based animation renderer library, which is also used in Telegram application.

Structure-Aware Fuzzing is way-to-go strategy choice to fuzz big projects.





Fuzzing Example: Telegram



Finding the best suitable function

```
1. auto animation = rlottie::Animation::loadFromData(std::string(rawData), std::string(cacheKey));
```

or

```
1. auto animation = rlottie::Animation::loadFromFile("absolute_path/test.json"); // giving the input traditionally as argv[1]
```

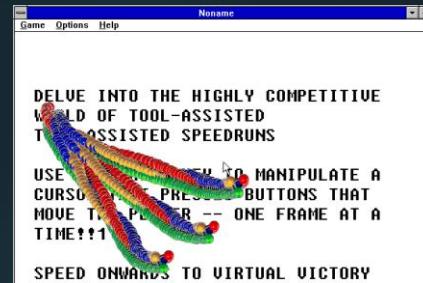


Fuzzing Example: Telegram



Acceptable variables by rlottie library

```
1. https://github.com/landn172/lottie-miniapp/blob/master/lottie-json.md
2. var data = {
3.   v: '4.6.3', // version
4.   fr: 30, // 30 fps
5.   ip: 0,
6.   op: 73,
7.   w: 250, // width
8.   h: 275, // height
9.   nm: 'B',
10.  ddd: 0,
11.  assets: [],
12.  layers: []
13. };
14. var assets = ..
15. var layers = ..
```



Fuzzing Example: Telegram



What is Protobuf? (Protocol buffers)

```
message Person {  
    optional string name = 1;  
    optional int32 id = 2;  
    optional string email = 3;  
}
```

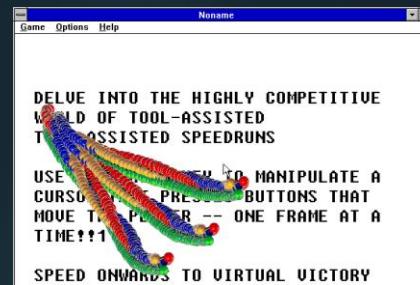
A proto definition.

```
// Java code  
Person john = Person.newBuilder()  
    .setId(1234)  
    .setName("John Doe")  
    .setEmail("jdoe@example.com")  
    .build();  
output = new FileOutputStream(args[0]);  
john.writeTo(output);
```

Using a generated class to persist data.

```
// C++ code  
Person john;  
fstream input(argv[1],  
             ios::in | ios::binary);  
john.ParseFromIstream(&input);  
id = john.id();  
name = john.name();  
email = john.email();
```

Using a generated class to parse persisted data.



Fuzzing Example: Telegram



Let's get lazy and ask ChatGPT to convert example JSON into protobuf message!

You

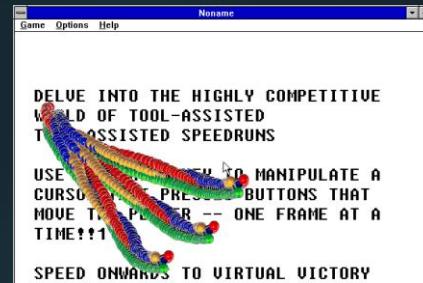
```
var data = {  
    v: '4.6.3', // version  
    fr: 30, // 30 fps  
    ip: 0,  
    op: 73,  
    w: 250, // width  
    h: 275, // height  
    nm: 'B',  
    ddd: 0,  
    assets: [], // 附 assets  
    layers: [] // 层 , 附layers  
};  
  
var assets = {  
    id: 'comp_5',  
    layer: [  
        {  
            ddd: 0,  
            ind: 1,  
            ty: 4,  
            nm: 'Shape Layer 1',  
        }  
    ]  
};
```

Can you convert this JSON data into protobuf message?

ChatGPT

```
To represent the provided JSON data in Protocol Buffers (protobuf) format, you would need to define appropriate protobuf messages for both the outer `data` structure and the inner `assets` and `layers` structures. Here's an example of how you might define these messages:
```

```
syntax = "proto3";  
  
message Data {  
    string v = 1; // version  
    int32 fr = 2; // 30 fps  
    int32 ip = 3;  
    int32 op = 4;  
    int32 w = 5;  
    int32 h = 6;  
    string nm = 7;  
    int32 ddd = 8;  
    repeated string assets = 9;  
    repeated Layer layers = 10;  
}  
  
message Asset {  
    string id = 1;  
    repeated Layer layer = 2;  
}  
  
message Layer {  
    int32 ddd = 1;  
    int32 ind = 2;  
    int32 ty = 3;  
    string nm = 4;  
}
```



Fuzzing Example: Telegram



Using aflplusplus-libprotobuf-mutator (developer: plummer @ github)

```
DEFINE_AFL_PROTO_FUZZER(const LottieMessage &composition,  
                        unsigned char **out_buf) {  
  
    Json::Value json_composition = lottieMessageToJSON(composition); // trimmed..  
    Json::StreamWriterBuilder writer;  
    std::string json_string = Json::writeString(writer, json_composition);  
    *out_buf = (unsigned char *)json_string.c_str();  
    return json_string.length();  
}
```



Fuzzing Example: Telegram



The example harness that can consume those protobuf mutated inputs

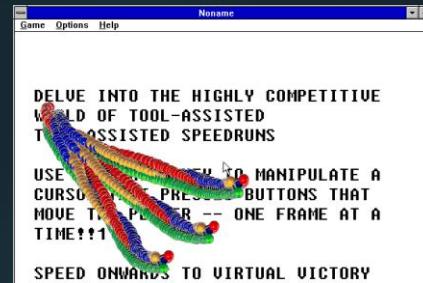
```
while (__AFL_LOOP(10000)) {
    const unsigned char *raw_buf = __AFL FUZZ_TESTCASE_BUF; // it is a JSON string and rlottie expects a json string!
    int len = strlen(reinterpret_cast<const char *>(raw_buf));
    const std::string data(raw_buf, raw_buf + len);

    const std::string CACHE_KEY = "I_DONT_KNOW_WHAT_IS_CACHE_KEY_BUT_WHATEVER";
    std::map<int32_t, int32_t> *colorReplacementMap = new std::map<int32_t, int32_t>();
    auto player = rlottie::Animation::loadFromData(data, CACHE_KEY,
                                                    colorReplacementMap, ".");
    if (player) {
        size_t frame_count = player->totalFrame();

        uint32_t w = 420;
        uint32_t h = 666;

        auto buffer = std::unique_ptr<uint32_t[]>(new uint32_t[w * h]);

        for (size_t frame = 0; frame < frame_count; frame++) {
            rlottie::Surface surface(buffer.get(), w, h, w * 4);
            player->renderSync(frame, surface);
        }
    }
}
```



Fuzzing Example: Telegram



Important Note: You are still dependent on your seeds, the mutator is not going to magically expand itself into more functions!

```
american fuzzy lop ++4.09a {fuzzer01} ./afl_fuzz [explore]
process timing
  run time : 0 days, 7 hrs, 2 min, 0 sec
  last new find : 0 days, 0 hrs, 6 min, 6 sec
  last saved crash : none seen yet
  last saved hang : none seen yet
cycle progress
  now processing : 42.588 (14.0%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : libprotobuf_mutator.so
  stage execs : 288/300 (96.00%)
  total execs : 14.0M
  exec speed : 14.5k/sec
fuzzing strategy yields
  bit flips : disabled (custom-mutator-only mode)
  byte flips : disabled (custom-mutator-only mode)
  arithmetics : disabled (custom-mutator-only mode)
  known ints : disabled (custom-mutator-only mode)
  dictionary : n/a
  havoc/splice : 0/0, 0/0
  py/custom/rq : unused, 7/14.0M, unused, unused
    trim/eff : disabled, disabled
  strategy: explore state: in progress
```

overall results

cycles done : 19.3k
corpus count : 301
saved crashes : 0
saved hangs : 0

map coverage

map density : 1.10% / 1.86%
count coverage : 3.11 bits/tuple

findings in depth

favored items : 53 (17.61%)
new edges on : 53 (17.61%)
total crashes : 0 (0 saved)
total timeouts : 0 (0 saved)

item geometry

levels : 3
pending : 0
pend fav : 0
own finds : 7
imported : 8
stability : 100.00%

[cpu000: 20%]

```
american fuzzy lop ++4.09a {new_seeds} ./afl_fuzz [fast]
process timing
  run time : 0 days, 0 hrs, 0 min, 36 sec
  last new find : 0 days, 0 hrs, 0 min, 31 sec
  last saved crash : 0 days, 0 hrs, 0 min, 36 sec
  last saved hang : none seen yet
cycle progress
  now processing : 12.91 (70.6%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : libprotobuf_mutator.so
  stage execs : 319/690 (46.23%)
  total execs : 554k
  exec speed : 16.0k/sec
fuzzing strategy yields
  bit flips : disabled (custom-mutator-only mode)
  byte flips : disabled (custom-mutator-only mode)
  arithmetics : disabled (custom-mutator-only mode)
  known ints : disabled (custom-mutator-only mode)
  dictionary : n/a
  havoc/splice : 0/0, 0/0
  py/custom/rq : unused, 15/554k, unused, unused
    trim/eff : disabled, disabled
  strategy: explore state: in progress
```

overall results

cycles done : 92
corpus count : 17
saved crashes : 1
saved hangs : 0

map coverage

map density : 0.28% / 1.39%
count coverage : 2.21 bits/tuple

findings in depth

favored items : 10 (58.82%)
new edges on : 11 (64.71%)
total crashes : 6 (1 saved)
total timeouts : 0 (0 saved)

item geometry

levels : 3
pending : 0
pend fav : 0
own finds : 14
imported : 0
stability : 100.00%

[cpu002: 16%]

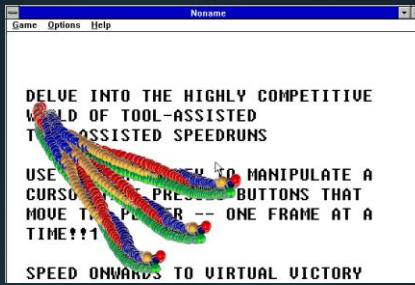
```
$ ls -al corpus/ | wc
188      1498     15454
$ ls -al small_corpus/ | wc
6        42       360
```



What can we expect in future?

- More userspace emulation frameworks for testing
- More CI/CD friendly workspace will be available
- Exploitation helpers (/w simulation engines and AI?)
- Blackbox testing is going to be more like greybox





Resources

- All mentioned research is available @ devilinside.me (ELF is @ tmpout.sh Volume III)
- Harness examples are in “Projects I Would Like to Fuzz(PILF)” repository (<https://github.com/echel0nn/PILF>)



Thank you!

