When virtualization encounters AFL

Jack Tang, @jacktang310
Moony Li, @Flyic

What We Will Cover

- Who We Are
- Approach
- Implementation
- Case Study
- Demo

Who We Are

Jack Tang

- @jacktang310
- 10+ years security
- Browser
- Document
- Mac/Windows Kernel
- Virtualization Vulnerability

Moony Li

- @Flyic
- 7 years security
- Sandcastle
- Deep Discovery
- Exploit Detection
- Mac/Windows Kernel
- Android Vulnerability

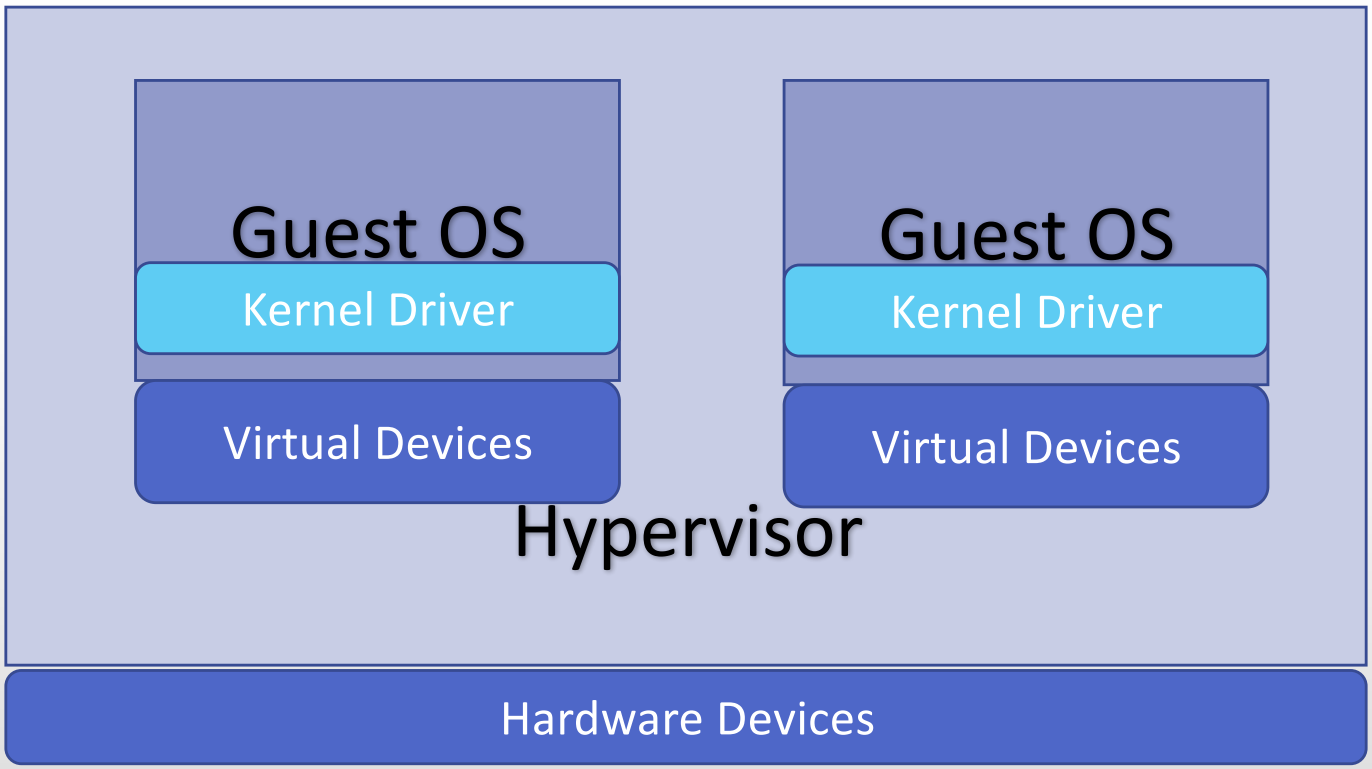
TREND MICRO

Approaches Comparison

Approach	Notes	Pros	Cons
Fuzz in guest OS	1.Capture 2.Replay & Fuzz	1.Simple 2.Ignorance of I/O protocol	1.No code coverage 2.Incomplete by design
Conformance fuzzing test	1.Symbol Execution	1.Fuzz deeper	1.No code coverage 2.Acdemical
Code Review		1.Fexible	1.Cost effort 2.No scalable

TREND MICRO

Target: Virtual Devices



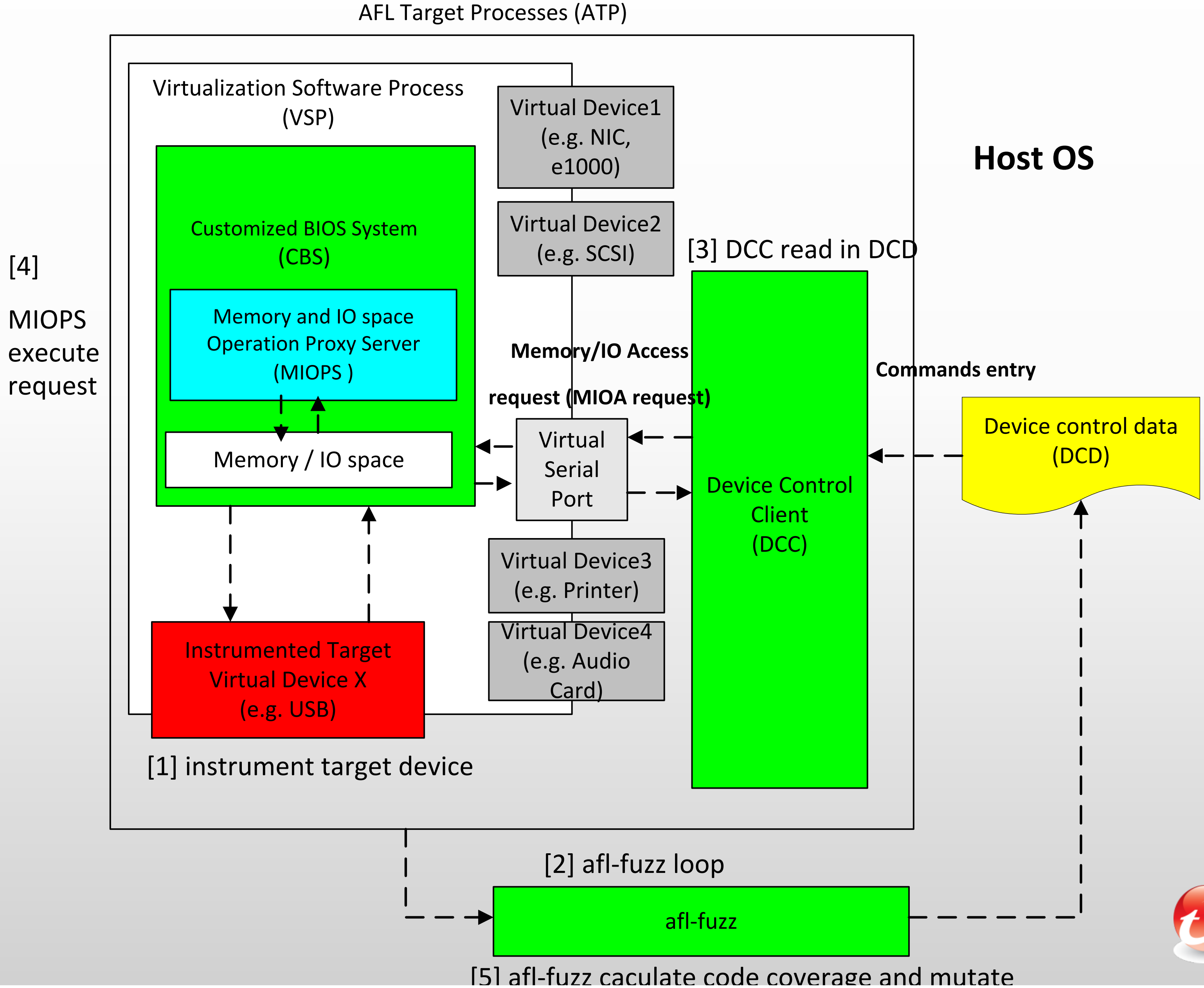
Our Approach

- Portable
  - Customized BIOS as common format Virtual Disk
  - Serial port
- Performance
  - sizeof(BIOS)<sizeof(OS).
- Direct
  - Bypass device driver layer
- Code coverage feedback & control
  - AFL integration

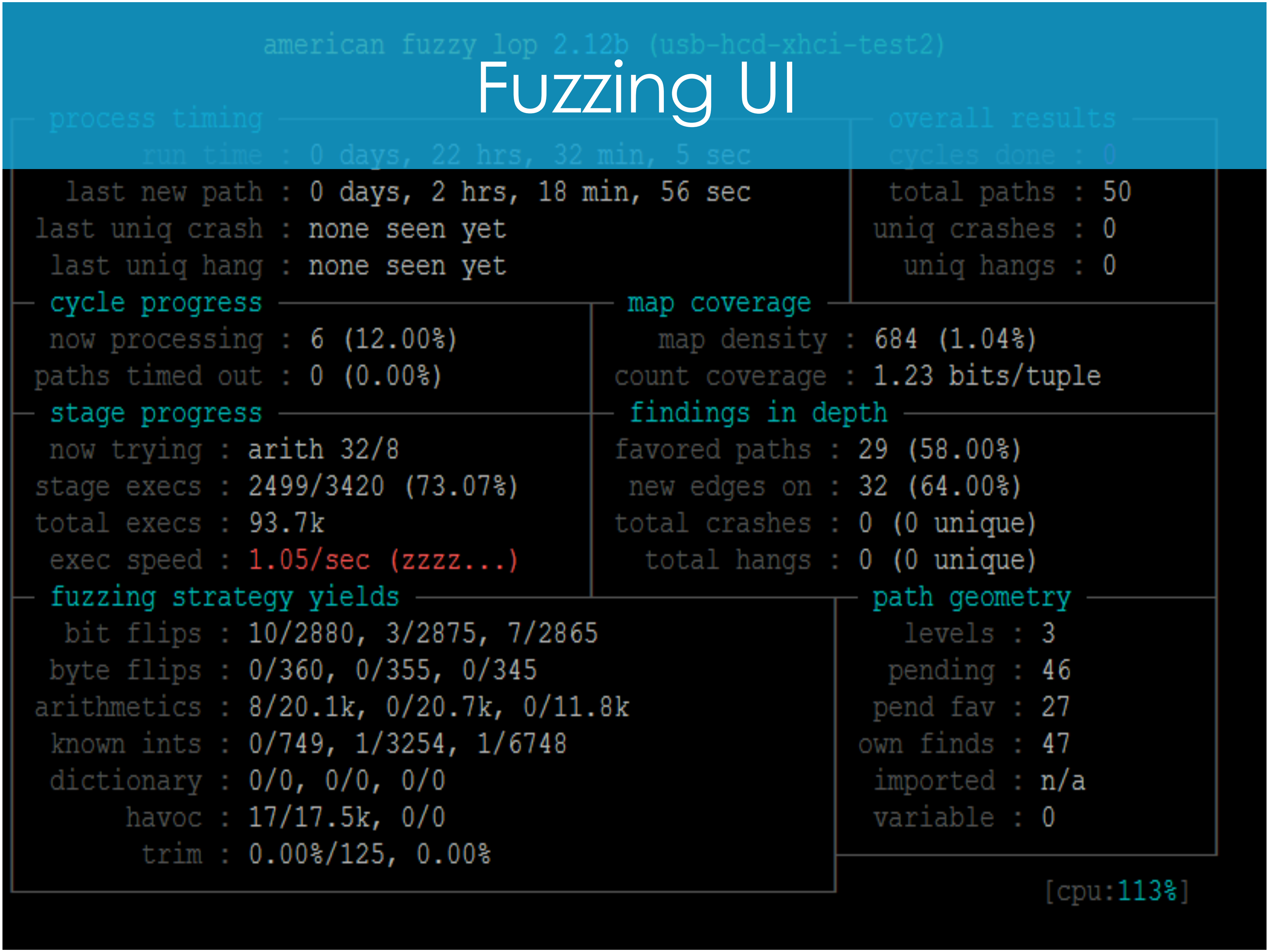
Implementation

- Architecture
- Modules
- Work Flow
- AFL Tips

Architecture Overview



Fuzzing UI


american fuzzy lop 2.12b (usb-hcd-xhci-test2)
process timing
run time : 0 days, 22 hrs, 32 min, 5 sec
last new path : 0 days, 2 hrs, 18 min, 56 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
overall results
cycles done : 0
total paths : 50
uniq crashes : 0
uniq hangs : 0
cycle progress
now processing : 6 (12.00%)
paths timed out : 0 (0.00%)
map coverage
map density : 684 (1.04%)
count coverage : 1.23 bits/tuple
stage progress
now trying : arith 32/8
stage execs : 2499/3420 (73.07%)
total execs : 93.7k
exec speed : 1.05/sec (zzzz...)
findings in depth
favored paths : 29 (58.00%)
new edges on : 32 (64.00%)
total crashes : 0 (0 unique)
total hangs : 0 (0 unique)
fuzzing strategy yields
bit flips : 10/2880, 3/2875, 7/2865
byte flips : 0/360, 0/355, 0/345
arithmetics : 8/20.1k, 0/20.7k, 0/11.8k
known ints : 0/749, 1/3254, 1/6748
dictionary : 0/0, 0/0, 0/0
havoc : 17/17.5k, 0/0
trim : 0.00%/125, 0.00%
path geometry
levels : 3
pending : 46
pend fav : 27
own finds : 47
imported : n/a
variable : 0
[cpu:113%]

Modules-CBS 1/3

- Customized BIOS System(Seabios)
  - Decision principle:
    - Portability
  - Contains MIOPS(Memory and IO space Operation Proxy Server)
  - VSP (Virtual Software Process) is its runtime instance
- Virtual serial port
  - Communication bridge

Modules- How To Customize CBS 2/3

- **Traditional BIOS vs Customized BIOS**

POST phase
(Power On Self Test)

Boot phase
(Boot Operating System)

BIOS runtime service phase
(BIOS service for runtime OS)

Modules- How To Customize CBS 3/3

POST phase
(Power On Self Test)

- Detect physical memory
- Platform hardware setup (for example: PCI bus, clock…)
- Necessary device hardware setup (for example: serial port device for communication)
- Recognize the devices
- Start to run MIOPS (Memory and IO space Operation Proxy Server) handle loop.

TREND MICRO

Modules-MIOPS 1/2

- Memory and IO space Operation Proxy Server
- MIOA Request- (Memory I/O Access) request
- Tips:
  - Remove interruption in CBS
  - MIOPS polls virtual device result of request with timeout

Modules-MIOPS 2/2

- **Job:**
  - **Receive MIOA**
  - **Execution it**

MIOA example:
“inb <address>”
“inw <address>”
“inl <address>”
“outb <address> <value>”
“outw <address> <value>”
“outl <address> <value>”
“write <address> <value> <length>”
“read <address> <length>”

```
struct request_MIOA* process_MIOA_request(char* request_string)
{
    struct request_MIOA* pReq = NULL;
    pReq = parse_MIOA_request(request_string);
    if(!pReq)
        return NULL;
    switch(pReq->command_id)
    {
        case e_inb:
            do_inb(pReq);
            break;
        case e_inw:
            do_inw(pReq);
            break;
        case e_inb:
            do_inl(pReq);
            break;
        case e_outb:
            do_outb(pReq);
            break;
        case e_outw:
            do_outw(pReq);
            break;
        case e_outl:
            do_outl(pReq);
            break;
        case e_read:
            do_read(pReq);
            break;
        case e_write:
            do_write(pReq);
            break;
        default:
            return NULL;
    } ?  end switch pReq- >command_id ?
    return pReq;
} ?  end process_MIOA_request ?
```

Modules-DCC 1/2

- **Device Control Client**
- **DCD – Device Control Data**
  - **Formatted suitable for AFL**
- **Steps in Job:**
  - **1. Launch VSP to load CBS**

    mkfifo jack_pipe
    mkfifo jack_pipe1
    qemu-system-x86_64 -bios out/bios.bin -serial pipe:jack_pipe -serial pipe:jack_pipe1

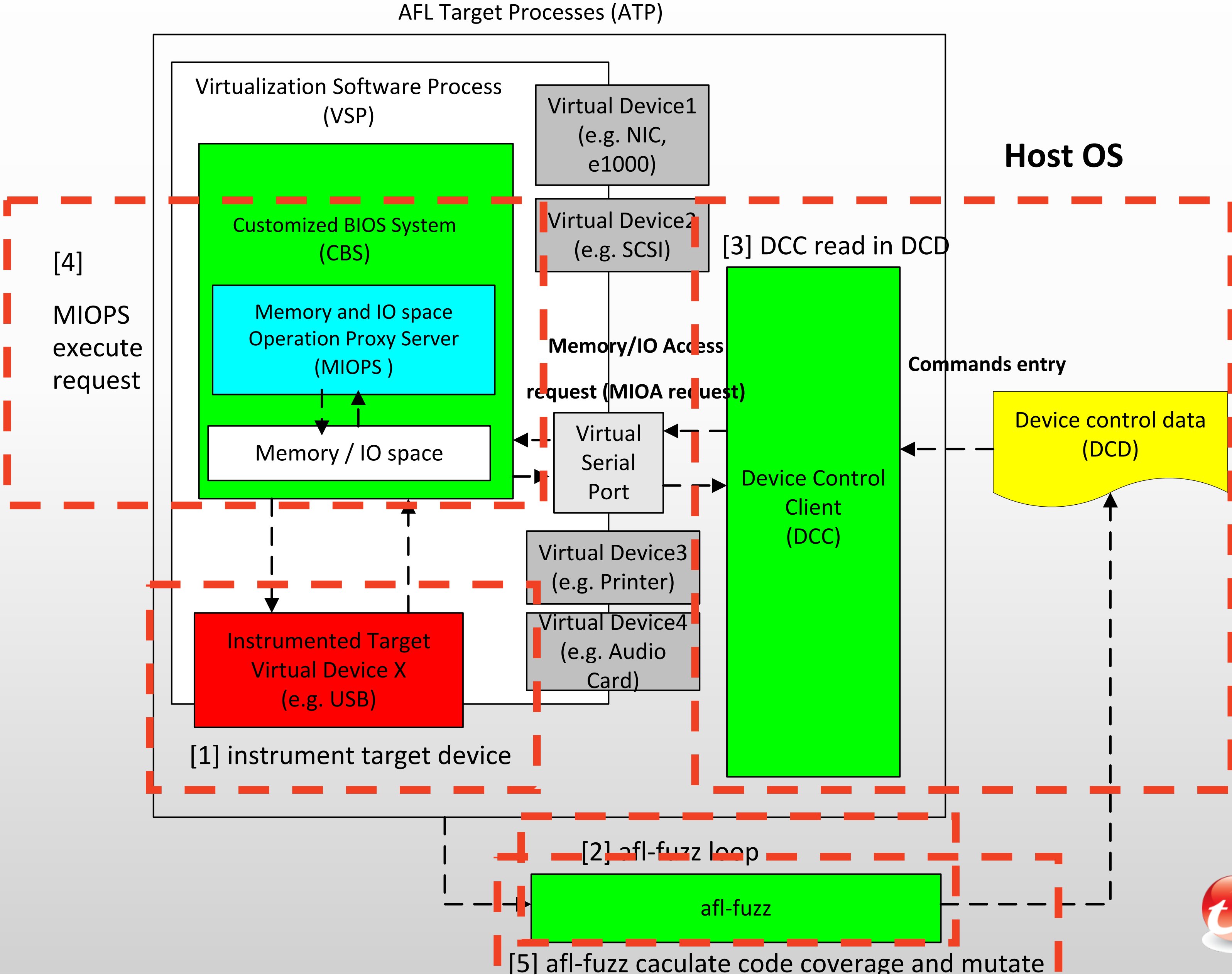
  - **2. Ping MIOPS in CBS**

Modules-DCC 2/2

- **Steps in Job:**
  - **3. Initialize target virtual dev**
    - **e.g. USB XHCI device initializa**
  - **4. Parse DCD to MIOA**

```
outl 0xcf8 0x8000e800
ECHO
inw 0xcfc
ECHO 0x1033
outl 0xcf8 0x8000e800
ECHO
inl 0xcfc
ECHO 0x1941033
outl 0xcf8 0x8000e804
ECHO
inw 0xcfc
ECHO 0x0000
outl 0xcf8 0x8000e804
ECHO
outw 0xcfc 0x7
ECHO
outl 0xcf8 0x8000e804
ECHO
inw 0xcfc
ECHO 0x0007
outl 0xcf8 0x8000e810
ECHO
outl 0xcfc 0xffffffff
ECHO
outl 0xcf8 0x8000e810
ECHO
inl 0xcfc
ECHO 0xffffc004
outl 0xcf8 0x8000e810
ECHO
outl 0xcfc 0xe0000000
ECHO
read 0xe0000000 1
ECHO 0x0000000000000040
read 0xe0000004 4
ECHO 0x0000000008001040
```

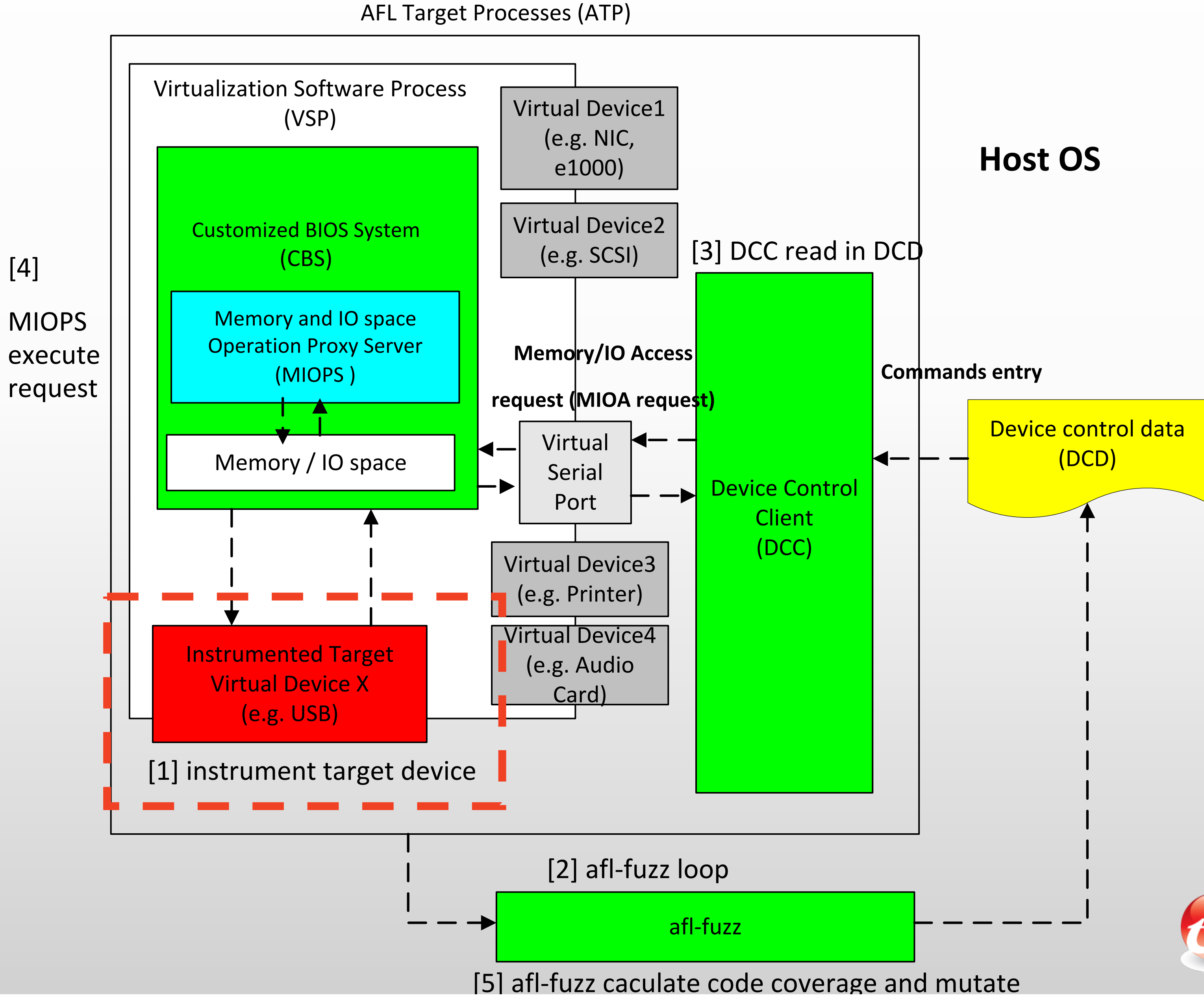
Workflow Overview



Work Flow-
Setup instrumentation-Step1/4

- **Step1:Instrument the target devices**
  - **source code is available**
    - **Afl-gcc compile source code in part**
  - **source code NOT available**
    - **Instrument executable file statically + restricted instrumentation range**

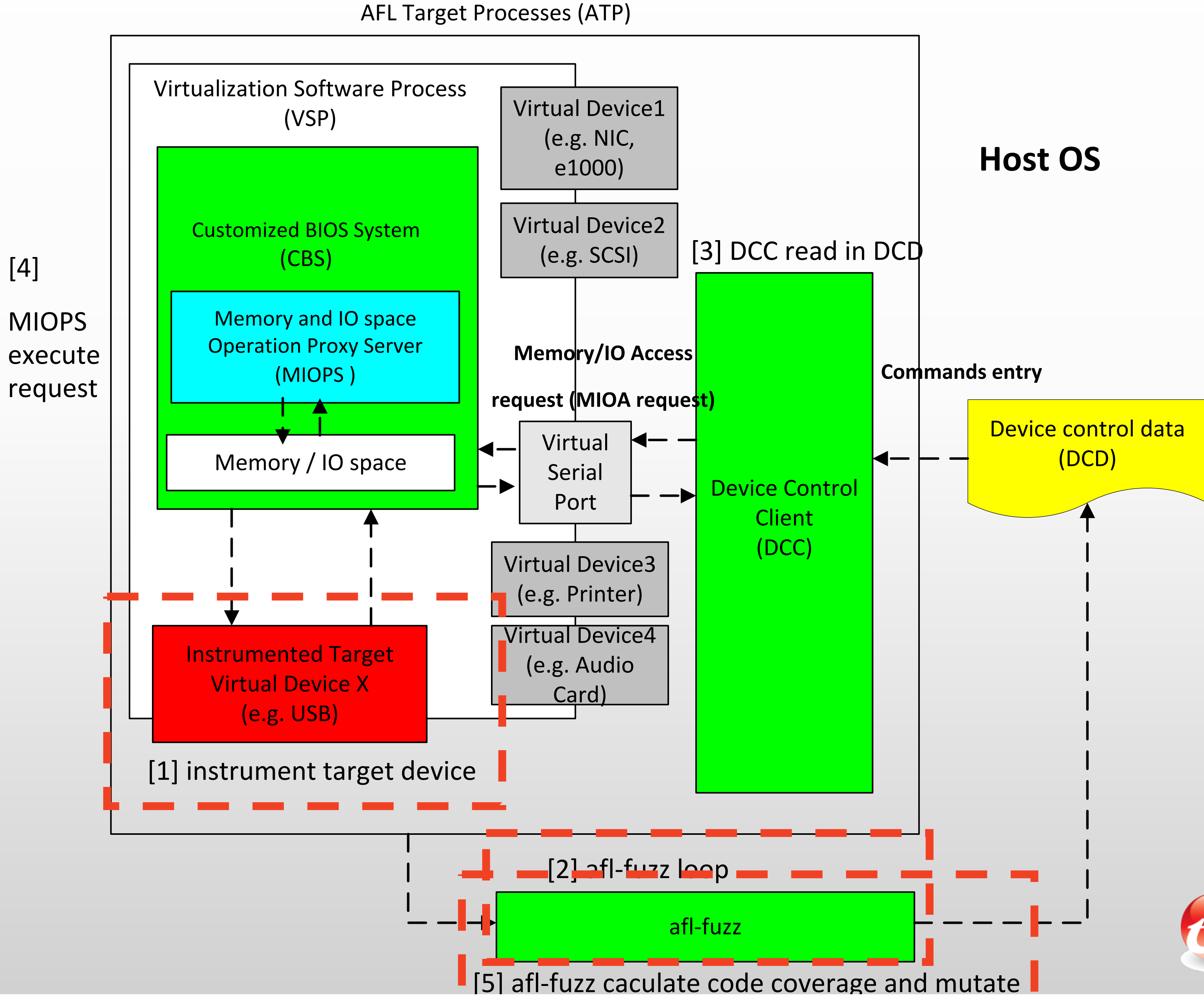
Workflow Overview- Step 1



Work Flow-
Setup Trace & Feedback – Step 2/4

- **Step2:Launch Afl-fuzz loop**
  - **afl-fuzz -t 90 -m 2048 -i ${test_root}/IN/ -o ${test_root}/OUT/ ${DCC} @@**
  - **Afl-fuzll will automatically mutate DCD and calculate code coverage in the loop**

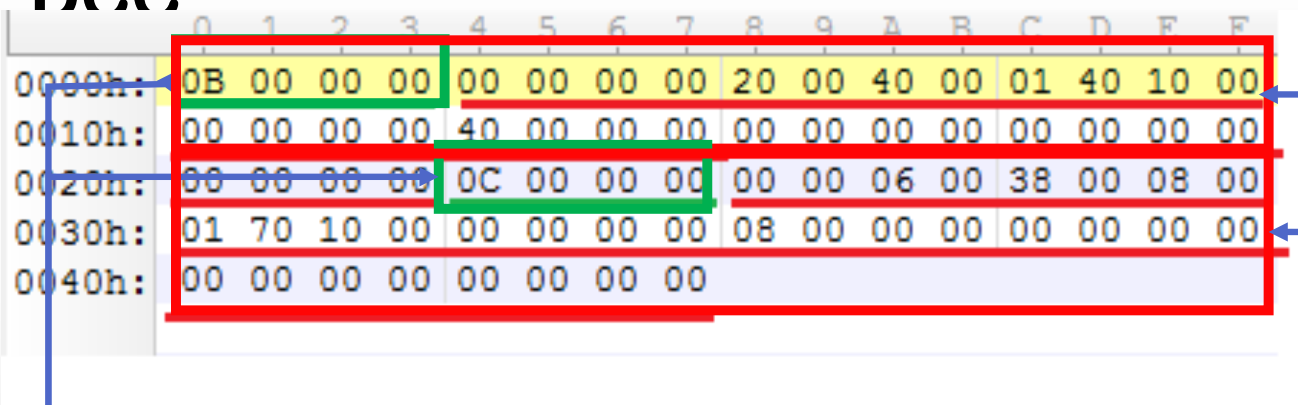
Workflow Overview- Step 2



TREND MICRO

Work Flow- Encode DCD and read in – Step 3

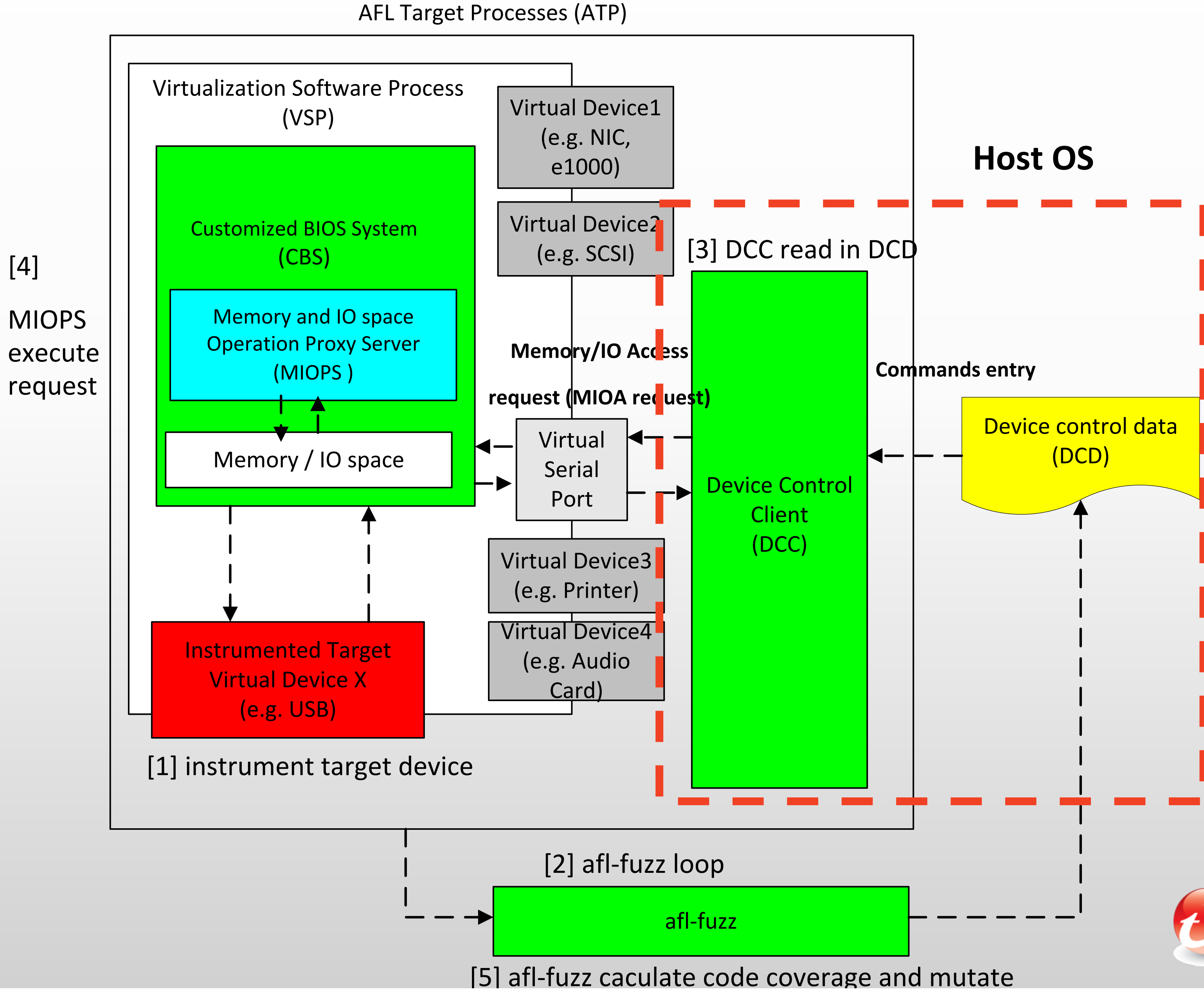
- **Step3:Encode MIOA Requests to DCD and read into DCC**



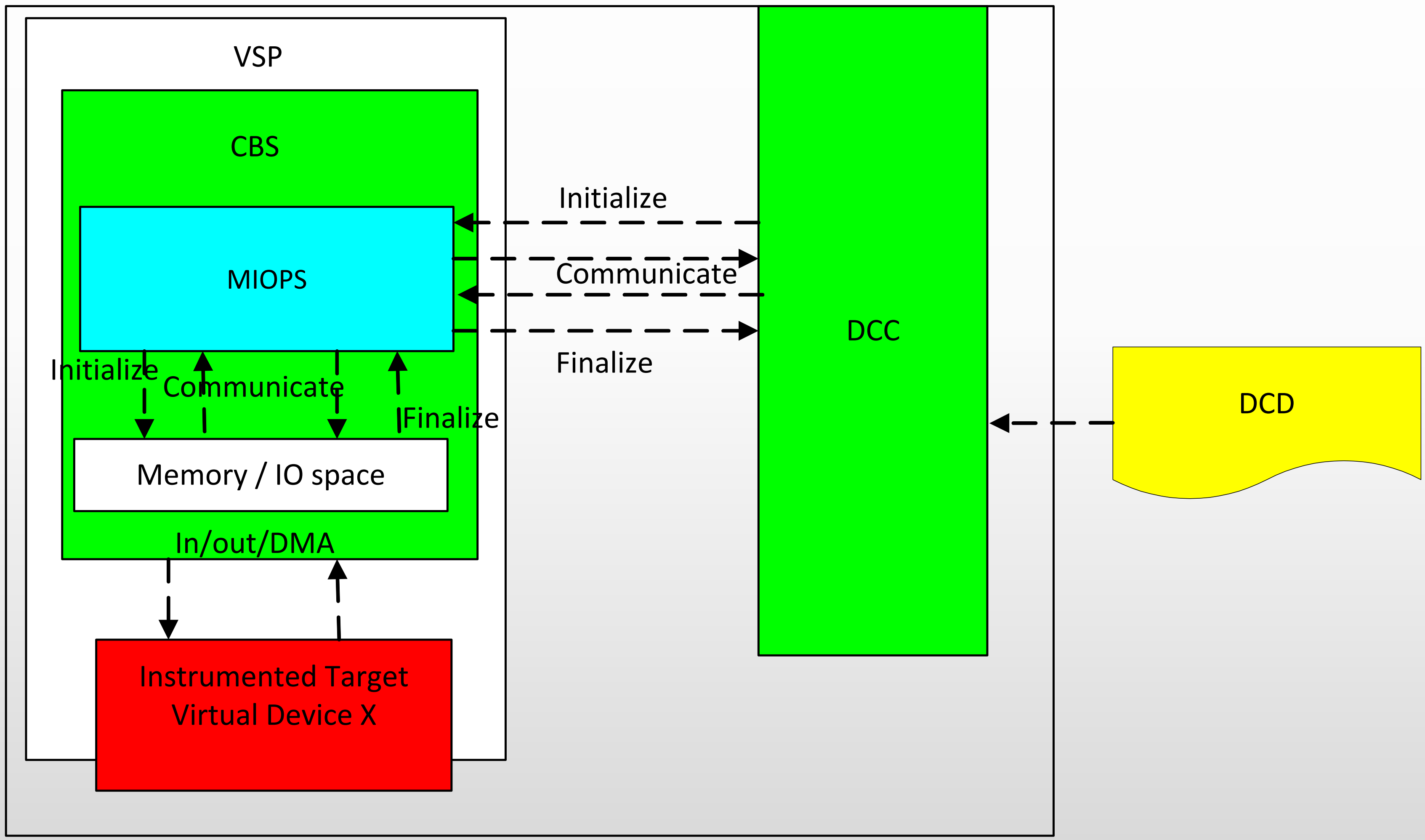

```
typedef struct _command_entry
{
   u32   slotid;
   u32 _command_id;
   void*    inctx;
   u8 _input_buf[32];

}command_entry_t;
```

Workflow Overview – Step3



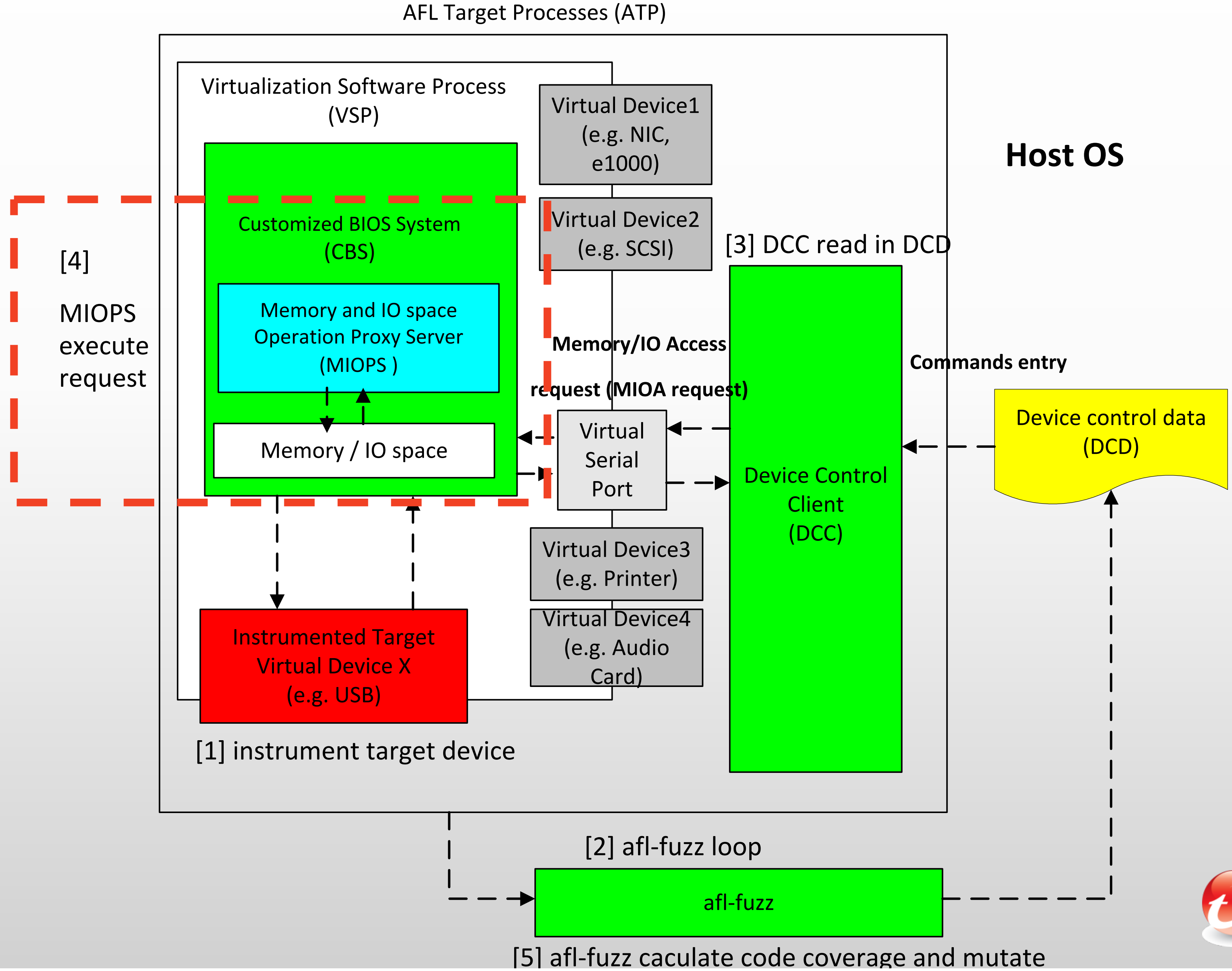
Work Flow-
Control Devices Communicate –Step 4/4



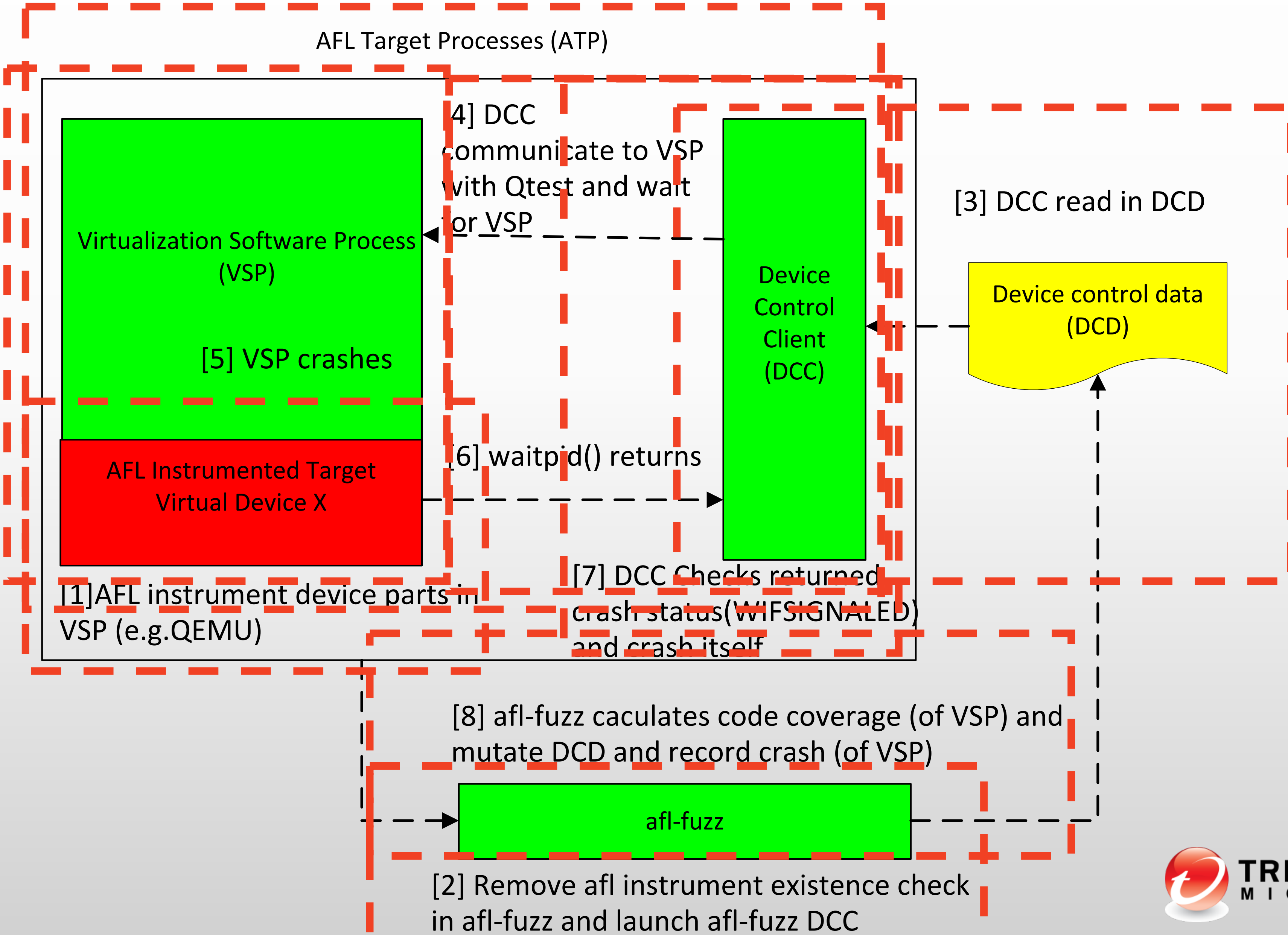
Work Flow-
Control Devices Communicate - Step 4/4

- **Step4: Communication Loop**
  - **CBS starts up**
    - **Start & discover devices**
    - **Start MIOPS in it**
  - **MIOPS starts up**
    - **Initiate target devices**
      - ***E.g. doorbell/command/event address of USB XHCI device***
  - **Communicate between MIOPS and DCC until crash**
    - **DCC read in DCD**
    - **MIOPS execute data**

Workflow Overview – Step 4



Work Flow-
In view of AFL



AFL Tips- Part Instrumentation - Compile Source1/2

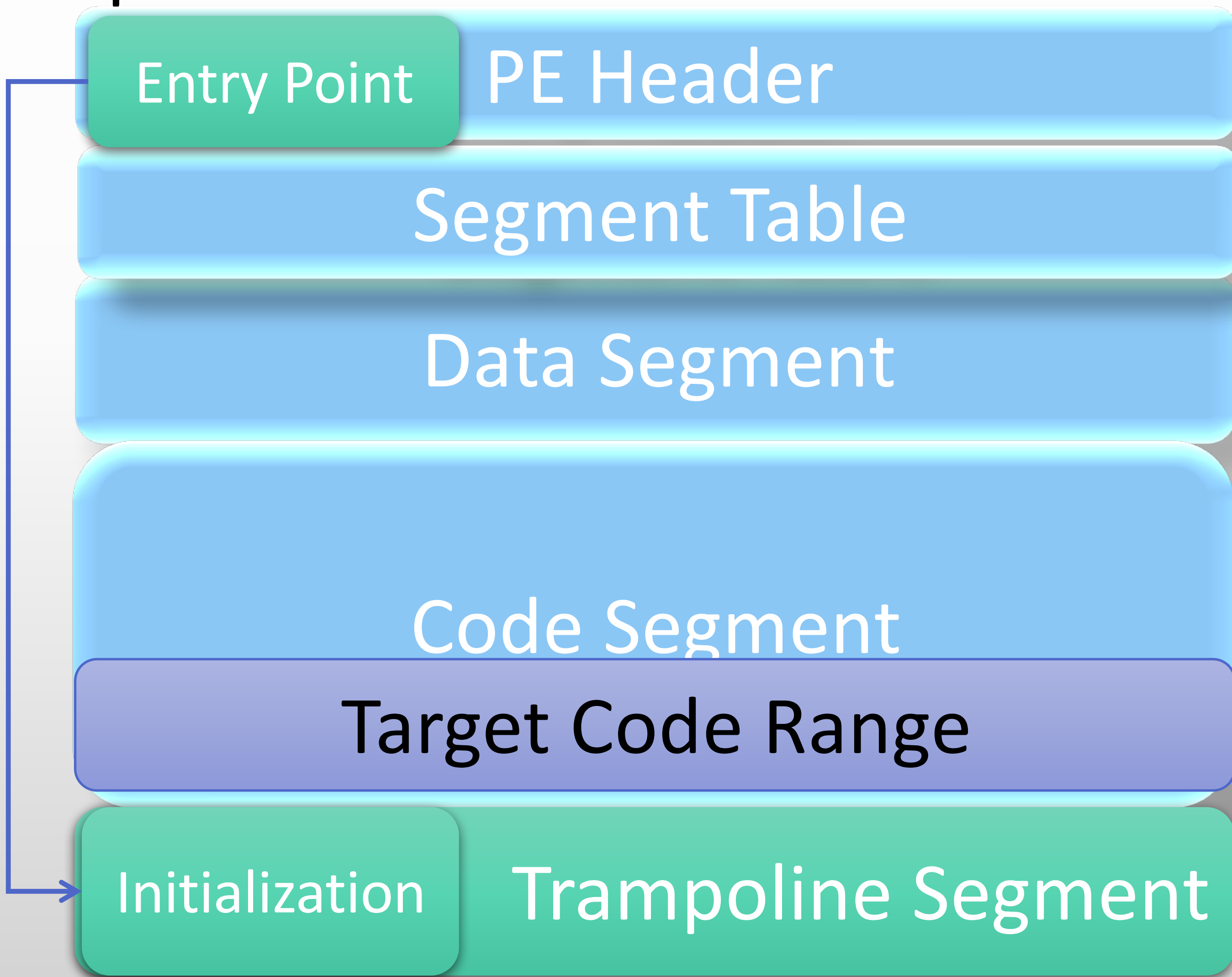
Tips: Compile the source code in part using afl-gcc, for example:

Makefile:

```
%.o: %.c
	$(call quiet-command,\
	if [ $@ = "hw/usb/hcd-xhci.o" -o $@ = "hw/usb/dev-storage.o" ] ;\
	then \
		echo afl-gcc... $< $@   ;\
		afl-gcc $(QEMU_INCLUDES) $(QEMU_CFLAGS) $(QEMU_DGFLAGS) $(CFLAGS) $($@-cflags) -c -o $@ $< ;\
	else \
		echo cc... $<  $@ ;  \
		$(CC) $(QEMU_INCLUDES) $(QEMU_CFLAGS) $(QEMU_DGFLAGS) $(CFLAGS) $($@-cflags) -c -o $@  $< ;\
```

AFL Tips-
Part Instrumentation – Instrument PE 2/2

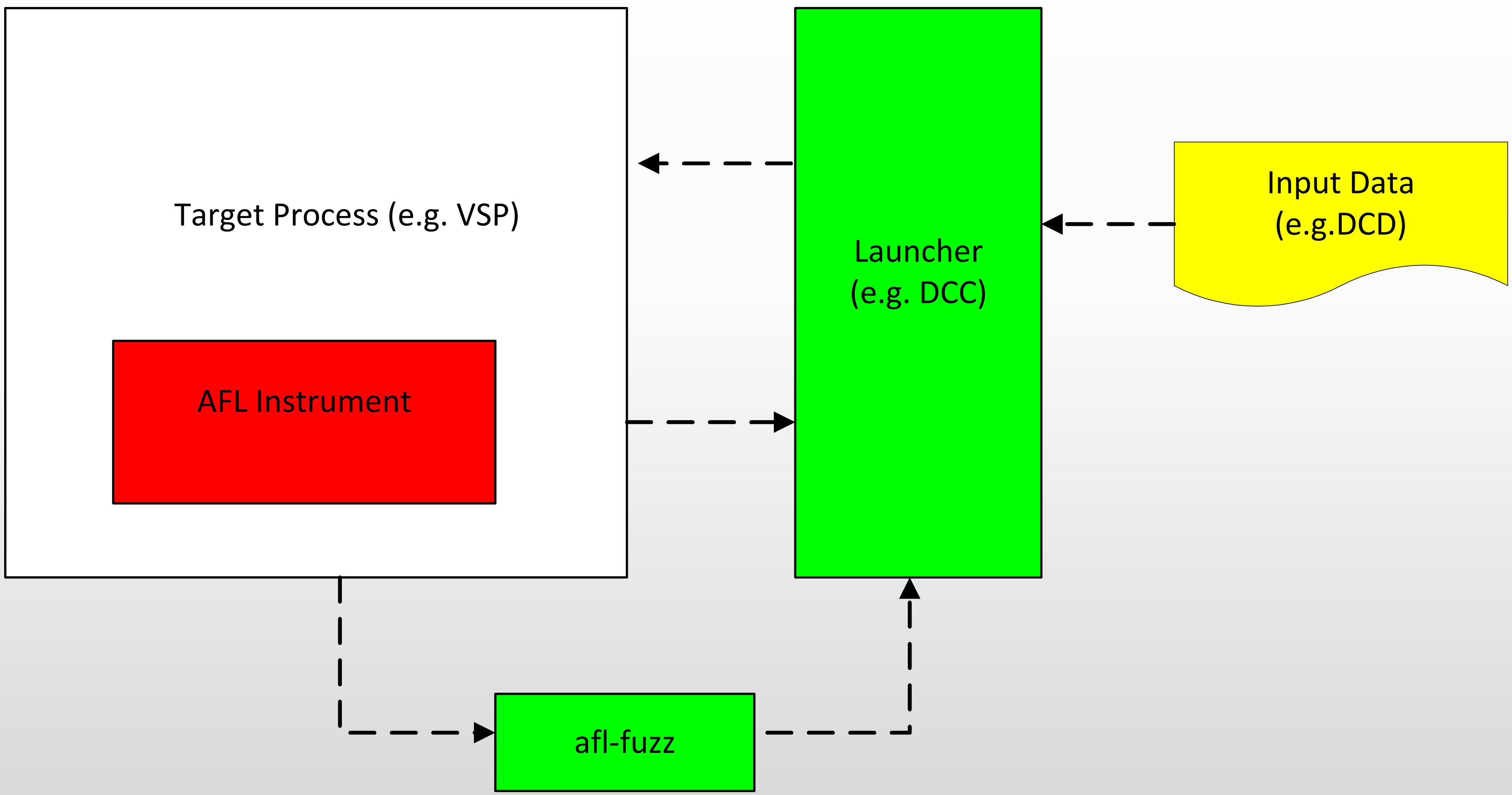
Tips: Instrument the executable file for close-source software




```
static const u8* trampoline_f

".align 4\n"
"leal -16(%%esp), %%esp\n"
"movl %%edi,  0(%%esp)\n"
"movl %%edx,  4(%%esp)\n"
"movl %%ecx,  8(%%esp)\n"
"movl %%eax, 12(%%esp)\n"
"movl $0x%08x, %%ecx\n"
"call __afl_maybe_log\n"
"movl 12(%%esp), %%eax\n"
"movl  8(%%esp), %%ecx\n"
"movl  4(%%esp), %%edx\n"
"movl  0(%%esp), %%edi\n"
"leal 16(%%esp), %%esp\n"
```

AFL Tips-
Multiple Processes 1/3



AFL Tips- Multiple Processes 2/3

- Trace info is shared across processes in nature
  - *static inline void afl_maybe_log(abi_ulong cur_loc)*
  - Trace cur_loc in shared memory mechanism
- Remove instrumentation check
  - In Afl-fuzz
  - Because Launcher(e.g.DCC) is not instrumented
- Deliver crash info from Target to Launcher
  - fork & waitpid
  - Check connection(e.g. serial port status)

TREND MICRO™

AFL Tips- Multiple Processes 3/3

```
fpid = fork();
if (fpid == 0)
{
    //In child process
    launchVSP(argv);
    exit(0);
}
else
{
    //In parent process (DCC)
    waitpid(fpid, &status, 0);
    if (WIFEXITED(status))
    {
    //Child process normal exit, bypass
    }
    else if (WIFSIGNALED(status))
    {
    //Child process crash signal, crash self
    crashMyself();
    }
}
```

TREND MICRO

AFL Tips- Misc

- **Adjust parameter for afl-fuzz to avoid start failure**
  - -t, set more time for fuzz cycle
  - -m, set more memory for virtual machine
  - E.g. afl-fuzz -t 9 -m 2048 -i ${test_root}/IN/ -o ${test_root}/OUT/ ${DCC} @@
- …

What We Will Cover

- Case Study
  - Fuzz FDC and Reproduce Venom Vulnerability(CVE-2015-3456)
- Demo

Case Study 1/3

1. afl-gcc fdc.c

```
%.o: %.c
	$(call quiet-command,\
	if [ $@ = "hw/block/fdc.o" ] ;\
	then \
		echo afl-gcc... $< $@    ;\
		afl-gcc $(QEMU_INCLUDES) $(QEMU_CFLAGS) $(QEMU_DGFLAGS) $(CFLAGS) $($@-cflags) -c -o $@ $< ;\
	else \
		echo cc... $<  $@ ;  \
		$(CC) $(QEMU_INCLUDES) $(QEMU_CFLAGS) $(QEMU_DGFLAGS) $(CFLAGS) $($@-cflags) -c -o $@  $< ;\
	fi)
```

2. Designing input case file(i.e. DCD) format

```
struct fdc_command
{
unsigned char cid;
unsigned int args_count;
unsigned int args[0];
};
```

Case Study 2/3

3. Prepare 30 input case file

- each case for one floppy disk controller command
  - For example: FD_CMD_READ, FD_CMD_WRITE, FD_CMD_SEEK…

4. Preparing DCC

- Parsing input case file and translating to MIOA

```
struct fdc_command
{
        cid = 0x8e
        args_count = 0x5
        args[] = [0x45, 0x12, 0x34, 0x7f, 0x98]
};
```

```
"outb 0x3f5 0x8e"
"outb 0x3f5 0x45"
"outb 0x3f5 0x12"
"outb 0x3f5 0x34"
"outb 0x3f5 0x7f"
"outb 0x3f5 0x98"
```

Case Study 3/3

5. Using commands to start testing

afl-fuzz -t 99000 -m 2048 -i IN/ -o OUT/ <DCC command> @@

Demo

Reference

- http://venom.crowdstrike.com/
- http://www.slideshare.net/CanSecWest/csw2016-tang-virtualizationdevice-emulator-testing-technology
- http://lcamtuf.coredump.cx/AFL/
- https://www.youtube.com/watch?v=O9P7kXm5WSg
- Filesystem Fuzzing with American Fuzzy Lop

https://events.linuxfoundation.org/sites/events/files/slides/AFL%20filesystem%20fuzzing,%20Vault%202016_0.pdf

- Triforce-run-afl-on-everything

https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2016/june/project-triforce-run-afl-on-everything/

- https://www.seabios.org/SeaBIOS
- https://www.seabios.org/Execution_and_code_flow
- http://wiki.qemu.org/Features/QTest
- http://www.intel.com/content/www/us/en/io/universal-serial-bus/extensible-host-controler-interface-usb-xhci.html
- http://wiki.osdev.org/Floppy_Disk_Controller

Thanks very much