trapfuzzer ：

coverage-guided binary fuzzing with breakpoint

Sili Luo| @hac425

TRACK 2

HITB SECCONF SIN-2021

About Me

- Security Researcher at Huawei ROOT Lab
- Focus on software vulnerability research
- Github: https://github.com/hac425xxx
- Twitter: https://twitter.com/hac425/
- Blog: https://www.cnblogs.com/hac425

Outline

1. Background
2. Implementation of trapfuzzer
3. How to Fuzz with trapfuzzer and results
4. Future Plans

Background

What is Fuzzing?

```python
def fuzzer(max_length=100, char_start=32, char_range=32):
    """A string of up to `max_length` characters
       in the range [`char_start`, `char_start` + `char_range`]"""
    string_length = random.randrange(0, max_length + 1)
    out = ""
    for i in range(0, string_length):
        out += chr(random.randrange(char_start, char_start + char_range))
    return out
```

What is Coverage-Guided Fuzzing?

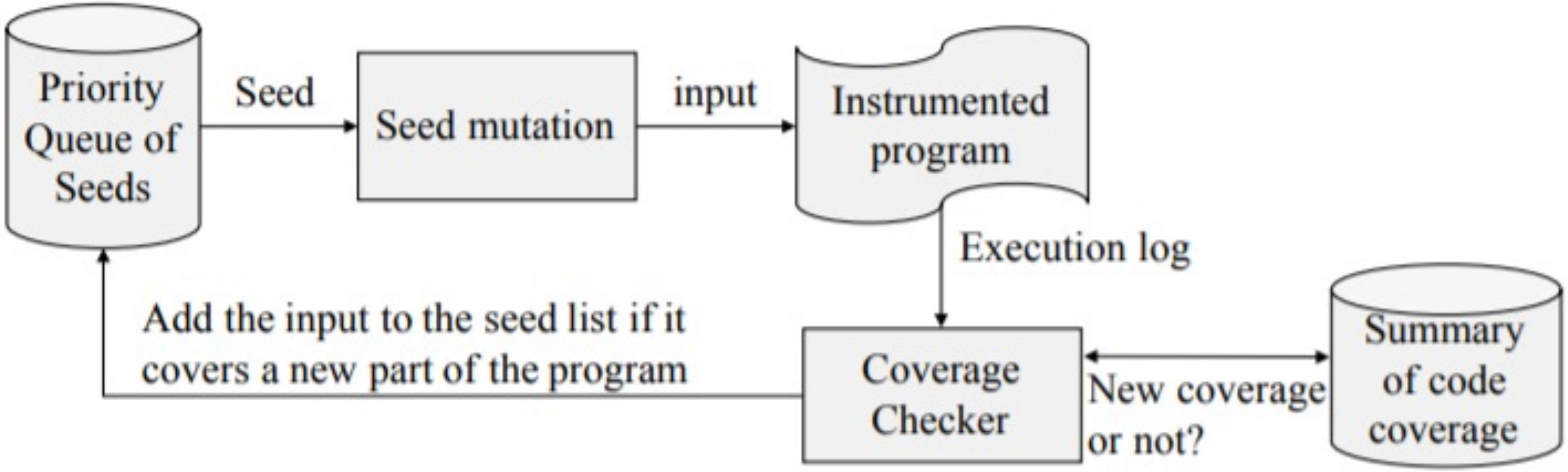


Fig. 1: fuzz testing process.

Background

and achieves lightweight, low-overhead coverage guided fuzzing for closed source code by:

1. Enumerating the start offset of every basic block in the program/library. This is done with a IDAPython script
2. At runtime, in the fuzzed process, replacing the first byte of every undiscovered basic block breakpoint instruction (int3 on Intel). The original byte and the corresponding offset in the c bitmap are stored in a dedicated shadow memory mapping whose address can be compute address of the modified library, and
3. Installing a SIGTRAP handler that will:
   a. Retrieve the faulting address and compute the offset in the library as well as the addr corresponding entry in the shadow memory
   b. Mark the basic block as found in the global coverage bitmap
   c. Replace the breakpoint with the original byte
   d. Resume execution

CVE-2020-11764
An out-of-bounds write (of presumably image pixels) on the heap in the copyIntoFrameBuffer function.

CVE-2020-11763
A bug that caused a std::vector to be read out-ouf-bounds. Afterwards, the calling code would write into an element slot of this vector, thus likely corrupting memory.

CVE-2020-11762
An out-of-bounds memcpy that was reading data out-of-bounds and afterwards potentially writing it out-of-bounds as well.

CVE-2020-11760, CVE-2020-11761, CVE-2020-11758
Various out-of-bounds reads of pixel data and other data structures.

CVE-2020-11765
An out-of-bounds read on the stack, likely due to an off-by-one error previously overwriting a string null terminator on the stack.

CVE-2020-11759

https://googleprojectzero.blogspot.com/2020/04/fuzzing-imageio.html

- Prep
  - IDA analysis
  - Basic blocks file generation
  - Modification of the exe/dll files
- Execution
  - Catch 0xCC exceptions
  - If in the basic block list
    - Record location
    - Replace 0xCC with original value
    - EIP = EIP – 1

Vendor	CVE count
Microsoft	27
Adobe	45
Apple*	2

http://www.powerofcommunity.net/poc2018/jaanus.pdf

Inspiration

Combining my previous fuzzing experience and these two security research, I realized:

1. The existing excellent Fuzz tools (AFL, honggfuzz) **are not perfect**, and there are still unsupported or incompletely supported scenarios, such as complex large, closed-source programs and some new platforms.
2. Using **relatively inefficient instrument methods** for fuzz testing can also obtain **better results** than complete black box fuzz testing, such as instruments with breakpoints.

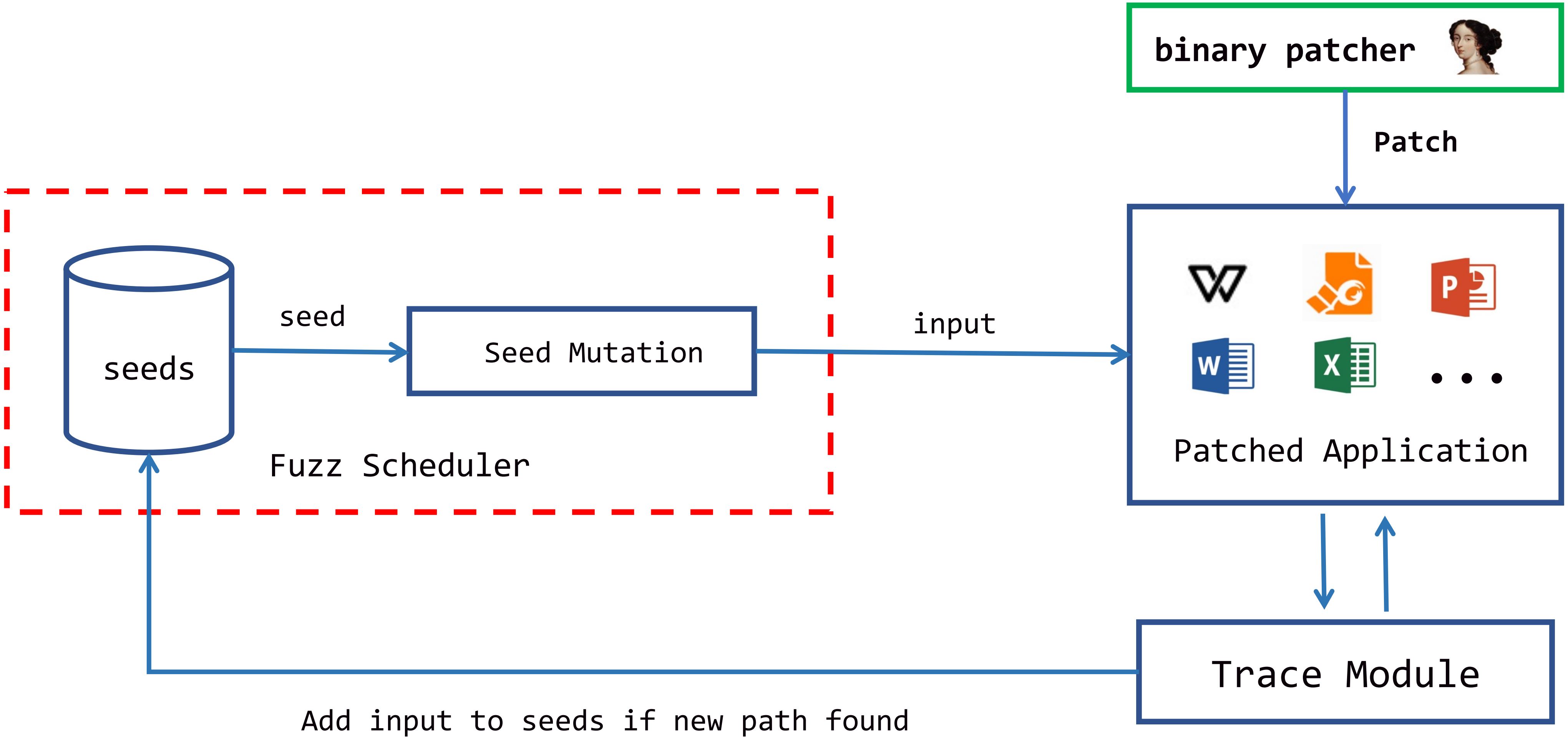
I found that **fuzzing tools for large closed-source Linux software are rare**, commonly used linux fuzzing tools are Peach and AFL, they have some shortcomings:

1. Peach: fuzzing without coverage information.
2. AFL Qemu Mode: only suitable for relatively small programs, such as image parse library.

So I decided to develop a fuzzer based on a breakpoint mechanism to support some scenarios that are not covered by existing tools, such as **large, closed-source file parsing programs and provide coverage support.**

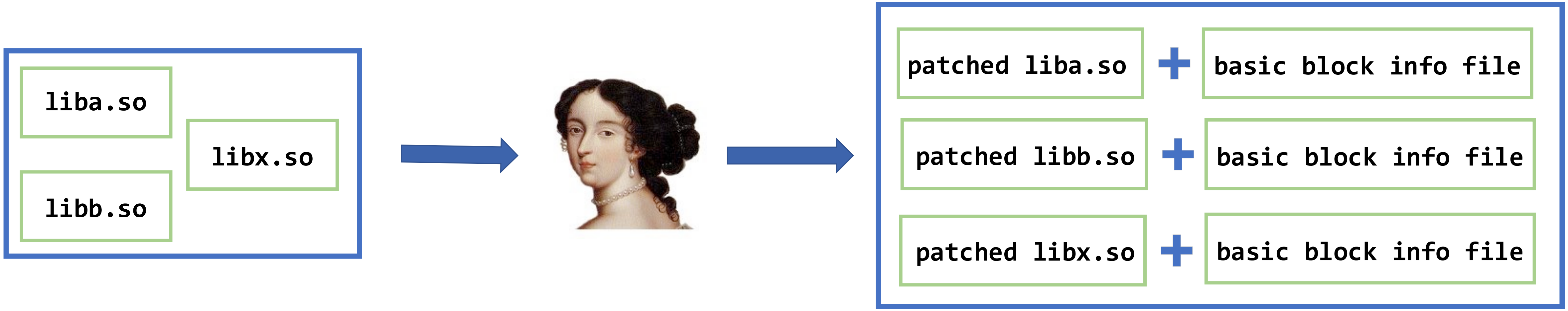
Implementation
(Version 0.1)

Overview


binary patcher
Patch
seeds
seed
Seed Mutation
input
Fuzz Scheduler
Patched Application
• • •
Trace Module
Add input to seeds if new path found

binary patcher

1. Use IDAPython script to get all basic blocks of binary
2. Replace the first instruction of every basic blocks with breakpoint instruction and save the original instruction to **basic-block-info-file**.



binary patcher – basic-block-info-file example

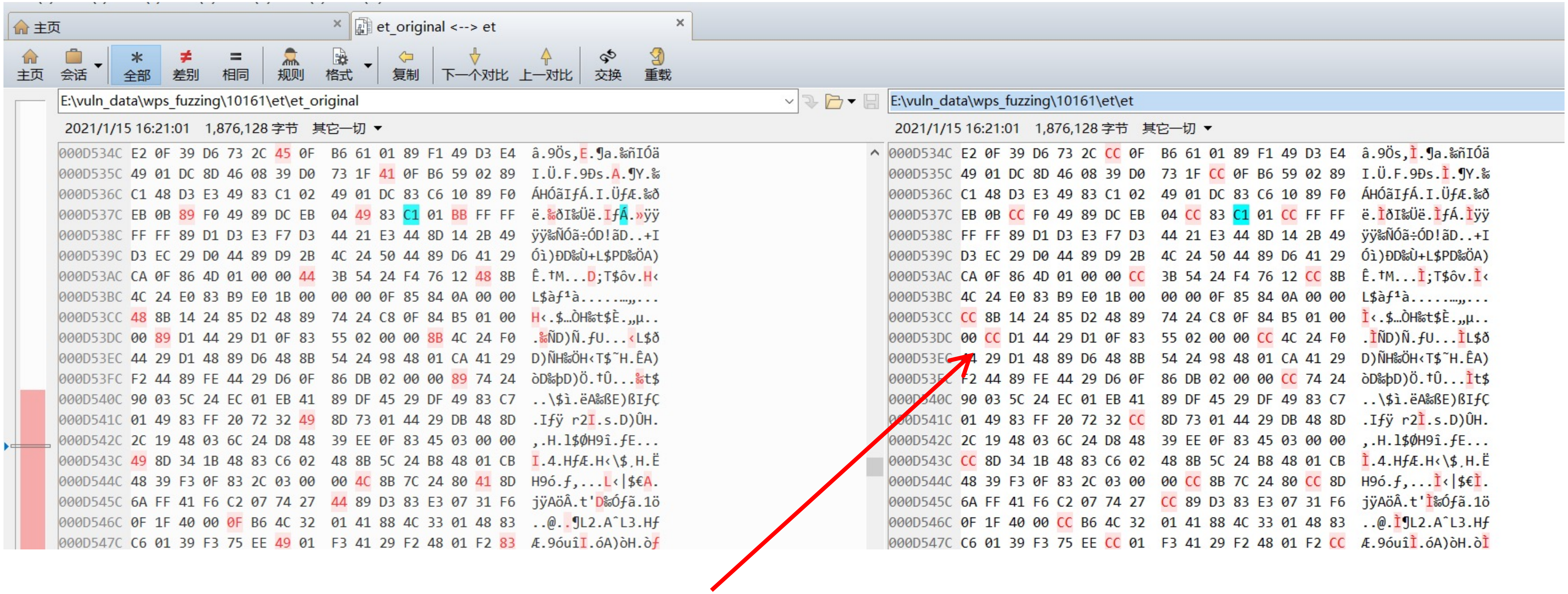
JSTARO24.OCX-bb.txt ×

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 00 60 B7 00 0D 00 00 00 4A 53 54 41 52 4F 32 34  .`·.....JSTARO24
0010h: 2E 4F 43 58 00 00 10 00 00 00 04 00 00 01 00 00  .OCX............
0020h: 00 FF 06 10 00 00 06 04 00 00 01 00 00 00 FF 0C  .ÿ...........ÿ.
0030h: 10 00 00 0C 04 00 00 01 00 00 00 FF 12 10 00 00  ...........ÿ....
0040h: 12 04 00 00 01 00 00 00 FF 18 10 00 00 18 04 00  ........ÿ.......
0050h: 00 01 00 00 00 FF 1E 10 00 00 1E 04 00 00 01 00  .....ÿ..........
0060h: 00 00 FF 24 10 00 00 24 04 00 00 01 00 00 00 FF  ..ÿ$...$.......ÿ
0070h: 2A 10 00 00 2A 04 00 00 01 00 00 00 FF 30 10 00  *...*.......ÿ0..
0080h: 00 30 04 00 00 01 00 00 00 FF 36 10 00 00 36 04  .0.......ÿ6...6.
0090h: 00 00 01 00 00 00 FF 3C 10 00 00 3C 04 00 00 01  ......ÿ<...<....
00A0h: 00 00 00 B8 44 10 00 00 44 04 00 00 01 00 00 00  ...¸D...D.......
00B0h: B8 A0 72 8E 00 A0 66 8E 00 01 00 00 00 8D A8 72  ¸ rŽ. fŽ......¨r
00C0h: 8E 00 A8 66 8E 00 01 00 00 00 B8 68 1A 09 00 68  Ž.¨fŽ.....¸h...h
00D0h: 0E 09 00 01 00 00 00 FF CB 07 69 00 CB FB 68 00  .......ÿË.i.Ëûh.
00E0h: 01 00 00 00 FF 8A 10 00 00 8A 04 00 00 01 00 00  ....ÿŠ...Š......
00F0h: 00 FF 90 10 00 00 90 04 00 00 01 00 00 00 FF 96  .ÿ...........ÿ–
0100h: 10 00 00 96 04 00 00 01 00 00 00 FF 9C 10 00 00  ...–.......ÿœ...
0110h: 9C 04 00 00 01 00 00 00 FF A2 10 00 00 A2 04 00  œ.......ÿ¢...¢..
0120h: 00 01 00 00 00 FF A8 10 00 00 A8 04 00 00 01 00  .....ÿ¨...¨.....
0130h: 00 00 8B AF 10 00 00 AF 04 00 00 01 00 00 00 8B  ..‹¯...¯.......‹
0140h: C0 10 00 00 C0 04 00 00 01 00 00 00 8B D0 10 00  À...À.......‹Ð..
```

Template Results - bb.bt

Name	Value	Start	Size	Color
uint32 rva_size	B76000h	0h	4h	Fg: Bg:
struct MODULE_NAME module_name	JSTARO24.OCX	4h	11h	Fg: Bg:
uint32 length	13	4h	4h	Fg: Bg:
ubyte data[13]		8h	Dh	Fg: Bg:
struct BB_INFO bi[0]	id:0, rva:0x1000, foff:0x400, instr size:0x1	15h	Dh	Fg: Bg:
uint32 rva	4096	15h	4h	Fg: Bg:
uint32 foff	1024	19h	4h	Fg: Bg:
uint32 instr_size	1	1Dh	4h	Fg: Bg:
ubyte instr[1]		21h	1h	Fg: Bg:
struct BB_INFO bi[1]	id:1, rva:0x1006, foff:0x406, instr size:0x1	22h	Dh	Fg: Bg:
struct BB_INFO bi[2]	id:2, rva:0x100c, foff:0x40c, instr size:0x1	2Fh	Dh	Fg: Bg:
struct BB_INFO bi[3]	id:3, rva:0x1012, foff:0x412, instr size:0x1	3Ch	Dh	Fg: Bg:
struct BB_INFO bi[4]	id:4, rva:0x1018, foff:0x418, instr size:0x1	49h	Dh	Fg: Bg:
struct BB_INFO bi[5]	id:5, rva:0x101e, foff:0x41e, instr size:0x1	56h	Dh	Fg: Bg:
struct BB_INFO bi[6]	id:6, rva:0x1024, foff:0x424, instr size:0x1	63h	Dh	Fg: Bg:
struct BB_INFO bi[7]	id:7, rva:0x102a, foff:0x42a, instr size:0x1	70h	Dh	Fg: Bg:

binary patcher - example



breakpoint instruction in x86

binary patcher - example

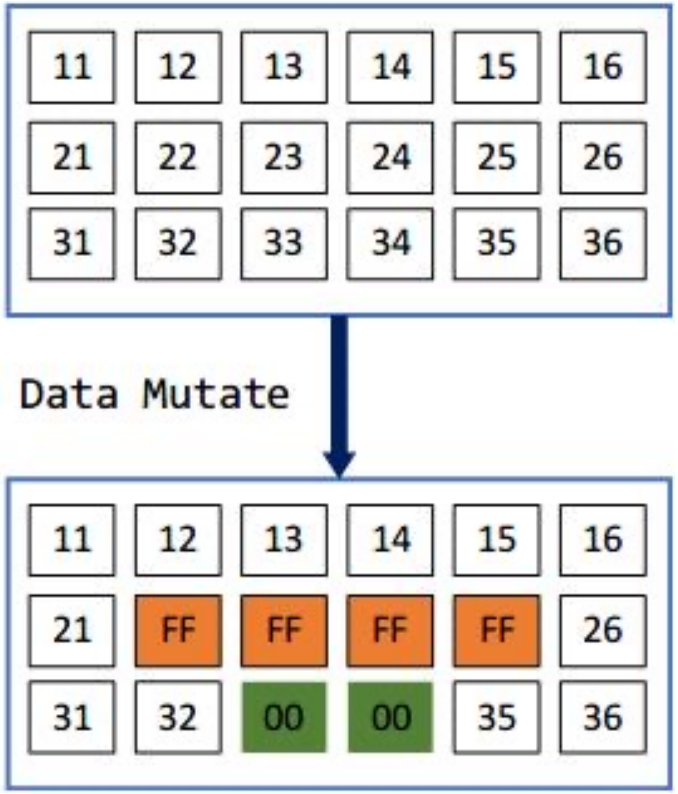
unpatched

```
.text:0000000000400B36                 mov     rbp, rsp
.text:0000000000400B39                 sub     rsp, 30h
.text:0000000000400B3D                 mov     [rbp+argc], edi
.text:0000000000400B40                 mov     [rbp+argv], rsi
.text:0000000000400B44                 mov     rax, fs:28h
.text:0000000000400B4D                 mov     [rbp+var_8], rax
.text:0000000000400B51                 xor     eax, eax
.text:0000000000400B53                 cmp     [rbp+argc], 1
.text:0000000000400B57                 jg      short loc_400B7C
.text:0000000000400B59                 mov     rax, [rbp+argv]
.text:0000000000400B5D                 mov     rax, [rax]
.text:0000000000400B60                 mov     rsi, rax
.text:0000000000400B63                 mov     edi, offset aSyntaxSInputFi ; "Syntax: %s <input file>\n"
.text:0000000000400B68                 mov     eax, 0
.text:0000000000400B6D                 call    _printf
.text:0000000000400B72                 mov     edi, 1          ; status
.text:0000000000400B77                 call    _exit
.text:0000000000400B7C ; ---------------------------------------------------------------------------
.text:0000000000400B7C
.text:0000000000400B7C loc_400B7C:                             ; CODE XREF: main+22↑j
.text:0000000000400B7C                 mov     [rbp+len], 0
.text:0000000000400B83                 mov     rax, [rbp+argv]
.text:0000000000400B87                 add     rax, 8
.text:0000000000400B8B                 mov     rax, [rax]
.text:0000000000400B8E                 lea     rdx, [rbp+len]
.text:0000000000400B92                 mov     rsi, rdx        ; len
.text:0000000000400B95                 mov     rdi, rax        ; path
.text:0000000000400B98                 call    read_to_buf
.text:0000000000400B9D                 mov     [rbp+data], rax
.text:0000000000400BA1                 mov     rax, [rbp+data]
.text:0000000000400BA5                 movzx   eax, byte ptr [rax]
.text:0000000000400BA8                 movsx   eax, al
.text:0000000000400BAB                 sub     eax, 41h
.text:0000000000400BAE                 cmp     eax, 4          ; switch 5 cases
.text:0000000000400BB1                 ja      loc_400C7F      ; jumptable 0000000000400BC1 default case
.text:0000000000400BB7                 mov     eax, eax
```

patched

```
.text:0000000000400B36                 mov     rbp, rsp
.text:0000000000400B39                 sub     rsp, 30h
.text:0000000000400B3D                 mov     [rbp-24h], edi
.text:0000000000400B40                 mov     [rbp-30h], rsi
.text:0000000000400B44                 mov     rax, fs:28h
.text:0000000000400B4D                 mov     [rbp-8], rax
.text:0000000000400B51                 xor     eax, eax
.text:0000000000400B53                 cmp     dword ptr [rbp-24h], 1
.text:0000000000400B57                 jg      short loc_400B7C
.text:0000000000400B59                 int     3               ; Trap to Debugger
.text:0000000000400B5A                 mov     eax, [rbp-30h]
.text:0000000000400B5D                 mov     rax, [rax]
.text:0000000000400B60                 mov     rsi, rax
.text:0000000000400B63                 mov     edi, offset aSyntaxSInputFi ; "Syntax: %s <input
.text:0000000000400B68                 mov     eax, 0
.text:0000000000400B6D                 call    sub_400770
.text:0000000000400B72                 mov     edi, 1
.text:0000000000400B77                 call    sub_4007F0
.text:0000000000400B7C
.text:0000000000400B7C loc_400B7C:                             ; CODE XREF: main+22↑j
.text:0000000000400B7C                 int     3               ; Trap to Debugger
.text:0000000000400B7C main            endp
.text:0000000000400B7C
.text:0000000000400B7D ; ---------------------------------------------------------------------------
.text:0000000000400B7D                 db      45h
.text:0000000000400B7D                 in      al, dx
.text:0000000000400B7D ; ---------------------------------------------------------------------------
.text:0000000000400B7F                 db    0
.text:0000000000400B80 qword_400B80    dq 48D0458B48000000h, 8D48008B4808C083h, 0C78948D68948EC5
```

Seed Mutation



data mutate example

RadamsaMutator

use radamsa to generate mutated testcases

https://gitlab.com/akihe/radamsa

TinyMutator

1. support basic mutation strategies, such as byte flipping and random insertion of boundary values (such as 0xFFFFFFFF) .
2. supporting config **the variation ratio** of mutation.

Fuzzer Module

Workflow

1. First load the initial testcases to seed queue and do **corpus distillation** on seed queue.
2. Then it will traverse the seed queue, mutate the testcase, then use the trace module to start the target process, finally get the coverage and execution status of the process.
3. If new coverage is found, the **new testcase will be added to the seed queue**.
4. If crash is found, the crash information is saved, such as registers, stack trace.

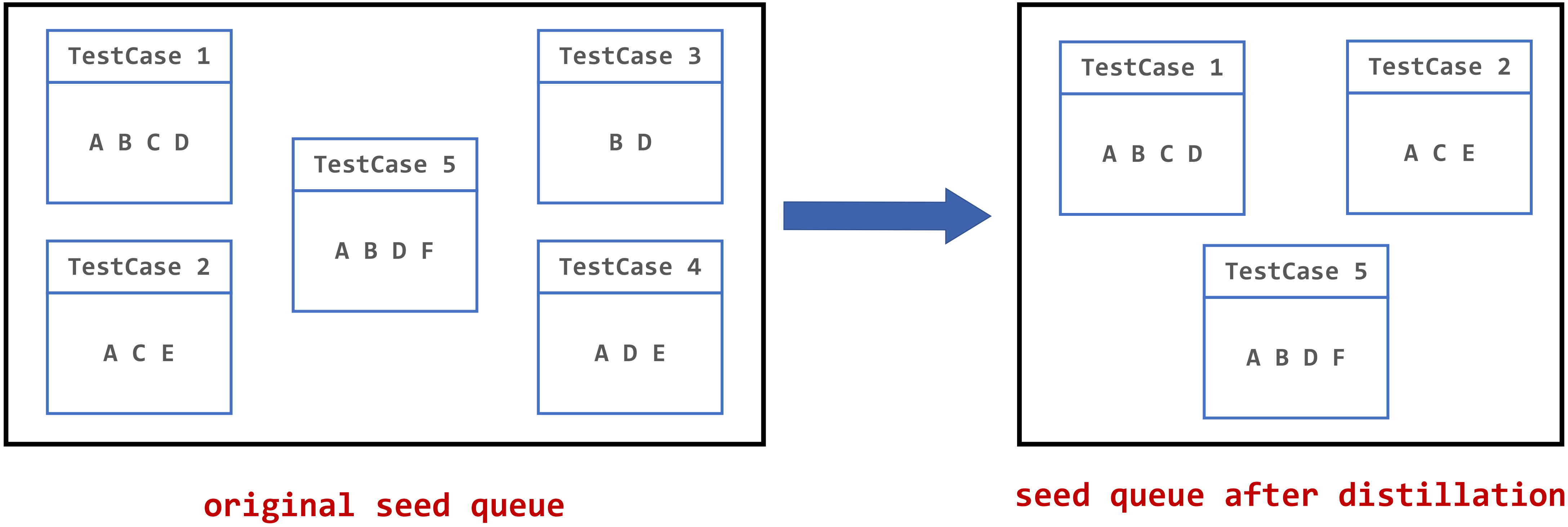
Scheduling strategies

- The score of testcases that discover new coverage will increase ↑
- The score of testcases that trigger DOS will decrease ↓

Fuzzer Module - Corpus Distillation

The workflow of corpus distillation is as follows:

1. let the program process the testcases in the seed queue one by one.
2. then only save the testcases that can generate new coverage.



Trace module - Theory

Prep

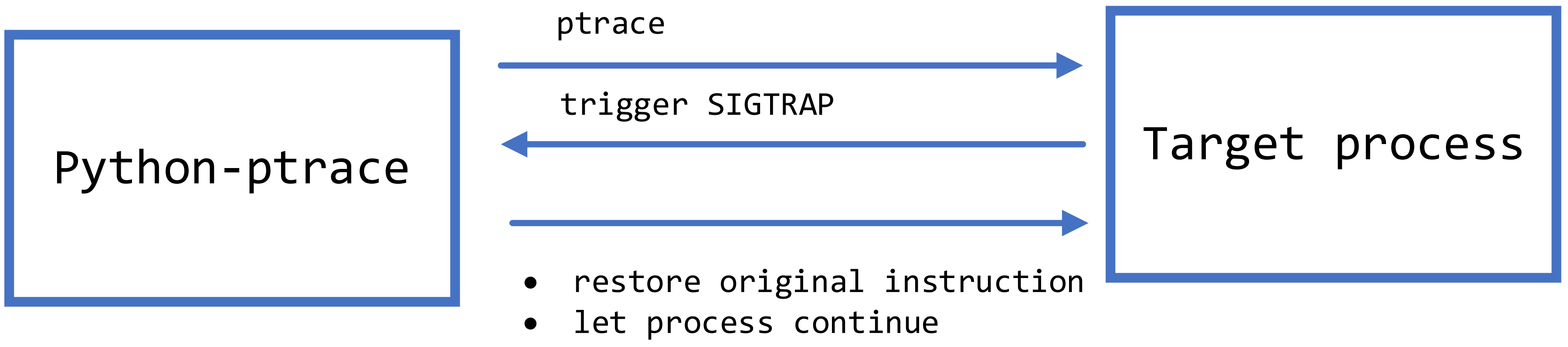
- Use bb-patcher module.
- Replace the first instruction of basic block with breakpoint instruction (0xcc)

Execution

1. Catch 0xCC exceptions
2. Record location
3. Replace 0xCC with original value
4. Let process continue

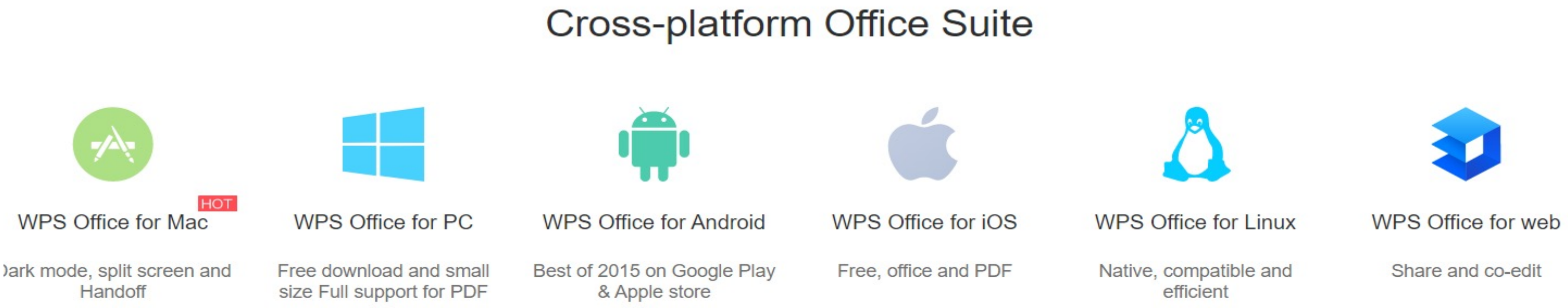
Trace module #1 (PythonPtraceTracer)

1. First use **create_and_attach_process** to create the target process
2. Use **cont** to let the process continue, and use **waitSignals** to wait for the process to trigger signals, such as SIGABORT, SIGTRAP.
3. If the process triggers the **SIGTRAP** signal, record the value of the PC at this time and replace the breakpoint instruction with the original instruction
4. then let the process continue to execute, **goto 2**



Lets Fuzz WPS

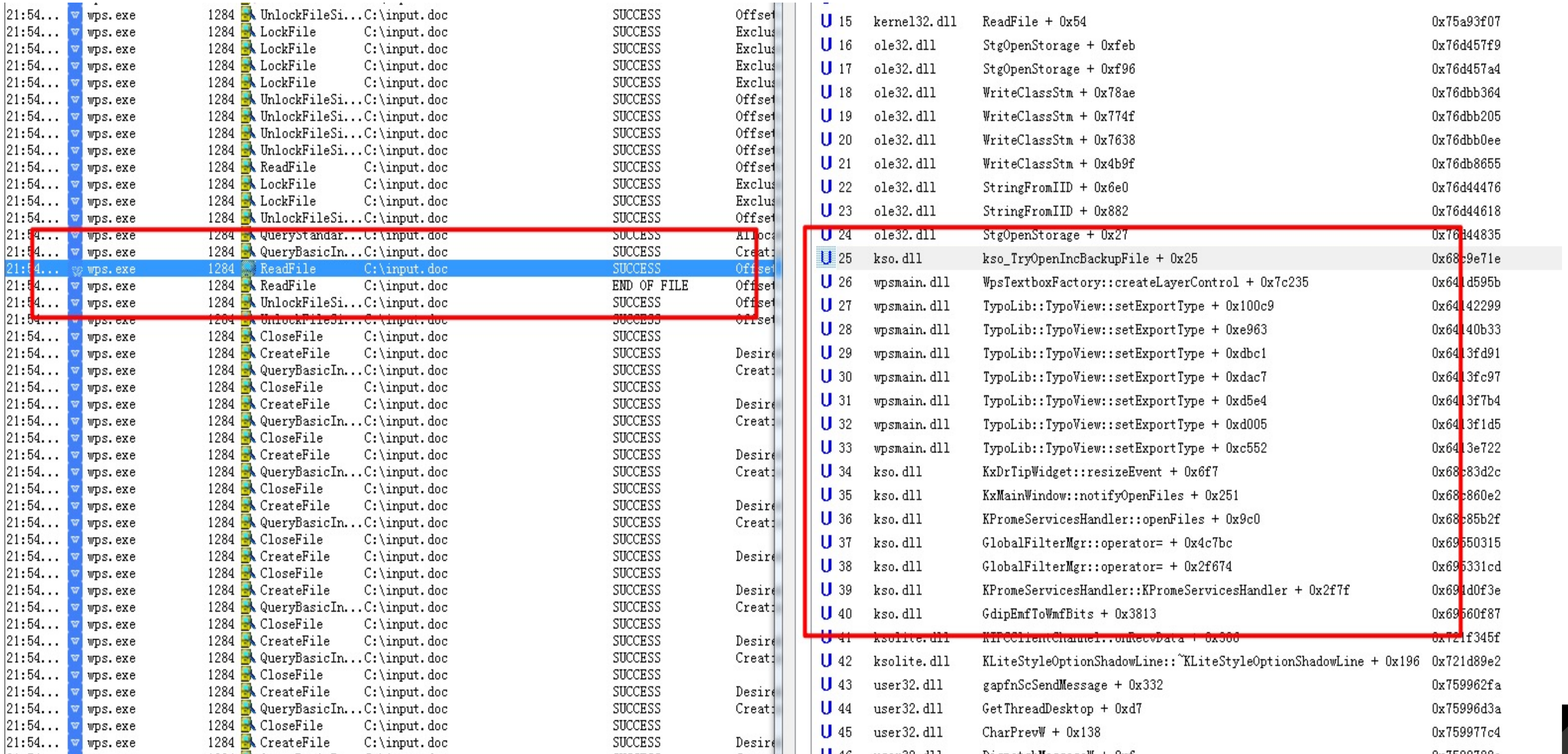
WPS is an office processing software in China that supports viewing and editing DOC, XLS, PPT and other files



- trapfuzzer collect coverage by patch breakpoint instruction to target module
- therefore, it is necessary to find which module need be patched, that is, the module that is responsible for **parsing the file**

Lets Fuzz WPS - Find Target Module

In Windows platform, we can use **process monitor** to monitor the behavior of the program at runtime, and be able to obtain the **call stack**, by using this information can quickly locate the **data processing module**.



Lets Fuzz WPS

What about the Linux platform?

ltrace

strace

No call stack

1. GDB can obtain the call stack stably
2. File operations of the process can be tracked through breakpoints
3. Use GDB's scripting mechanism to automate

Lets Fuzz WPS - Linux Version of FileMon

```python
class FopenGoodFile(gdb.Breakpoint):
    def __init__(self, name):
        super(FopenGoodFile, self).__init__(
            name, gdb.BP_BREAKPOINT, internal=False)
        self.hitcount = 0

    def stop(self):
        rdi = read_register("rdi")
        fname = read_memory(rdi, 0x100)
        # print("open: {}".format(fname))
        fname = fname[:fname.find(b"\x00")]

        if b".doc" in fname:
            current_frame = gdb.selected_frame()
            caller = current_frame.older().pc()
            print("[FopenGoodFile] open {}, set bp on
                fname, caller))
            get_backtrace()
            AddFpBp("*{}".format(caller))
        return False
```

```
[FreadGoodFp] fread fp: 62681600
#0  0x00007ffff69bb3f0 in __GI__IO_fread (buf=0x6cc2041, size=1, count=49147, fp=0x3bc7200) at iofread.c:31
#1  0x00007fffec241e1c in XMLPlatformUtils::readFileBuffer(void*, unsigned int, unsigned char*) () at /opt/kingsoft/wps-office/office6/libkso.
#2  0x00007fffec249311 in XMLReader::refreshCharBuffer() () at /opt/kingsoft/wps-office/office6/libkso.so
#3  0x00007fffec26c49d in ReaderMgr::peekNextChar() () at /opt/kingsoft/wps-office/office6/libkso.so
#4  0x00007fffec25da68 in XMLScanner::scanProlog() () at /opt/kingsoft/wps-office/office6/libkso.so
#5  0x00007fffec25d6b4 in XMLScanner::scanDocument(InputSource const&) () at /opt/kingsoft/wps-office/office6/libkso.so
#6  0x00007fffec25d2fd in XMLScanner::scanDocument(unsigned short const*) () at /opt/kingsoft/wps-office/office6/libkso.so
#7  0x00007fffec351743 in SAX2XMLReaderImpl::parse(unsigned short const*) () at /opt/kingsoft/wps-office/office6/libkso.so
#8  0x00007fffec43d37e in  () at /opt/kingsoft/wps-office/office6/libkso.so
#9  0x00007fffec41e8a3 in OpenXmlPackage::_Open(unsigned short const*, unsigned int, OpcPackageIOType) () at /opt/kingsoft/wps-office/office6/
#10 0x00007fffec5d6a04 in WordProcessDocument::Open(unsigned short const*, WordProcessDocument::DocumentType) () at /opt/kingsoft/wps-office/o
#11 0x00007fffe7dcc120 in  () at /opt/kingsoft/wps-office/office6/libwpsio.so
#12 0x00007fffe7d7aaf8 in filterpluginFormatCorrect_docx () at /opt/kingsoft/wps-office/office6/libwpsio.so
#13 0x00007ffff357cc3b in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#14 0x00007ffff358815e in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#15 0x00007ffff35a4cca in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#16 0x00007ffff358d04b in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#17 0x00007ffff358c502 in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#18 0x00007ffff35940bf in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#19 0x00007ffff3879ce1 in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#20 0x00007ffff305374b in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so


[ReadGoodFd] read fd: 29
#0  0x00007ffff6a4c180 in __GI___libc_read (fd=29, buf=0x6cc2041, nbytes=45056) at ../sysdeps/unix/sysv/linux/read.c:27
#1  0x00007ffff69c76e2 in __GI__IO_file_xsgetn (fp=0x3bc7200, data=<optimized out>, n=49147) at fileops.c:1364
#2  0x00007ffff69bb431 in __GI__IO_fread (buf=<optimized out>, size=1, count=49147, fp=0x3bc7200) at iofread.c:38
#3  0x00007fffec241e1c in XMLPlatformUtils::readFileBuffer(void*, unsigned int, unsigned char*) () at /opt/kingsoft/wps-office/office6/libkso.
#4  0x00007fffec249311 in XMLReader::refreshCharBuffer() () at /opt/kingsoft/wps-office/office6/libkso.so
#5  0x00007fffec26c49d in ReaderMgr::peekNextChar() () at /opt/kingsoft/wps-office/office6/libkso.so
#6  0x00007fffec25da68 in XMLScanner::scanProlog() () at /opt/kingsoft/wps-office/office6/libkso.so
#7  0x00007fffec25d6b4 in XMLScanner::scanDocument(InputSource const&) () at /opt/kingsoft/wps-office/office6/libkso.so
#8  0x00007fffec25d2fd in XMLScanner::scanDocument(unsigned short const*) () at /opt/kingsoft/wps-office/office6/libkso.so
#9  0x00007fffec351743 in SAX2XMLReaderImpl::parse(unsigned short const*) () at /opt/kingsoft/wps-office/office6/libkso.so
#10 0x00007fffec43d37e in  () at /opt/kingsoft/wps-office/office6/libkso.so
#11 0x00007fffec41e8a3 in OpenXmlPackage::_Open(unsigned short const*, unsigned int, OpcPackageIOType) () at /opt/kingsoft/wps-office/office6/
#12 0x00007fffec5d6a04 in WordProcessDocument::Open(unsigned short const*, WordProcessDocument::DocumentType) () at /opt/kingsoft/wps-office/o
#13 0x00007fffe7dcc120 in  () at /opt/kingsoft/wps-office/office6/libwpsio.so
#14 0x00007fffe7d7aaf8 in filterpluginFormatCorrect_docx () at /opt/kingsoft/wps-office/office6/libwpsio.so
#15 0x00007ffff357cc3b in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#16 0x00007ffff358815e in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#17 0x00007ffff35a4cca in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
#18 0x00007ffff358d04b in  () at /opt/kingsoft/wps-office/office6/libwpsmain.so
```

Lets Fuzz WPS

Related modules for processing doc files

```
"coverage_module_name": [
  "libkso.so",
  "libwpsapi.so",
  "libwpsmain.so",
  "wps"
],
```

Linux

```
"coverage_module_name": [
    "wps.exe",
    "wpsmain.dll",
    "kso.dll",
    "ksoapi.dll"
],
```

Windows

Failure and plan

WPS **can’t** execute within python-trace!

I decided to develop the trace module based on gdb.

The reasons are as follows:

1. Stable, few bugs, support multiple platforms and architectures
2. Support develop plugin with python
3. Open source, can be customized on demand

Implementation

(Version 0.2 - GdbPythonPluginTracer)

GDB Python API

Introduces some commonly used API of GDB Python Plugin

gdb.selected_frame().read_register(): read register value

gdb.selected_inferior().write_memory(): write process memory

gdb.selected_inferior().read_memory(): read process memory

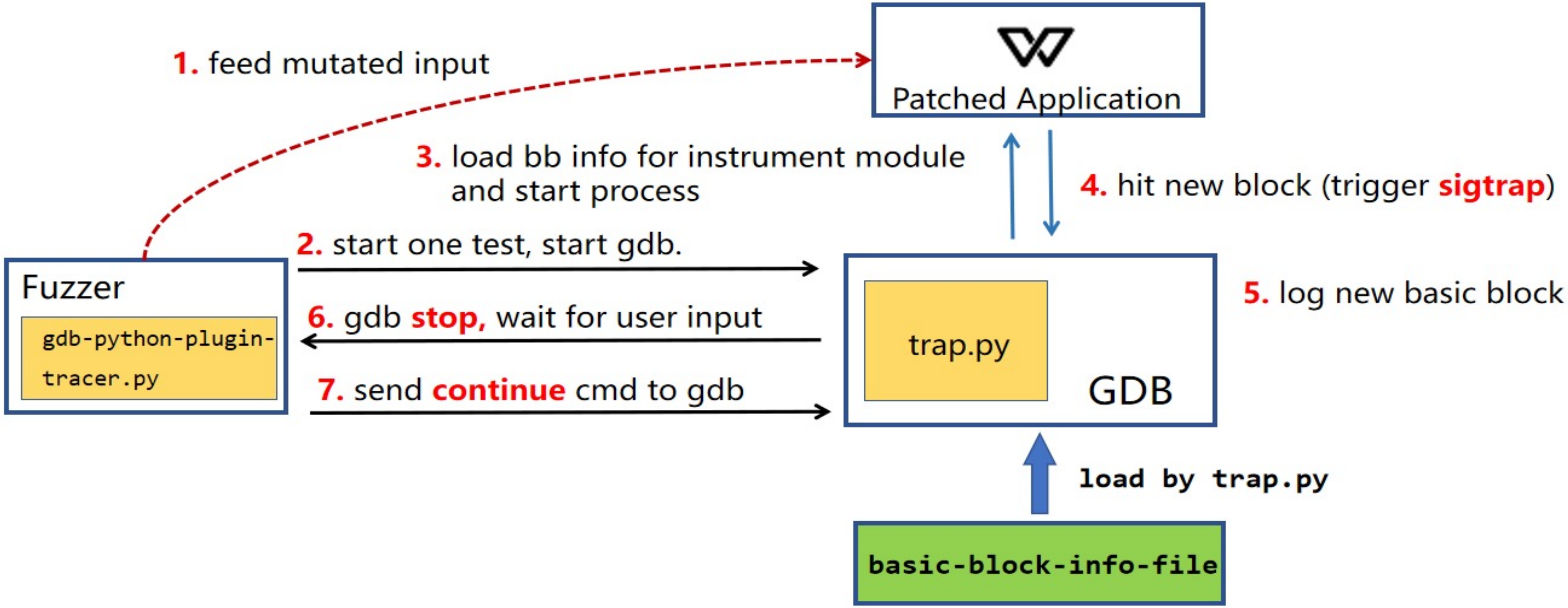
gdb.parse_and_eval(): execute gdb expression and get the result of expression

gdb.execute(): execute gdb command and get the result of command execution

gdb.events.stop.connect: register gdb's stop event callback function, for example, **when the process triggers a signal, it will enter the callback function for processing**.

Workflow

Due to the limitation of the gdb python script, the fuzzer needs to continuously use stdin and stdout to interact with gdb during the test.



Code – GDB Plugin

trap.py

```
def stop_handler(event):
    elif isinstance(event, gdb.StopEvent):
        pc = get_register("$pc") - 1
        hit_mod = None
        for mt in module_trace_list:
            if pc > mt['image_base'] and pc < mt['image_end']:
                hit_mod = mt
                break

        offset = pc - hit_mod['image_base']
        raw_byte = get_raw_byte_by_offset(offset)

        # restore original instructions, then set pc pc pc - 1
        write_memory(pc, raw_byte, 1)
        set_register("pc", pc)

        # log executed basic block
        mt['bbl-list'].append(offset)
    else:
        print("Unknown event {}".format(event))

# register callback
gdb.events.stop.connect(stop_handler)
```

Code – Tracer Part

gdb-python-plugin-tracer.py

```
def exec_with_gdb(self, timeout=30):
    command = "/usr/bin/gdb -q -x {}/cmd.gdb  --args {}".format(self.workspace, self.cmdline)
    self.p = subprocess.Popen(command, shell=True, cwd=self.workspace, stdin=subprocess.PIPE,
                              stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
    # for dos
    timer = Timer(timeout, self.timeout_handler)
    try:
        timer.start()
        while True:
            l = self.p.stdout.readline()
            # process hit breakpoint
            if "received signal SIGTRAP" in l:
                self.p.stdin.write("c\n")
            # test finished
            elif "[trapfuzzer] save_bb_trace" in l:
                break
    except Exception as e:
        pass
    finally:
        timer.cancel()

    self.p.kill()
    self.p.wait()
```

When to kill target process

Problem & Common Solution

WPS will not exit the program after parsing the file, but will stay on the GUI interface and wait for the user to operate, so the **fuzzer needs to manually kill the process.**

The commonly used method is to set a timeout, when the timeout occurs, the fuzzer will kill the process.

Disadvantage

1. The execution time of the program is fixed each time. It will **waste time** when process simple testcase (actually parsing time is less than timeout), and when process complex testcase (actually parsing time is greater than timeout), it will **result in an incomplete file parsing.**
2. Unable to detect **DOS vulnerabilities**.

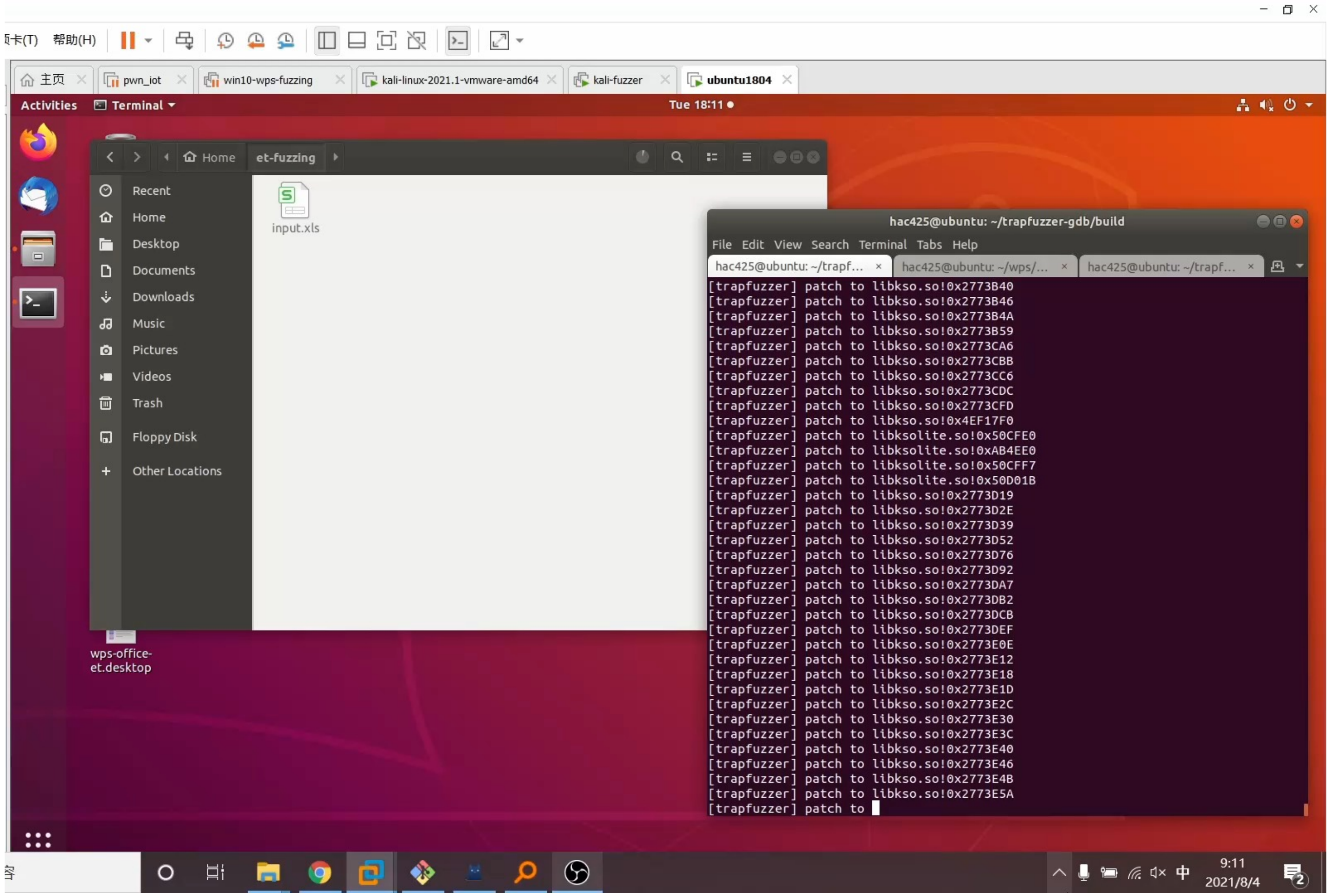
When to kill target process – our solution

1. First, use the trace module to trace the program execution and print the basic blocks executed by the program, and find the last basic block END_BBL executed after the process has parsed the input file.
2. In the following fuzzing, when the process reaches END_BBL, it means that the process has entered the GUI loop, and the process can be killed at this time.

During the fuzzing, record the average time (avg_time) of each testcase, and then set the DOS timeout to 10* avg_time.

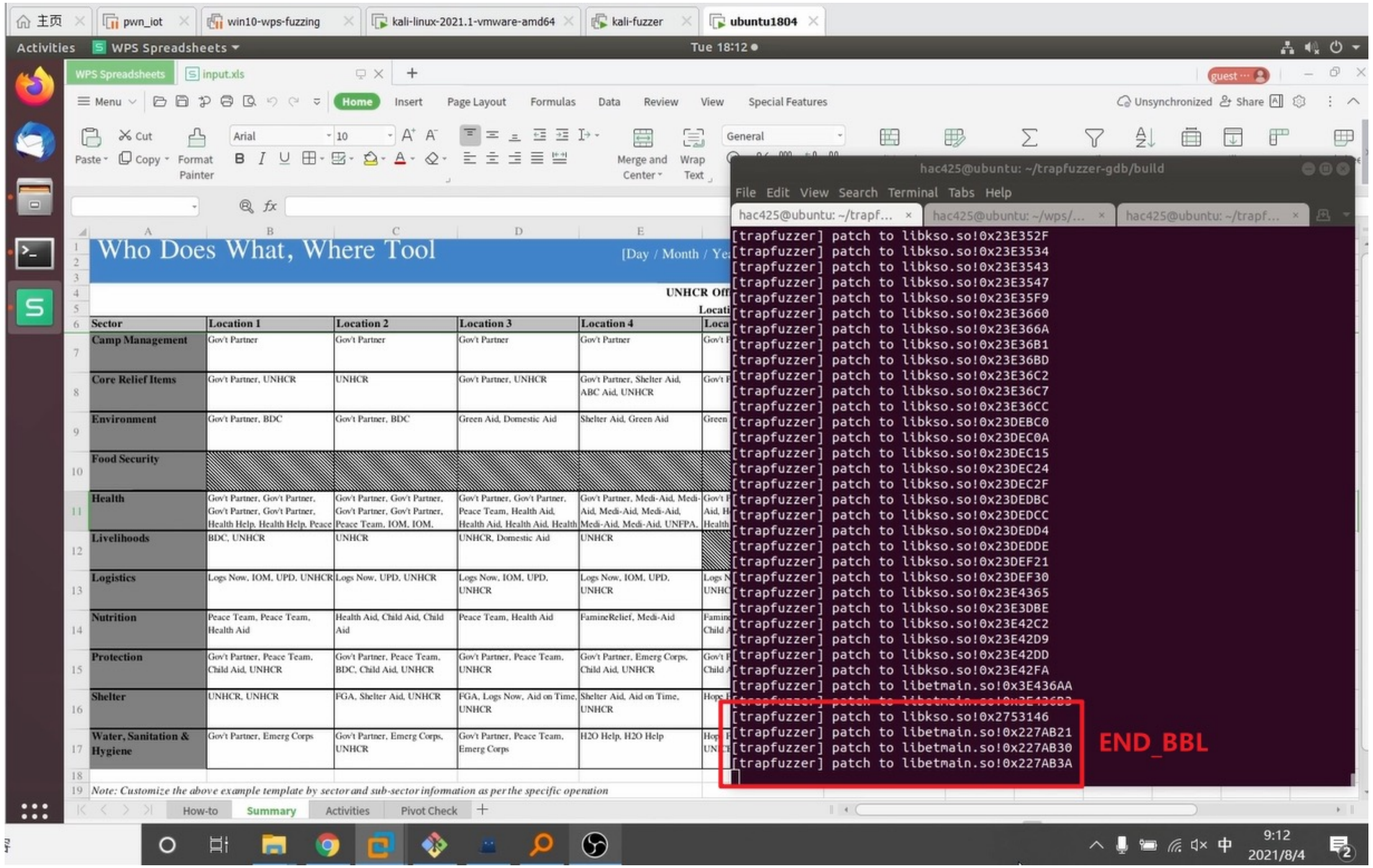
When the execution time is greater than timeout, it is considered that DOS bug has been found.

When to kill target process



HITB SECCONF SIN-2021

When to kill target process



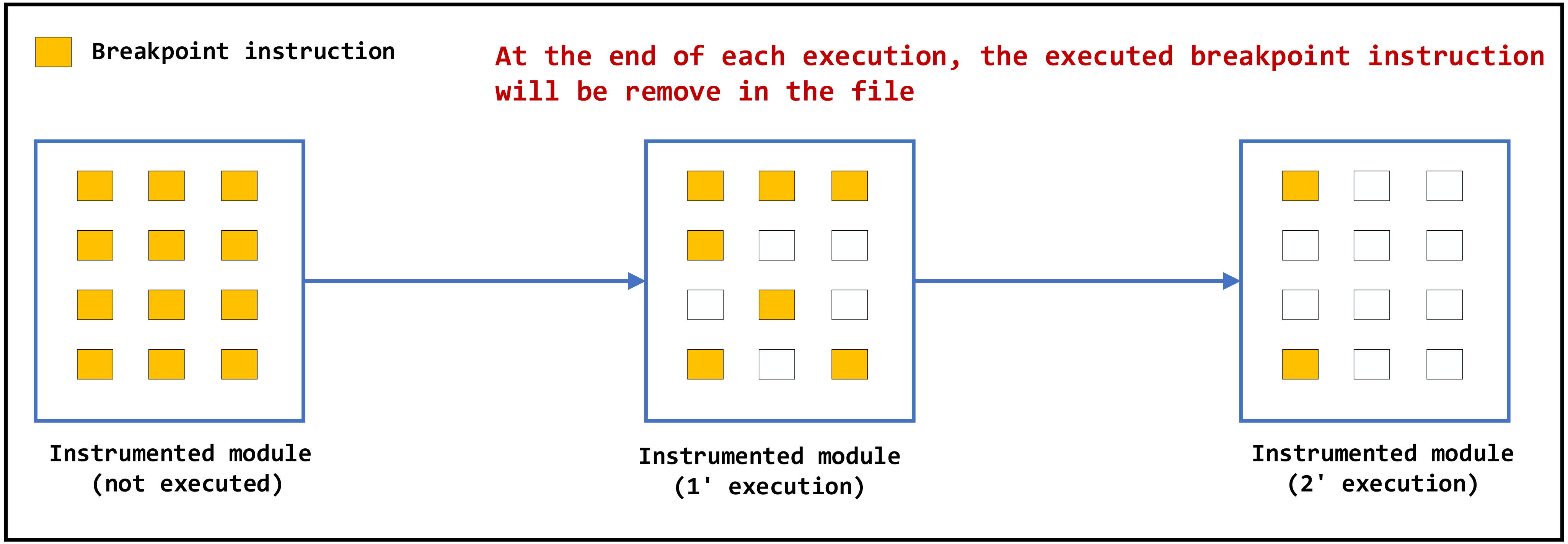
Speed up Instrument

For large programs, because the number of breakpoints is very large, it will take a lot of time for each execution.

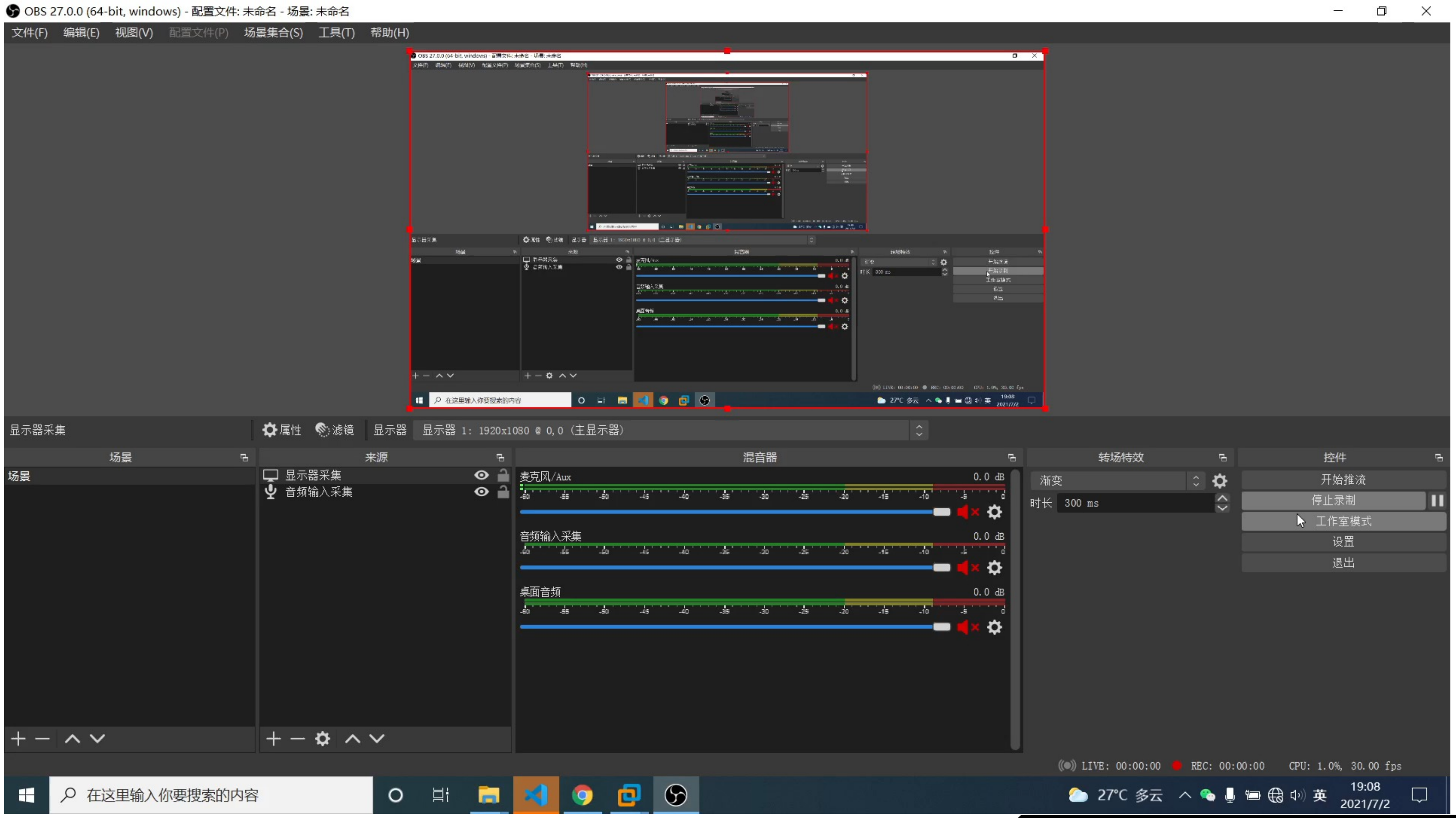
Program	desc	basic block count
WPP For Windows	Read and parse PPT file	275 1604
WPP For Linux	Read and parse PPT file	401 2478
Ichitaro 2021 Platinum	Read and parse doc、xls and so on	167 3576

Speed up Instrument - accelerated mode

In the accelerated mode, the fuzzer first obtains the executed basic blocks from the trace module, then patch the files related to these basic blocks and remove the breakpoint instructions at the corresponding positions in the files.



Lets Fuzz WPS Again!



Initial Results

Mutator	Crash Count	Time
RadamsaMutator	0	3 * 24h
TinyMutator	10+	24h

Reason

- RadamsaMutator don’t **consider the ratio of data mutation**, it may **seriously destroy** the file structure, causing the testcase to be discarded very early, so that deep vulnerabilities cannot be found.
- TinyMutator can only **mutates a small percentage of the data** in the sample file, so the use cases can enter a deeper code path and and the fuzzing is more efficient.

Implementation

(Version 0.3-trapfuzzer-gdb-tracer)

Why we need trapfuzzer-gdb-tracer

GdbPythonPluginTracer have some limits, such as:

1. It is inconvenient to debug, we need to use a python script to continuously send continue commands to the stdin of gdb.
2. We need to restart gdb for each test, the python runtime, and the communication overhead of fuzzer will cause additional performance overhead.

trapfuzzer-gdb-tracer has following advantages:

1. Written in C++, faster.
2. We can compile static-link GDB to reduce the requirements of environment.
3. Easy to debug.

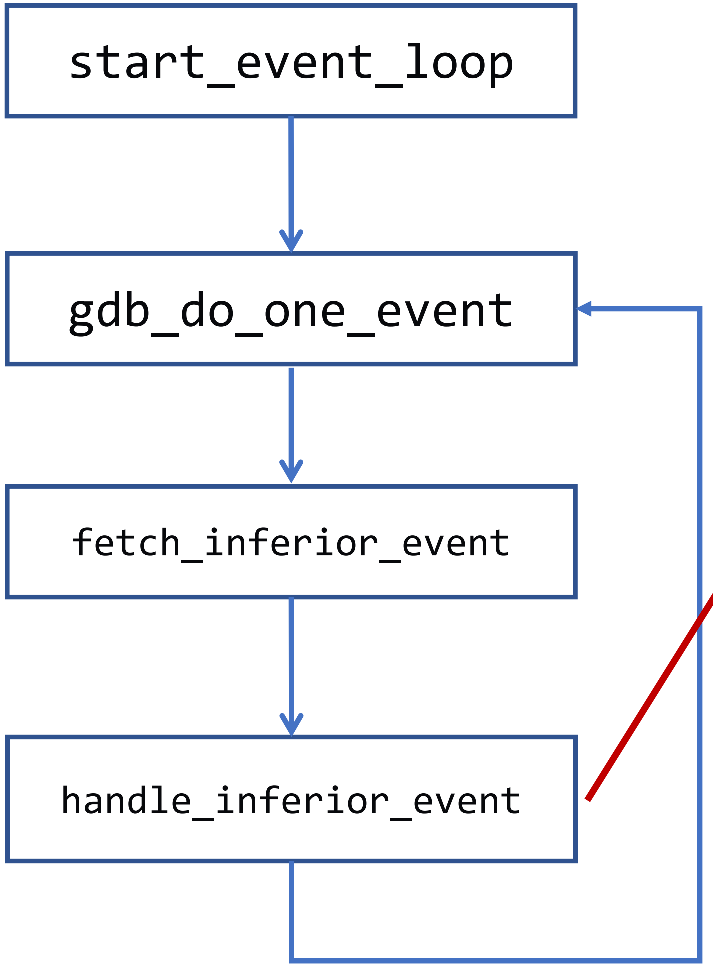
GDB Internals

On the Linux platform, gdb use ptrace to debug process.

When gdb starts the debugged program, it will call **start_event_loop** to start the event loop and wait for events from the target process, such as **hitting a breakpoint** , creating a child process, and so on.

```
start_event_loop () at ../../gdb/event-loop.c:370
captured_command_loop () at ../../gdb/main.c:359
captured_main at ../../gdb/main.c:1202
gdb_main  at ../../gdb/main.c:1217
main  at ../../gdb/gdb.c:32
```

GDB Internals




```
static void
handle_inferior_event(struct execution_control_state *ecs)
{
  switch (ecs->ws.kind)
  {
  case TARGET_WAITKIND_LOADED:
    ............
    ............

  case TARGET_WAITKIND_SPURIOUS:
    ............
    ............
  ...............

  case TARGET_WAITKIND_STOPPED:
    // handle signal event of target process, such as SIGTRAP
    handle_signal_stop(ecs);
    return;

```

GDB Internals

handle_signal_stop handles the situation where the debugged process stops due to receiving a signal,

```
static void
handle_signal_stop(struct execution_control_state *ecs)
{
  frame = get_current_frame();
  gdbarch = get_frame_arch(frame);

  /* Pull the single step breakpoints out of the target.  */
  if (ecs->event_thread->suspend.stop_signal == GDB_SIGNAL_TRAP)
  {
    // handle sigtrap signal

```

Modify GDB

Modify the handle_signal_stop function to let gdb automatically handle the breakpoint events of the process:

1. First it gets the value of the pc register, and then finds the module where the pc is located
2. Then according to pc and basic-block-info-file, get the original instruction of the position
3. Finally, replace the breakpoint instruction with the original instruction and let the process continue execution

Modify GDB

We can use add_com to add custom gdb commands

```
c = add_com ("load-trapfuzzer-info", class_run, load_trapfuzzer_info, _("Load trapfuzzer config.\n"
RUN_ARGS_HELP));
set_cmd_completer (c, filename_completer);
```

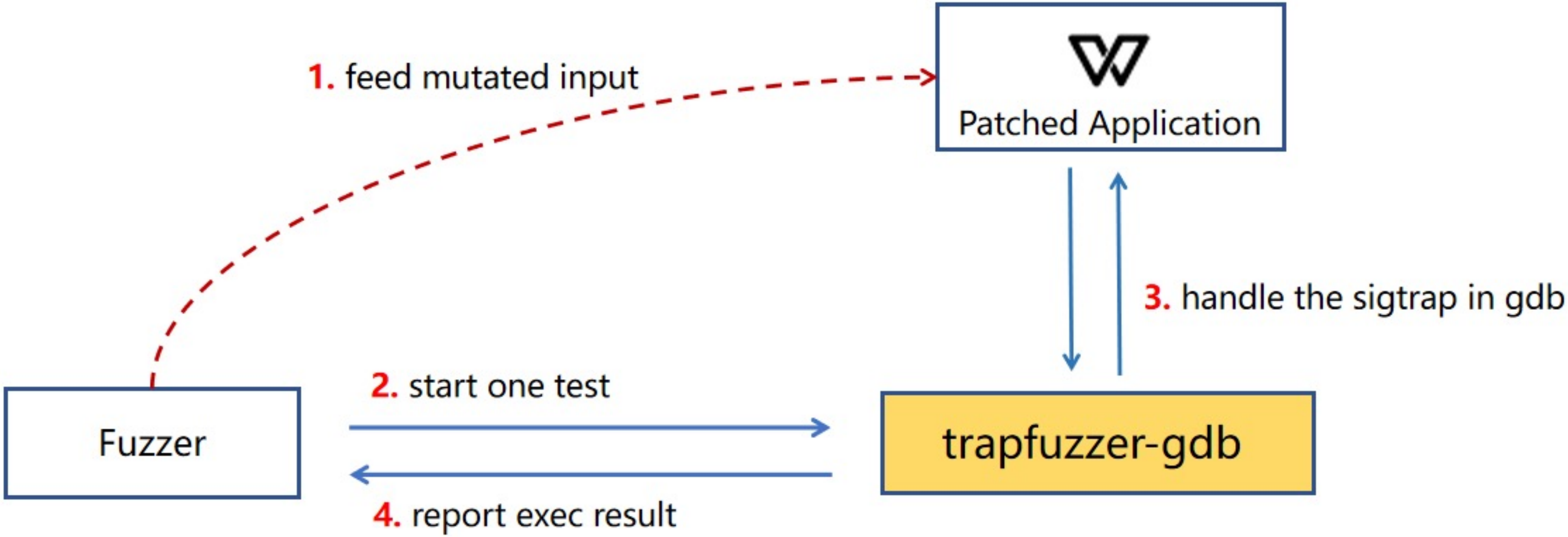
Use execute_command_to_string to execute gdb command

```
std::string get_context_string()
{
    std::string cmd_res = execute_command_to_string("i r", 0,  false);
    cmd_res += execute_command_to_string("x/4i $pc", 0,  false);
    cmd_res += execute_command_to_string("bt 4", 0,  false);
    return cmd_res;
}
```

Code for SIGTRAP

```
// handle sigtrap event.
if (ecs->event_thread->suspend.stop_signal == GDB_SIGNAL_TRAP)
  {
      pc = regcache_read_pc (regcache);
      // find module of pc
      COV_MOD_INFO* cmi = get_cov_mod_info_by_pc(pc);
      // get module offset
      unsigned int voff = pc - cmi->image_base;
      // remove breakpoint
      BB_INFO* info = cmi->bb_info_map[voff];
      target_write_memory(pc, info->instr, info->instr_size);

      if(g_debug)
          fprintf_unfiltered (gdb_stdlog, "[trapfuzzer] patch to %s!0x%X\n", cmi->module_name, voff);
      // log executed basic block
      cmi->bb_trace.push_back(voff);
      // let process continue.
      keep_going (ecs);
      return;
```

Architecture



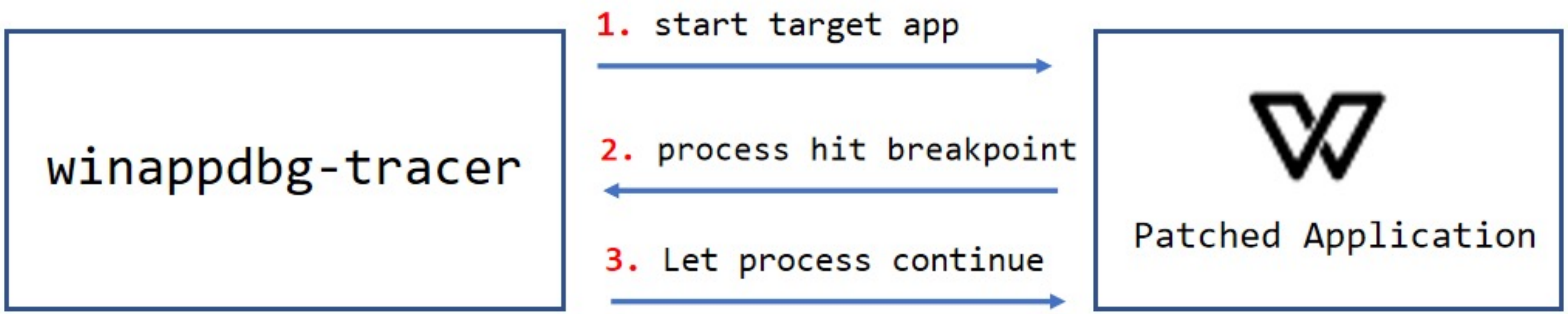
Implementation

(Version 0.4-Windows Support)

Windows Support #1 - winappdbg-tracer

Based on winappdbg

1. based on python.
2. winappdbg basically meets the demand, but there are still some shortcomings: **speed, incomplete access to the call stack.**



Windows Support #2 – DbgEngTracer

The DbgEng API is a series of APIs for developing debuggers provided by Microsoft. Users only need to register the corresponding callback function to implement a debugger.

Advantage:

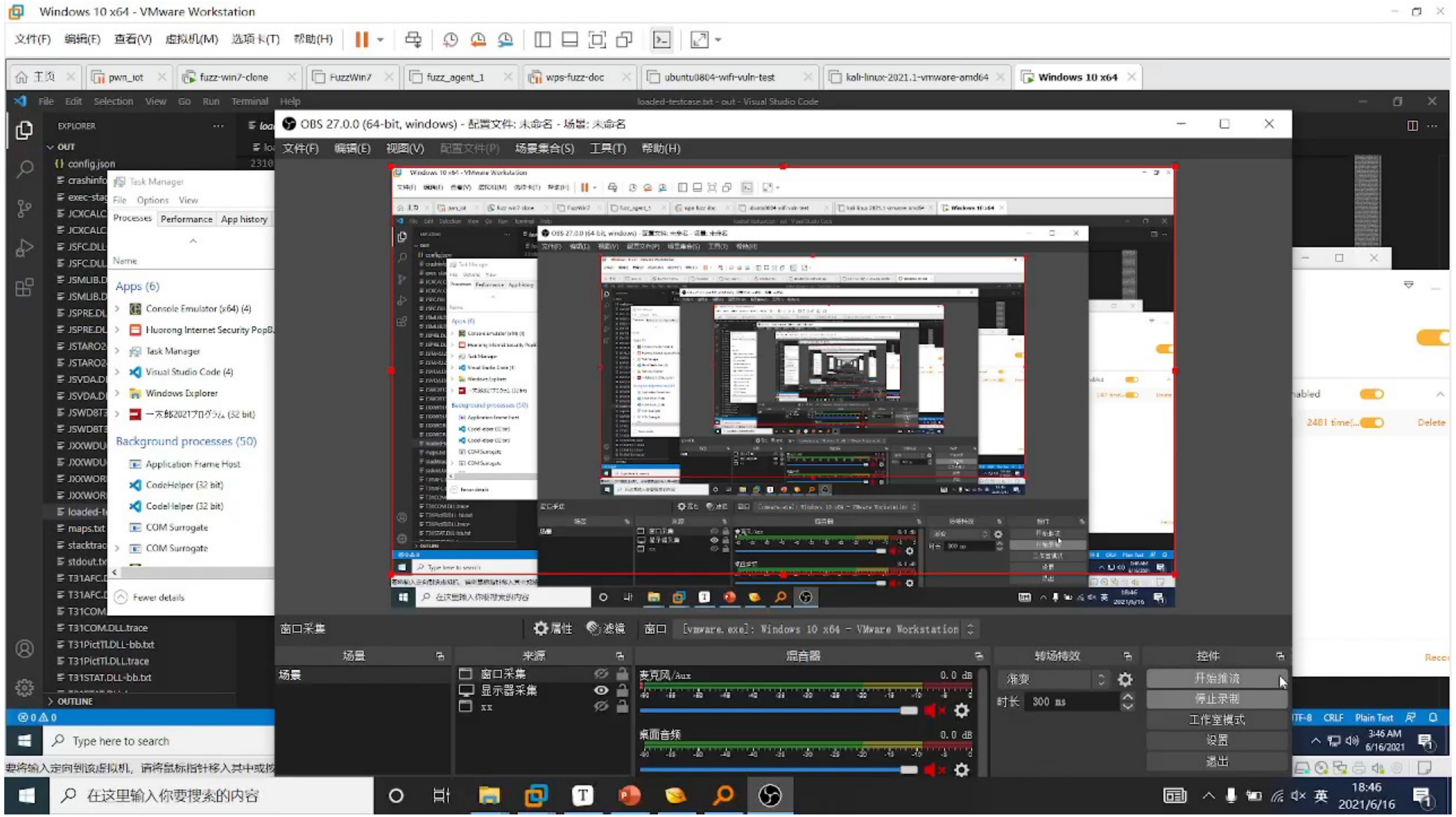
1. Fast execution speed.
2. DbgEng API can get the complete call stack of the program.

Windows Support #2 – DbgEngTracer

1. **WriteVirtual/ReadVirtual**: Read/Write process memory.
2. **GetStackTrace**: Get backtrace of process.

```
EventCallbacks::Exception()
{
    if (Exception->ExceptionCode == STATUS_BREAKPOINT)
    {
        COV_MOD_INFO *cmi = get_cov_mod_info_by_pc(Exception->ExceptionAddress);
        if (cmi != NULL)
        {
            BB_INFO *bi = cmi->bb_info_map[Exception->ExceptionAddress - cmi->image_base];
            printf("exec-bb: %s!0x%lx\n", cmi->module_name, bi->voff);

            cmi->bb_trace.push_back(bi->voff);
            if (g_Data->WriteVirtual(Exception->ExceptionAddress, bi->instr,
                                     bi->instr_size, &Done) != S_OK || Done != bi->instr_size)
            {
                return DEBUG_STATUS_NO_CHANGE;
            }
        }
        return DEBUG_STATUS_GO;
    }
```

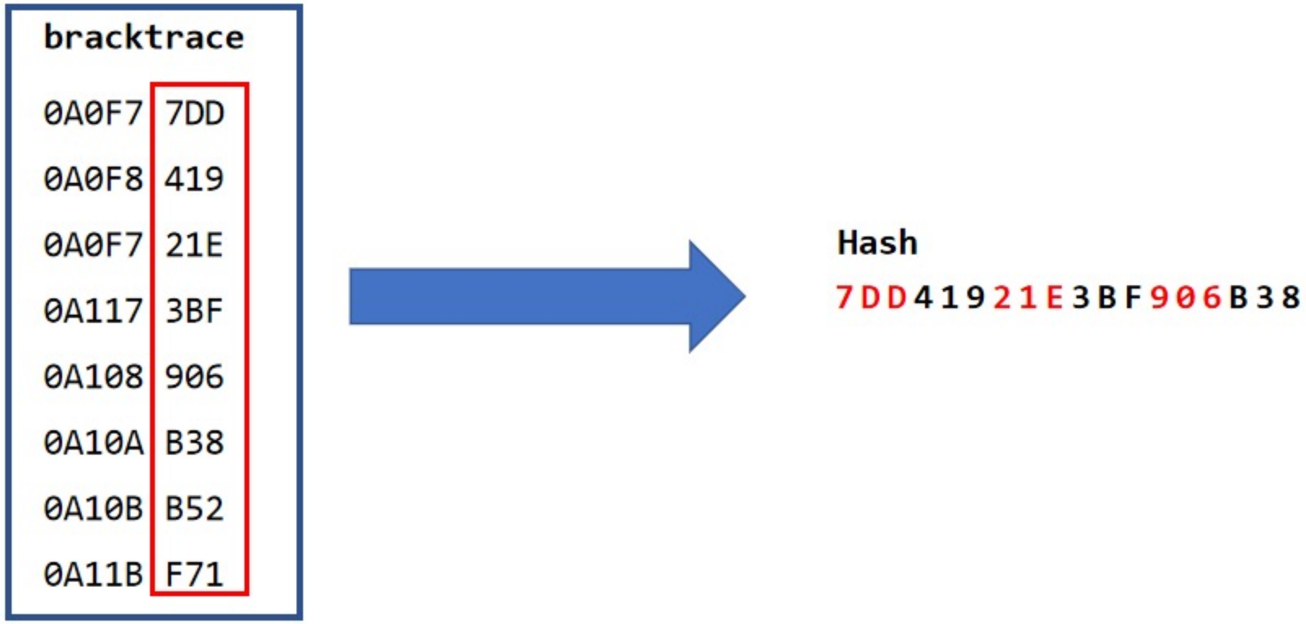
Example


Windows 10 x64 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
pwn_iot
fuzz-win7-clone
FuzzWin7
fuzz_agent_1
wps-fuzz-doc
ubuntu0804-wifi-vuln-test
kali-linux-2021.1-vmware-amd64
Windows 10 x64
loaded-testcase.txt - out - Visual Studio Code
OBS 27.0.0 (64-bit, windows) - 配置文件: 未命名 - 场景: 未命名
文件(F) 编辑(E) 视图(V) 配置文件(P) 场景集合(S) 工具(T) 帮助(H)
Task Manager
Processes Performance App history
Apps (6)
Console Emulator (x64) (4)
Huorong Internet Security PopB
Task Manager
Visual Studio Code (4)
Windows Explorer
Background processes (50)
Application Frame Host
CodeHelper (32 bit)
COM Surrogate
Fewer details
窗口采集
场景
来源
混音器
转场特效
控件
麦克风/Aux
桌面音频
开始推流
停止录制
工作室模式
设置
退出
Type here to search
在这里输入你要搜索的内容
18:46
2021/6/16

Triage

crash deduplication scheme:

1. first get the call stack of crash
2. then splice the lower 12 bits of each call stack as the hash of crash
3. de-duplicate crash according to hash



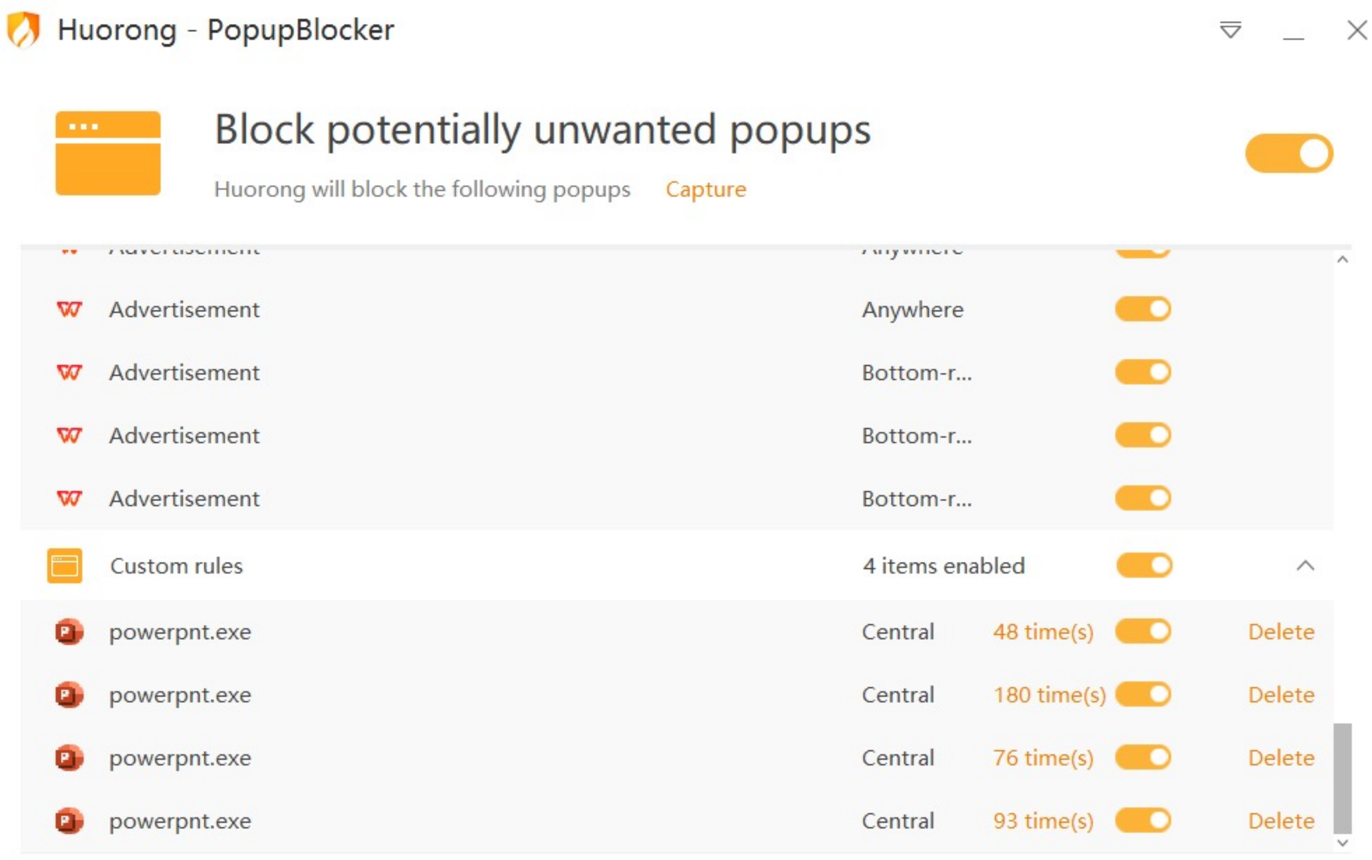
Dialog Box

automatically handle dialog box with autoit

```python
def POPUpKillerThread(self):
    while True:
        time.sleep(0.1)

        ppt_text = autoit.win_get_text('Microsoft PowerPoint')

        if u"出现严重错误" in ppt_text:
            autoit.control_click("[Class:#32770]", "Button1")

        if u"无法编辑此" in ppt_text:
            autoit.control_click("[Class:#32770]", "Button1")

        if u"密码" in ppt_text:
            autoit.control_click("[Class:#32770]", "Button1")
```

Dialog Box

automatically handle dialog box with [huorong]()



Other Features

Users can check the status of Fuzz through the management port during the fuzzing process

```
status
stage: fuzz
total dos:      59
total crash:    126
total new path: 878
speed: 18.0/min
trace mode: gdb-run
stage exec count:654
total bb count:  0
total exec time: 00 days 00h 39m 55s
coverage module: libkso.so,libwpsmain.so,libwpscloudsvrimp.so,libwpshtmlrw.so,libwpsinkdraw.so,libwpsio.so,libwpsmain.so,libwpstablestyle.so,libwpsuofrw.so,libwpswordtool.so,libwpsxmlrw.so,wp
config mutator:  Vanapagan-mutator
current mutator: VanapaganMutator
last crash:  2021-06-02 20:14:13
last new path:  2021-06-02 19:50:01
current time:  2021-06-02 20:14:18
avg_run_time:  3.7s, timeout ratio: 5
output: /home/hac425/ndisk/doc-output
current seed file: /home/hac425/ndisk/doc-output/trapfuzz-testcase-476.bin

h
status (s)
seed_info (si)
set_seed_exec_count (sc)
enable_debug (eb)
disable_debug (db)stop
testcase (t)
crash (c)
timeout_ratio (tr)
import_case dir
quit (q)
```

nc 127.0.0.1 8821

Start Fuzzing

Three elements of fuzzing

execution speed, seed quality/quantity, mutation algorithm

Ways to obtain samples

1. Obtained from some online sites that provide sample sets
2. **Crawl a large number of sample files through the grammar of the search engine**
3. Some open source projects will bring some testcases to test the program
4. **Testcases generated when using white box Fuzz tools such as AFL to test similar software**
5. The bug submission page of the target program or similar program
6. Generated with format conversion tool

Preparing the Environment

Ways for faster execution

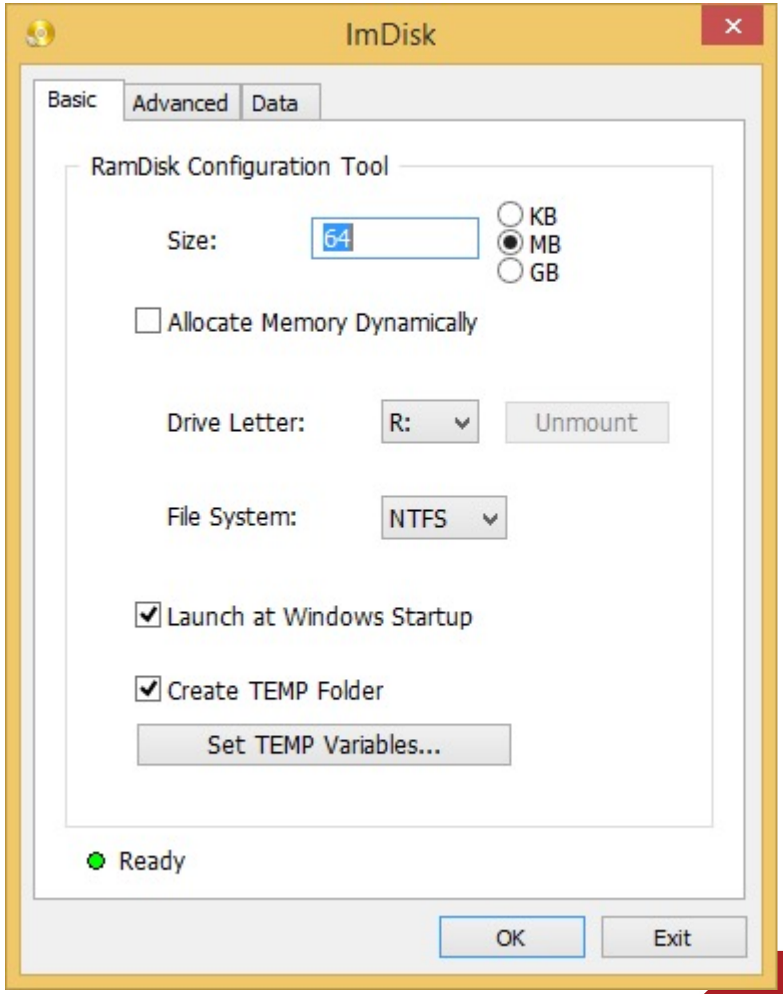
use ramdisk, but pay attention to scheduled backups!

Ramdisk in Linux

```
mount -t tmpfs -o size=4G tmpfs /tmp/ramdisk/
```

Ramdisk in Windows

ImDisk Toolkit



Equipment & Results

laptop with the following configuration:
i5-8700 (4 Cores 8 threads) / 16G DDR3 RAM / 1T SSD

Software	Platform	Time	Initial Seeds	Bug Found
WPS OFFICE	Linux & Windows	≈ 8 month	Crawl	200+
中望CAD	Linux	≈ 1 day	AFL (fuzz libredwg)	4+
WPS Photo	Windows	≈ 1 day	AFL (fuzz libpng)	15
Foxit PDF	Windows	≈ 7 day	Crawl	a few crashes
Honeyview	Windows	≈ 1 day	AFL (fuzz libpng)	a few crashes
Ichitaro 2021	Windows	≈ 7 day	trapfuzzer (fuzz WPS)	100+ unique crash.

Compare with existing tools

Tool	Scenarios	Advantage	Disadvantage
AFL	Library, small program	Coverage, Speed	Large software requires wrapper and high difficulty to user
Peach	protocol fuzzing, file fuzzing	easy to use after the model is written	model development is difficult
trapfuzzer	file fuzzing now	**easy to use, support large software, support coverage！**	Compared with AFL, there is **less feedback coverage and speed is slow.**

Conclusion

1. Seed is important and mutate ratio is also important
2. GDB and DbgEng API is very nice.
3. Fuzzing with coverage is much better than none

Future Plans

1. More architecture support (arm, mips)
2. Optimize distributed Fuzz scheduling
3. Automatic detect data mutation ratio and mutation range.
4. More precise crash deduplication mechanism
5. ……………………

Q&A

Thank you!

References

1. https://loda.hala01.com/2017/06/gdb.html
2. https://www.embecosm.com/appnotes/ean3/embecosm-howto-gdb-porting-ean3-issue-2.html