# functrace

## A quick way to analyze binaries


**Andrea Sindoni**

@invictus1306

WARCON 2019

# > About me

- Senior security researcher
- Specialized in reverse engineering and exploit development
- Low level developer

**Twitter**: @invictus1306

**Disclaimer**: This is my personal research, any views and opinions expressed are my own, not those of any employer

# > Agenda

- Motivation
- Application areas
- Techniques
- Problems
- Binary Instrumentation
- functrace
- Ghidra coverage script
- beebug integration
- Future features

# Motivation

# > Motivation

Analyze programs with no source code:

- It is time consuming

Simplify the reversing process:

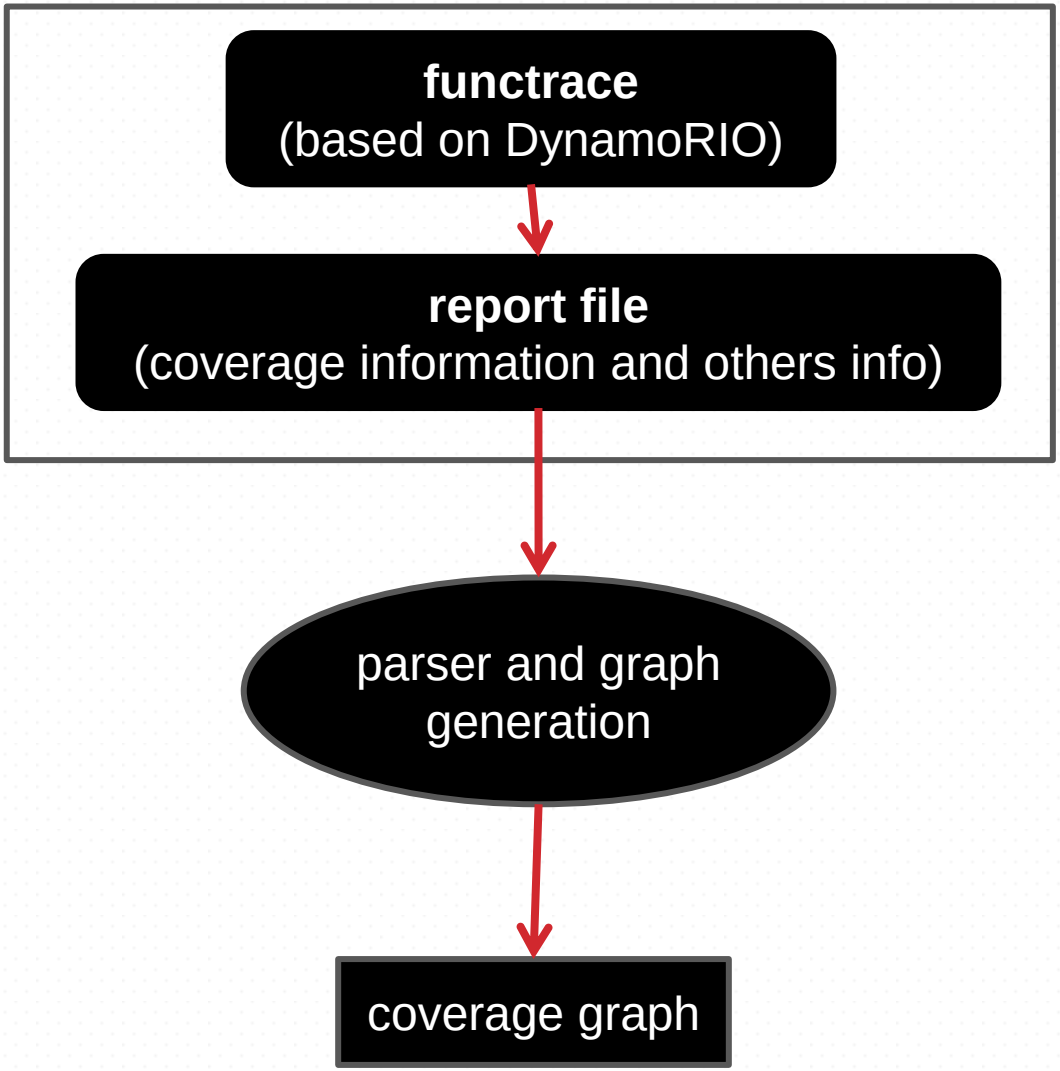
- Function / Basic Blocks tracing graph
- Code coverage
- Makes runtime checks quickly

Existing similar tool:

- https://github.com/gaasedelen/lighthouse

However, the tool presented here is not related to *drcov,* it get function information at runtime and more.

# > Motivation


functrace
(based on DynamoRIO)
report file
(coverage information and others info)
parser and graph generation
coverage graph

# Application areas

# > Application areas

- Vulnerability research
- Exploitable classification
- Fuzzing
- Malware analysis
- Unpacking & de-obfuscation
- Taint-analysis
- Understand the behavior of a program
- Profiling
- …

# Techniques

# > Techniques

How can we quickly monitor a process at runtime?

Different architectures, OS, instructions, executable binaries, etc.

On Linux we can use *ptrace*.

These are some well known software based on *ptrace*:

- strace
- ltrace
- gdb
- …

# > Techniques

## Relevant *ptrace* arguments

- strace (syscall, signal tracer)
  - PTRACE_ATTACH
  - PTRACE_SYSCALL
- ltrace (library call tracer, syscall, signal tracer)
  - PTRACE_ATTACH
  - PTRACE_SYSCALL
  - PTRACE_POKETEXT
  - Locating the PLT of a program
- GDB (it is possible to do a lot of stuff)
  - PTRACE_ATTACH
  - PTRACE_SYSCALL
  - PTRACE_POKETEXT
  - Many others

# > Techniques

## What can we do with the debugger?

- With the debugger we can:
  - Read/Write function arguments and return values
  - Disassemble some portion of code
  - Trace Functions
- Simple to use (e.g., breakpoints, scripting)
- Not totally transparent for the application under test

# > Techniques

e.g. simple tracer script (based on r2pipe)

```
1   import r2pipe
2
3   addr_list = []
4
5   r2 = r2pipe.open("./simple_crash", flags=['-d'])
6
7   r2.cmd('aa')
8
9   func_list = r2.cmdj('aflqj')
10
11  for function in func_list:
12      r2.cmd('db ' + str(function))
13
14  while True:
15      r2.cmd('dc')
16      pdj = r2.cmdj('pdj 1')
17      get_info = r2.cmdj('dij')
18      cur_addr = pdj[0]['offset']
19      addr_list.append(cur_addr)
20      signal = get_info['signal']
21      if signal == "SIGSEGV":
22          pc = r2.cmd('dr rip')
23          print "Crash at " + pc + " - signal: " + signal
24          break
25
```

https://github.com/radare/radare2-r2pipe

# Problems

# > Problems

Problems with debuggers:

- Too limited
- It’s very slow
  - Debugger trap too expensive
  - Variable-Length instruction complications
  - Set statically breakpoints can speedup the process
- No transparency (modify target’s memory)
- Automating the analysis on multi architecture is not a simple task

# Binary instrumentation

# > Binary instrumentation

## Binary Instrumentation

Short description: Injects instrumentation code into a binary to collect run-time information

- **Static** (instrument before runtime)
  - Instrument executable directly by inserting additional assembly instructions (e.g., Dyninst)
- **Dynamic** (instrument at runtime)
  - Injection
  - Just-in-time (JIT) instrumentation

# > Binary instrumentation

## Dynamic Binary Instrumentation (DBI)

**Injection**

- Simple to implement
- Simple architecture
- The control of code executed is less than JIT instrumentation

**JIT instrumentation**

- Original code never executes (e.g., code cache, JIT)
- Good control of executed code at runtime
- Complex architecture
- Different frameworks exist

# > Binary instrumentation


Intel Pin


DynamoRIO

Valgrind


Frida

# > Binary instrumentation

Intel Pin

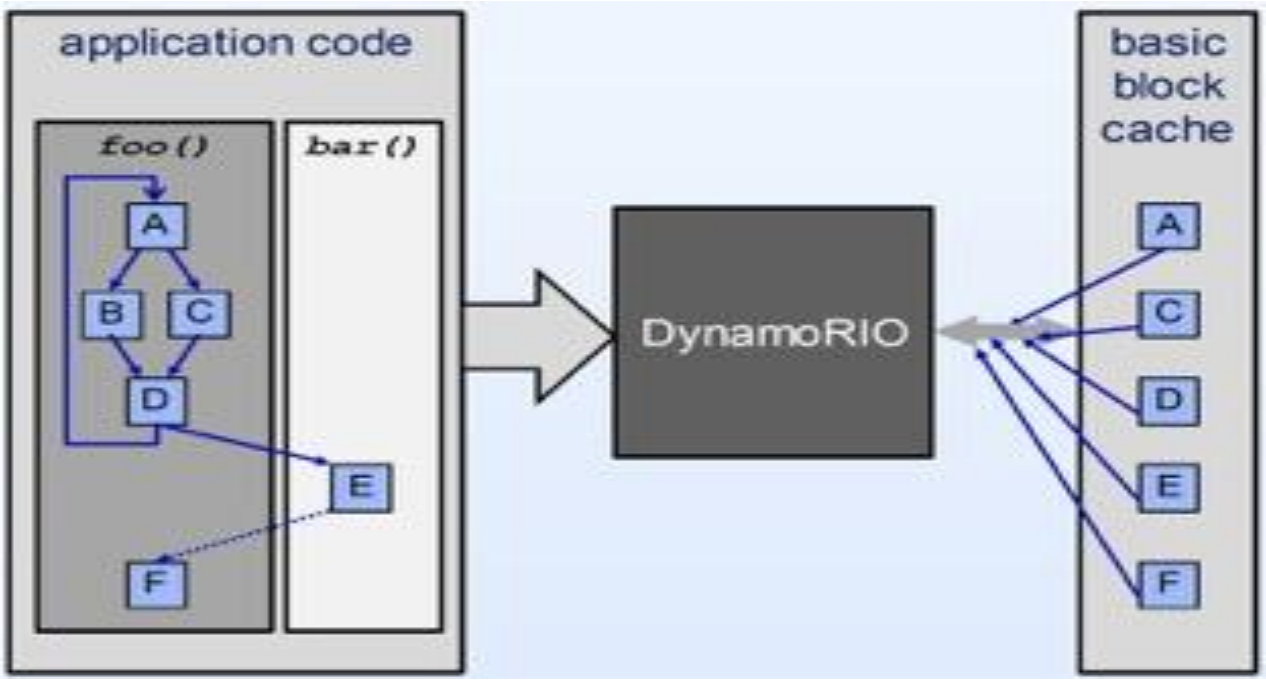
DynamoRIO

Valgrind

Frida

# > Binary instrumentation

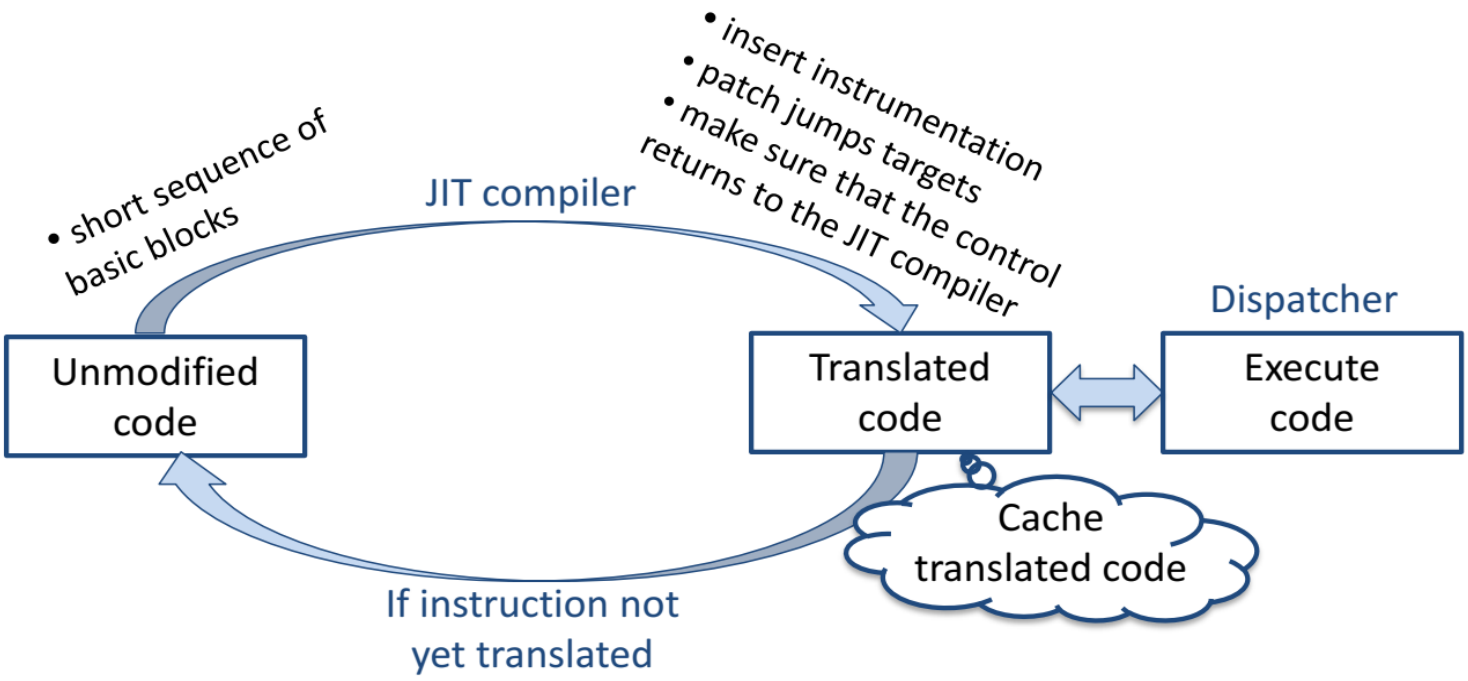
## Dynamic Binary Instrumentation (DBI)




- Copy each basic block into a code cache
- User instrumentation code is added (JIT compiler)

# > Binary instrumentation

## Dynamic Binary Instrumentation (DBI)




- Original code never executed

# functrace

# > functrace

## functrace

*https://github.com/invictus1306/functrace*

These are some implemented features (based on DynamoRIO):

- disassemble all the executed code
- disassemble a specific function
- get arguments of a specific function (dump if these are addresses)
- get return value of a specific function (dump if this is an address)
- monitors application signals
- generate a report file
- Ghidra coverage script (based on the functrace report file)

# > functrace

## functrace - verbose

```
$../../DynamoRIO-Linux-7.0.0-RC1/bin64/drrun -c libfunctrace.so -report_file test0 -verbose -- ../tests/simple_test
Please enter a message:
test
Nothing will happen today! This is the default message
$cat test0
[MOD];simple_test;0x0000000000400000;/home/invictus1306/Documents/article/functrace/tests/simple_test
[IMPORTS]:
Name: puts
Name: strlen
Name: __stack_chk_fail
Name: printf
Name: __libc_start_main
Name: fgets
Name: strcmp
Name: __gmon_start__
Name: memcpy
Name: malloc
[EXPORTS]:
Offset: 0x0000000000200e20 Name: __JCR_LIST__
Offset: 0x00000000000006e0 Name: deregister_tm_clones
Offset: 0x0000000000000720 Name: register_tm_clones
Offset: 0x0000000000000760 Name: __do_global_dtors_aux
Offset: 0x00000000002010b8 Name: completed.7594
Offset: 0x0000000000200e18 Name: __do_global_dtors_aux_fini_array_entry
Offset: 0x0000000000000780 Name: frame_dummy
Offset: 0x0000000000200e10 Name: __frame_dummy_init_array_entry
Offset: 0x0000000000000bf8 Name: __FRAME_END__
Offset: 0x0000000000200e20 Name: __JCR_END__
Offset: 0x0000000000200e18 Name: __init_array_end
Offset: 0x0000000000200e28 Name: _DYNAMIC
Offset: 0x0000000000200e10 Name: __init_array_start
Offset: 0x0000000000000a5c Name: __GNU_EH_FRAME_HDR
Offset: 0x0000000000201000 Name: _GLOBAL_OFFSET_TABLE_
Offset: 0x00000000000009c0 Name: __libc_csu_fini
Offset: 0x0000000000201060 Name: data_start
Offset: 0x00000000002010b0 Name: stdin@@GLIBC_2.2.5
Offset: 0x00000000002010ac Name: _edata
Offset: 0x00000000000009c4 Name: _fini
Offset: 0x0000000000201060 Name: __data_start
```

# > functrace

## disassemble all the executed code

```
$../../DynamoRIO-Linux-7.0.0-RC1/bin64/drrun -c libfunctrace.so -report_file test1 -disassembly -- ../tests/simple_test
Please enter a message:
coverage
Congrats! This is the secret message. Your input is coverage
Very GOOD!
$cat test1
[FUNC];0x00000000004006b0;0x00000000004006da;0x00000000004006b0;_start
TAG  0x00000000004006b0
 +0    L3                    31 ed                xor    ebp, ebp
 +2    L3                    49 89 d1             mov    r9, rdx
 +5    L3                    5e                   pop    rsi
 +6    L3                    48 89 e2             mov    rdx, rsp
 +9    L3                    48 83 e4 f0          and    rsp, 0xf0
 +13   L3                    50                   push   rax
 +14   L3                    54                   push   rsp
 +15   L3                    49 c7 c0 c0 09 40 00 mov    r8, 0x004009c0
 +22   L3                    48 c7 c1 50 09 40 00 mov    rcx, 0x00400950
 +29   L3                    48 c7 c7 51 08 40 00 mov    rdi, 0x00400851
 +36   L3                    e8 77 ff ff ff       call   0x0000000000400650
END 0x00000000004006b0

[NOFUNC];0x0000000000400650;0x0000000000400650;0x0000000000400650;None
TAG  0x0000000000400650
 +0    L3                    ff 25 e2 09 20 00    jmp    <rel> qword ptr [0x0000000000601038]
END 0x0000000000400650

[NOFUNC];0x00000000004006f6;0x00000000004006fb;0x00000000004006f6;None
```

bbstart bbend pc sym

bb disassembly

# > functrace

## get arguments and return value of a specific function

```
$../../DynamoRIO-Linux-7.0.0-RC1/bin64/drrun -c libfunctrace.so -report_file test2 -wrap_function print_secr -wrap_function_args 3 -- ../tests/simple_test
Please enter a message:
coverage
Congrats! This is the secret message. Your input is coverage
Very GOOD!
```

```
[ARG];print_secr;0x00000000004007a6;0;3;0x0000000000601070
[DUMP];0x00000000004007a6;0x54 0x68 0x69 0x73 0x20 0x69 0x73 0x20 0x74 0x68 0x65 0x20 0x73 0x65 0x63 0x72 0x65 0x74 0x20 0x6d 0x65 0x73 0x73 0x61 0x67 0x65
[ARG];print_secr;0x00000000004007a6;1;3;0x00007ffc2d53d7c0
[DUMP];0x00000000004007a6;0x63 0x6f 0x76 0x65 0x72 0x61 0x67 0x65
[ARG];print_secr;0x00000000004007a6;2;3;0x0000000000000002
[NOFUNC];0x0000000000400690;0x0000000000400690;0x0000000000400690;None
[NOFUNC];0x0000000000400696;0x000000000040069b;0x0000000000400696;None
[FUNC];0x00000000004007a6;0x00000000004007fd;0x00000000004007c3;print_secr
[NOFUNC];0x0000000000400680;0x0000000000400680;0x0000000000400680;None
[NOFUNC];0x0000000000400686;0x000000000040068b;0x0000000000400686;None
[FUNC];0x00000000004007a6;0x00000000004007fd;0x00000000004007dd;print_secr
[NOFUNC];0x0000000000400640;0x0000000000400640;0x0000000000400640;None
[NOFUNC];0x0000000000400646;0x000000000040064b;0x0000000000400646;None
[FUNC];0x00000000004007a6;0x00000000004007fd;0x00000000004007f7;print_secr
[FUNC];0x0000000000400851;0x0000000000400942;0x00000000004008e0;main
[RET];print_secr;0x00000000004007a6;0x0000000000603830
[DUMP];0x00000000004007a6;0x47 0x4f 0x4f 0x44
```

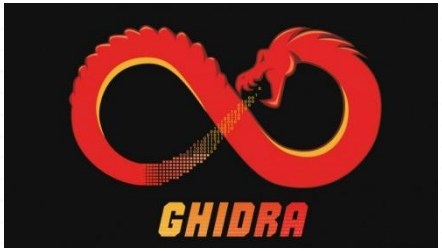
Function argument 1

Function return value

# > functrace

## Ghidra coverage script

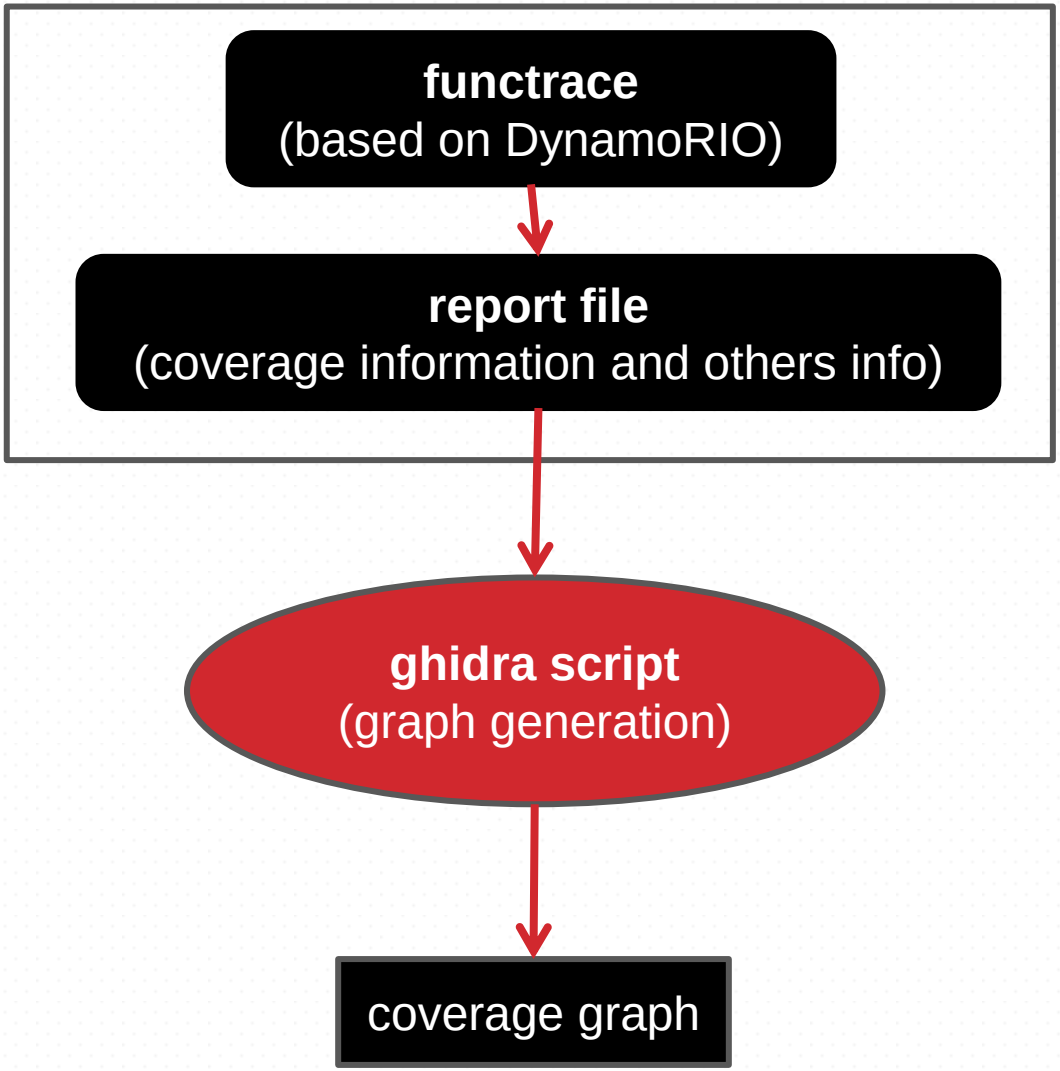
https://ghidra-sre.org/




Script features

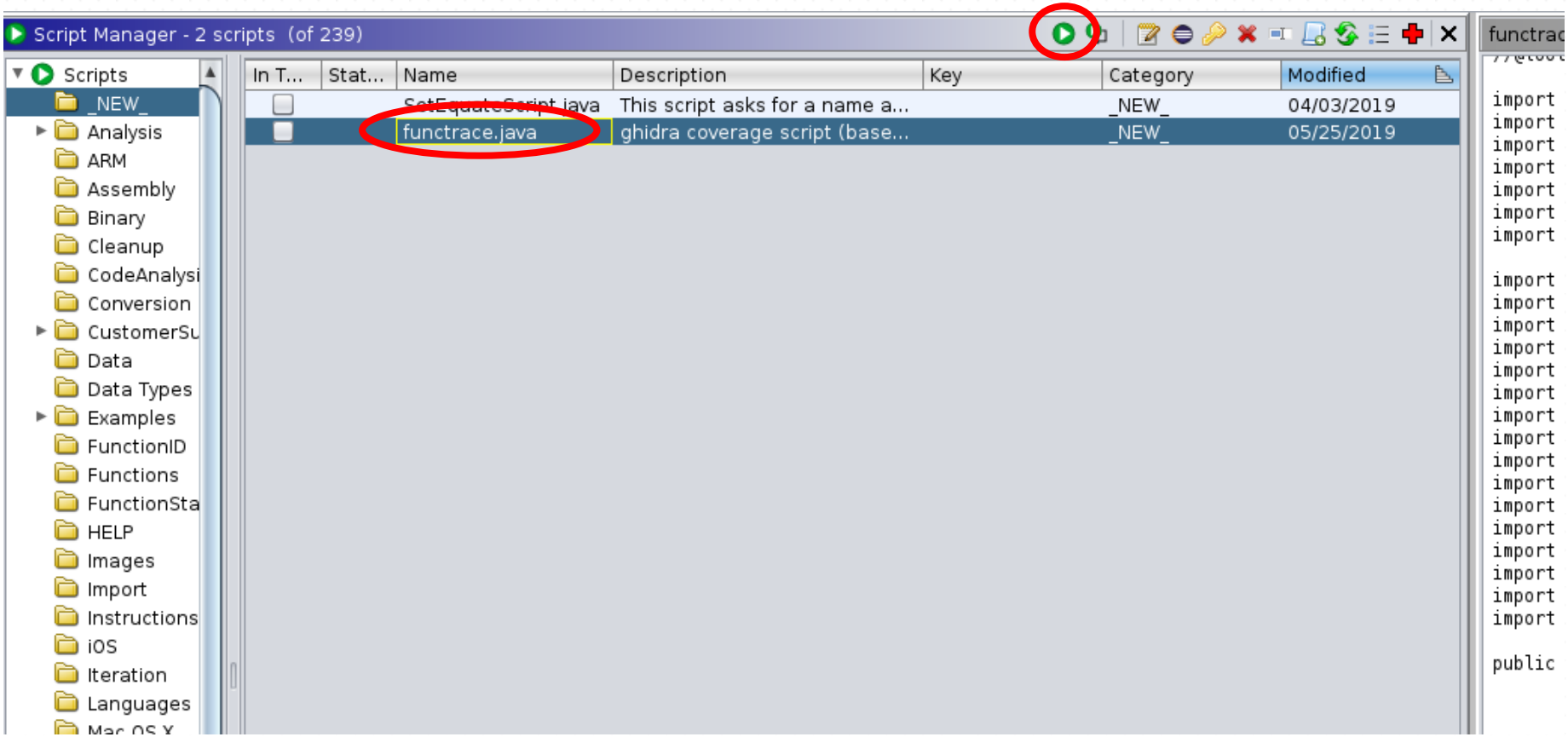
- Create coverage
  - Colors all the executed blocks
- Create comments
  - Function arguments
  - Function return value
- In case of *CRASH*
  - Color the fault block with a different color

# > functrace


functrace
(based on DynamoRIO)
report file
(coverage information and others info)
ghidra script
(graph generation)
coverage graph

# > functrace

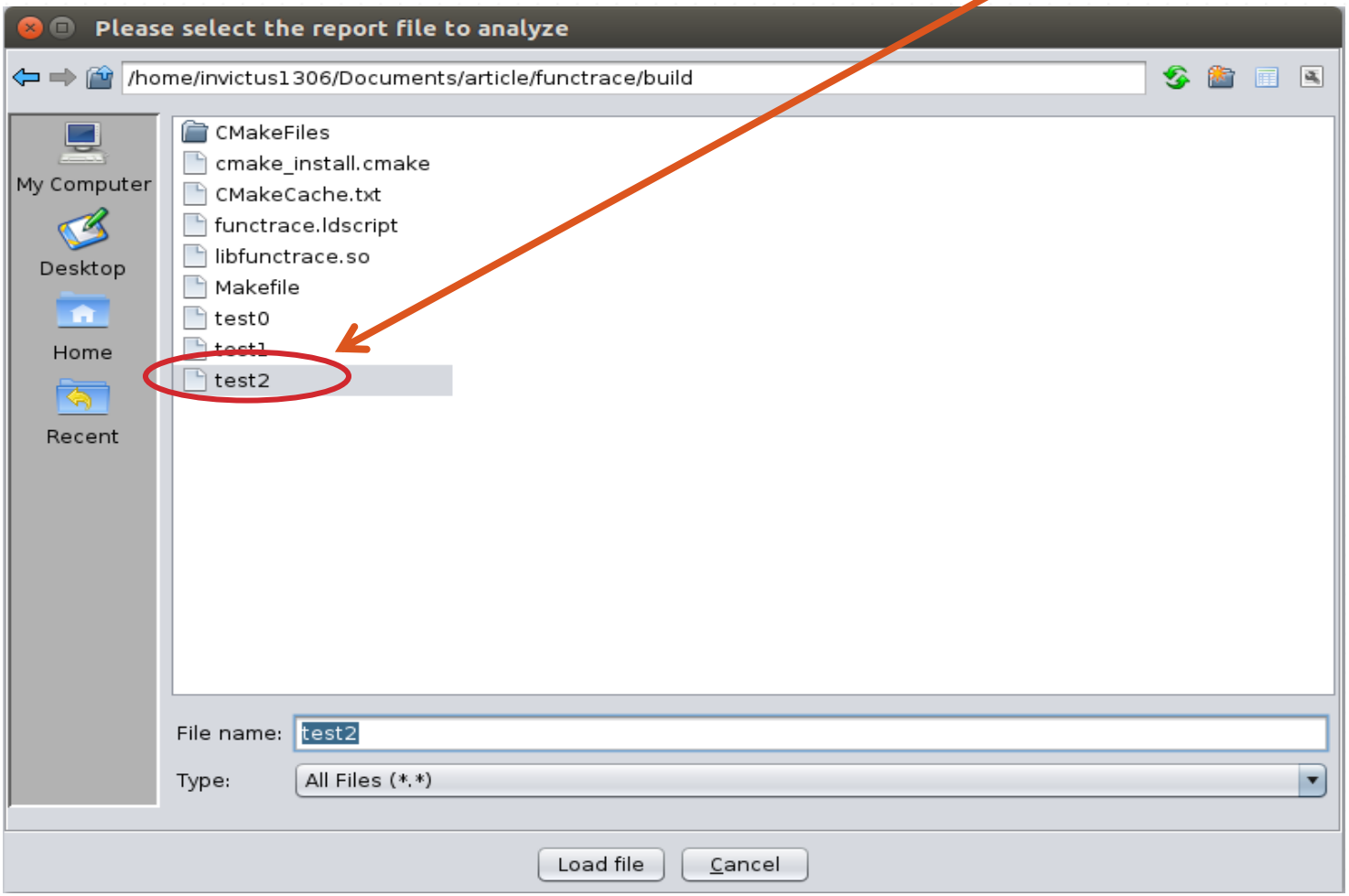
## Ghidra coverage script usage



# > functrace

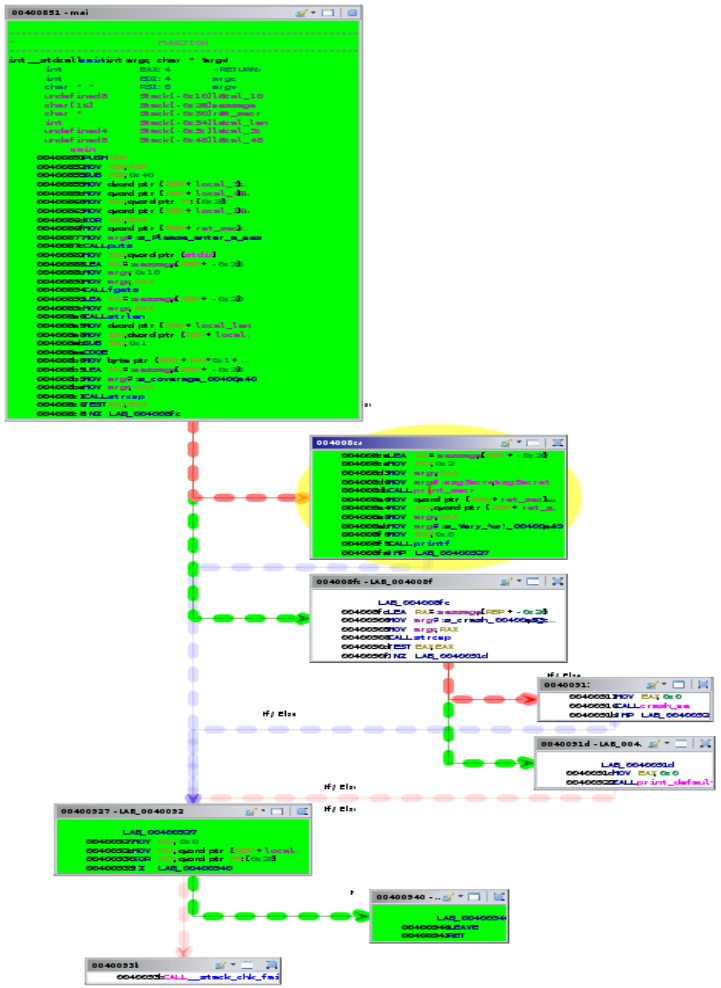
```
$ drrun -c libfunctrace.so -report_file test2 -wrap_function
print_secr -wrap_function_args 3 -- ./tests/simple_test
```

## Ghidra coverage script usage



# > functrace

## Ghidra coverage script



```
004007a6 - print_secr
*************************************************************
* Argument 0 : 0x0000000000401070                           *
* Argument 0 is an address. dump: 0x54 0x68 0x69 0x73 0x20 0x6... *
* Argument 1 : 0x00007fffc305e8b0                           *
* Argument 1 is an address. dump: 0x63 0x6f 0x76 0x65 0x72 0x6... *
* Argument 2 : 0x0000000000000002                           *
* return value is 0x0000000000603830                        *
* The return value is an address. dump: 0x47 0x4f 0x4f 0x44 *
*                                                           *
*************************************************************
char * __stdcall print_secr(char * msg_sec, char * messa...
      char *              RAX:8          <RETURN>
      char *              RDI:8          msg_sec
      char *              RSI:8          message
      int                 EDX:4          num
      char *              Stack[-0x10]:8 ret_secr
      undefined8          Stack[-0x20]:8 local_20
      undefined8          Stack[-0x28]:8 local_28
      undefined4          Stack[-0x2c]:4 local_2c
                print_secr
      004007a6 PUSH  RBP
      004007a7 MOV   RBP,RSP
      004007aa SUB   RSP,0x30
      004007ae MOV   qword ptr [RBP + local_20]...
      004007b2 MOV   qword ptr [RBP + local_28]...
      004007b6 MOV   dword ptr [RBP + local_2c]...
      004007b9 MOV   msg_sec,0x4
      004007be CALL  malloc
      004007c3 MOV   qword ptr [RBP + ret_secr]...
      004007c7 MOV   RAX,qword ptr [RBP + ret_s...
      004007cb MOV   num,0x8
      004007d0 MOV   message=>DAT_004009d8,DAT_...
      004007d5 MOV   msg_sec,RAX
      004007d8 CALL  memcpy
      004007dd MOV   num,qword ptr [RBP + local...
      004007e1 MOV   RAX,qword ptr [RBP + local...
      004007e5 MOV   message,RAX
      004007e8 MOV   msg_sec=>s_Congrats!_%s._Y...
      004007ed MOV   EAX,0x0
      004007f2 CALL  printf
      004007f7 MOV   RAX,qword ptr [RBP + ret_s...
      004007fb LEAVE
      004007fc RET
```

# DEMO

# beebug integration

# > beebug integration

## beebug

https://github.com/invictus1306/beebug

Old features (based on radare2):

- Classify if a crash could be exploitable:
  - Stack overflow on libc
  - Crash on Program Counter
  - Crash on branch
  - Crash on write memory
  - Heap vulnerabilities
  - Read access violation (some exploitable cases)
- Help to analyze a crash (graph view)

# > beebug integration

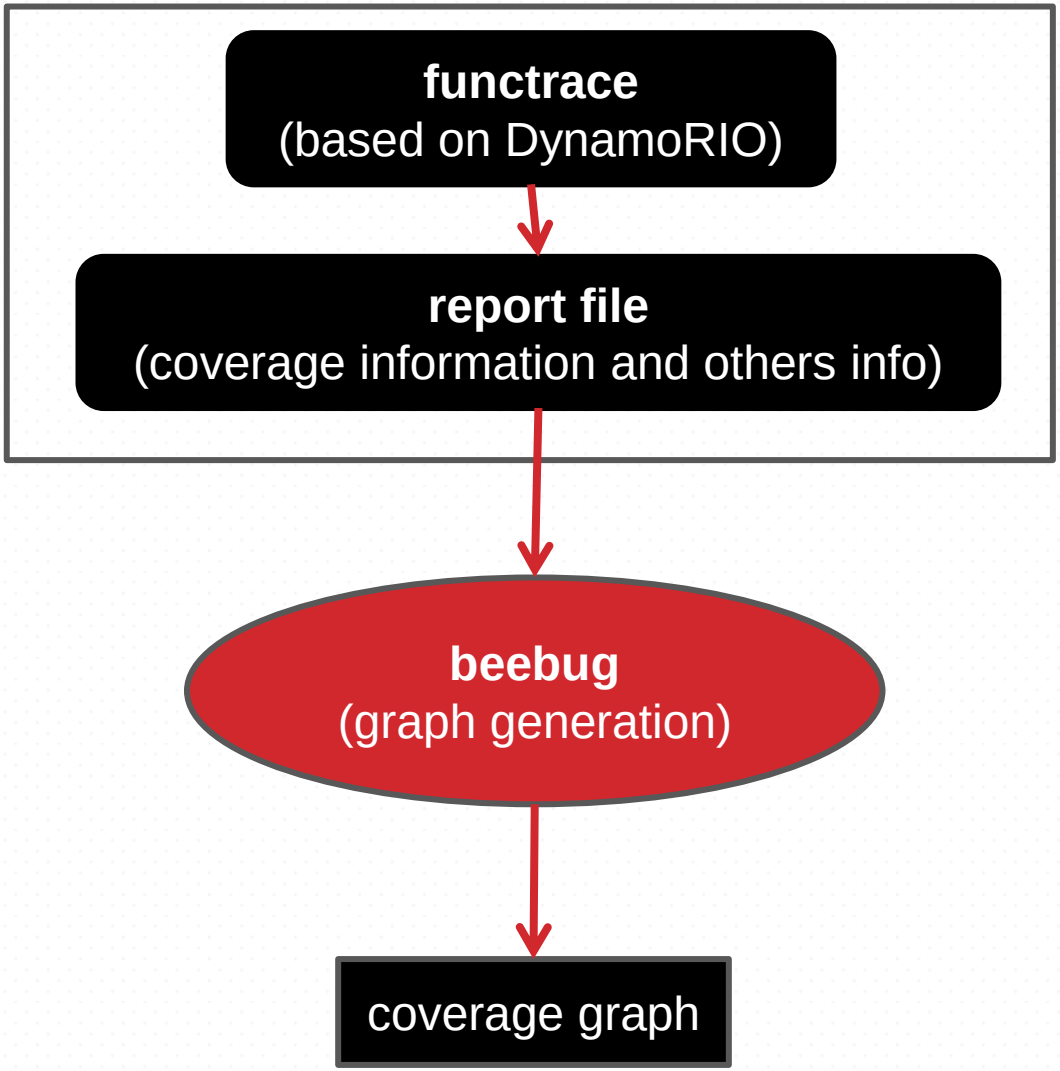
**New features**:

- *DBI* support
- Graph based on *functrace*

We can use *beebug* for:

- Crash analysis
- Graph Generation (*DBI*)
- Crash analysis + Graph Generation

# > beebug integration



# > beebug integration

## Crash analysis (based on r2)

```
invictus1306@invictus1306-VirtualBox:~/Documents/article/beebug$ python3 beebug.py -t ./simple_crash -a
Process with PID 7086 started...
File dbg:///home/invictus1306/Documents/article/beebug/simple_crash  reopened in read-write mode
= attach 7086 7086
ptrace (PT_ATTACH): Operation not permitted
child stopped with signal 11
[+] SIGNAL 11 errno=0 addr=0x00000000 code=1 ret=0
ptrace (PT_ATTACH): Operation not permitted
ptrace (PT_ATTACH): Operation not permitted
Invalid write crash - Generally it is exploitable, the write value/address could be tainted - Invalid write of size 2
backtrace
0  0x400552           sp: 0x0                 0    [sym.vuln]
1  0x400574           sp: 0x7ffcf22b8298      24   [main]  main+25
2  0x7f7ec0d3c830     sp: 0x7ffcf22b82b8      32   [??]  sym.libc_start_main+240
3  0x400459           sp: 0x7ffcf22b8378      192  [??]  entry0+41

registers
rax = 0x00000000
rbx = 0x00000000
rcx = 0x7f7ec10e0b20
rdx = 0x01ca2010
r8 = 0x01ca2000
r9 = 0x0000000d
r10 = 0x7f7ec10e0b78
r11 = 0x00000000
r12 = 0x00400430
r13 = 0x7ffcf22b8390
r14 = 0x00000000
r15 = 0x00000000
rsi = 0x01ca2020
rdi = 0x7f7ec10e0b20
rsp = 0x7ffcf22b8280
rbp = 0x7ffcf22b8290
rip = 0x00400552
rflags = 0x00010202
orax = 0xffffffffffffffff
```

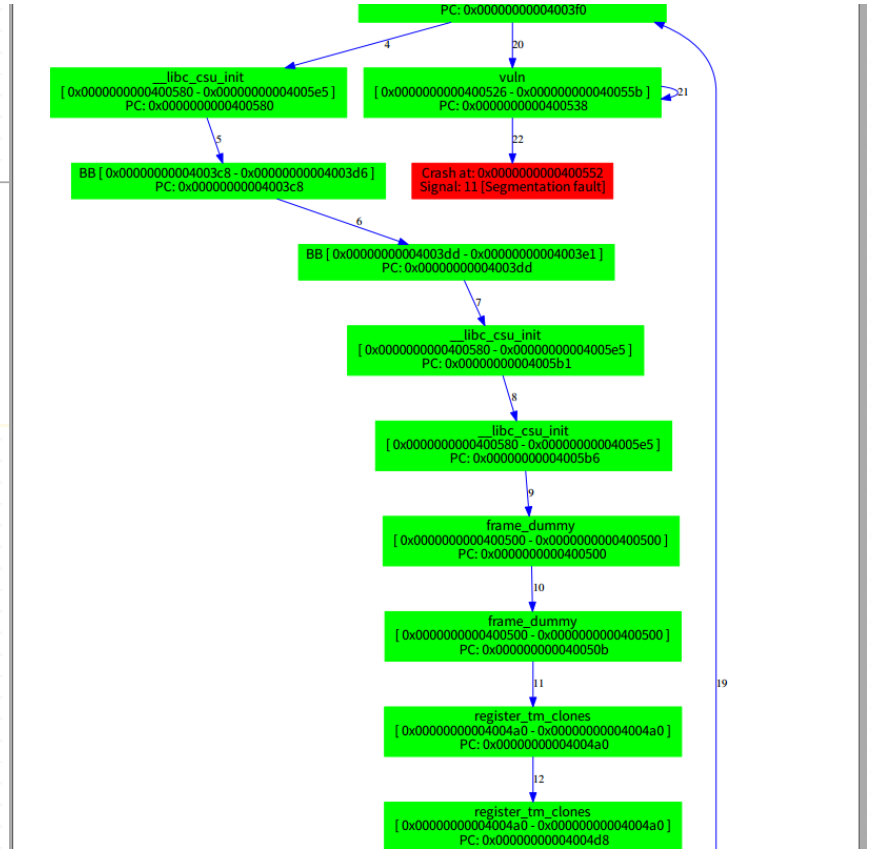
# > beebug integration

## Graph Generation (based on functrace)

```
$python3 –t ./simple_crash –i –r report1 –g graph1
```

Configuration file (instrumentation)

```
[dynamorio]
drrun               = /home/invictus1306/Documents/article/DynamoRIO-Linux-7.0.0-RC1/bin64/drrun
client              = /home/invictus1306/Documents/article/functrace/build/libfunctrace.so
[instrumentation]
disassembly         = True
disas_func          = main
wrap_function       =
wrap_function_args  = 0
cbr                 = True
verbose             = False
.
```

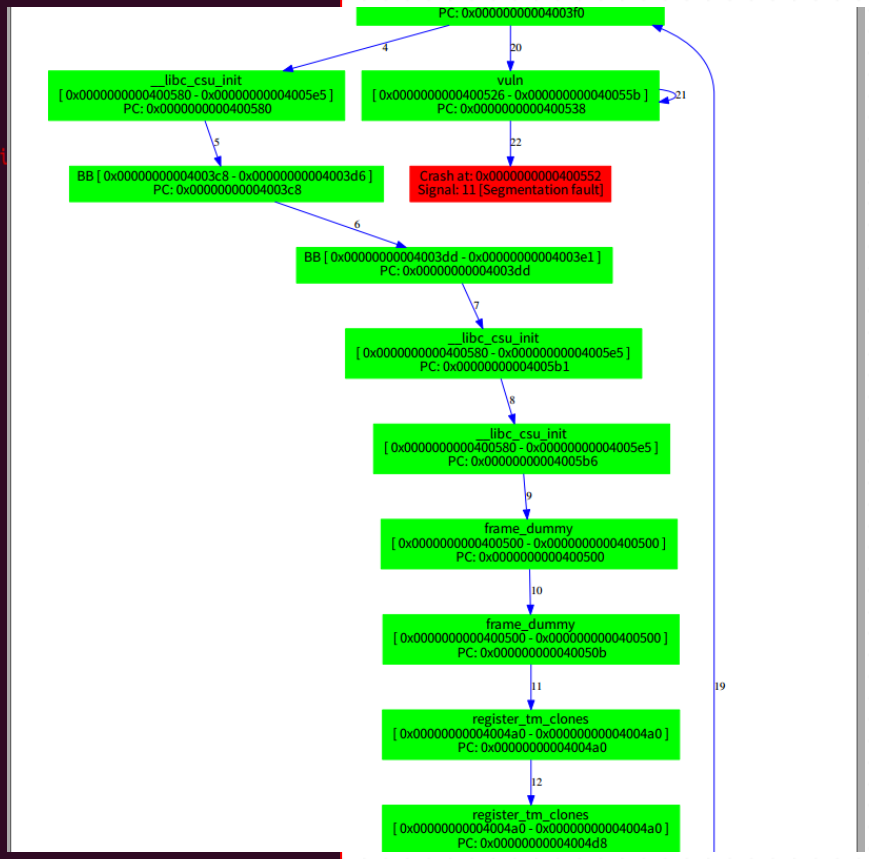


# > beebug integration

## Crash analysis + Graph Generation

```
invictus1306@invictus1306-VirtualBox:~/Documents/article/beebu$ python3 beebug.py -t ./simple_crash -i -r report1 -g graph1 -a
Process with PID 8292 started...
File dbg:///home/invictus1306/Documents/article/beebug/simple_crash  reopened in read-write mode
= attach 8292 8292
ptrace (PT_ATTACH): Operation not permitted
child stopped with signal 11
[+] SIGNAL 11 errno=0 addr=0x00000000 code=1 ret=0
ptrace (PT_ATTACH): Operation not permitted
ptrace (PT_ATTACH): Operation not permitted
Invalid write crash - Generally it is exploitable, the write value/address could be tainted - Invali
backtrace
0  0x400552          sp: 0x0               0    [sym.vuln]
1  0x400574          sp: 0x7fff2af79d88    24   [main]  main+25
2  0x7fb60fc3c830    sp: 0x7fff2af79da8    32   [??]  sym.libc_start_main+240
3  0x7fb60fff67cb    sp: 0x7fff2af79e38    144  [??]  sym.dl_rtld_di_serinfo+29051
4  0x400459          sp: 0x7fff2af79e68    48   [??]  entry0+41

registers
rax = 0x00000000
rbx = 0x00000000
rcx = 0x7fb60ffe0b20
rdx = 0x014d8010
r8 = 0x014d8000
r9 = 0x0000000d
r10 = 0x7fb60ffe0b78
r11 = 0x00000000
r12 = 0x00400430
r13 = 0x7fff2af79e80
r14 = 0x00000000
r15 = 0x00000000
rsi = 0x014d8020
rdi = 0x7fb60ffe0b20
rsp = 0x7fff2af79d70
rbp = 0x7fff2af79d80
rip = 0x00400552
rflags = 0x00010202
orax = 0xffffffffffffffff
```



# DEMO

# > beebug integration

## beebug limitation

- If the program requires user input (at runtime), it is not possible to add it (based on r2pipe)
- graph view (based on pydot/graphiz) does not work well with large programs

# Future features

# > Future features

- Ghidra plugin
  - visual setup interface
  - run DR directly from Ghidra
  - store and compare different coverage analysis
- Add more functionality to functrace
- Support for Android

# Thank you!

*https://github.com/invictus1306/functrace*

**Andrea Sindoni**

**@ invictus1306**