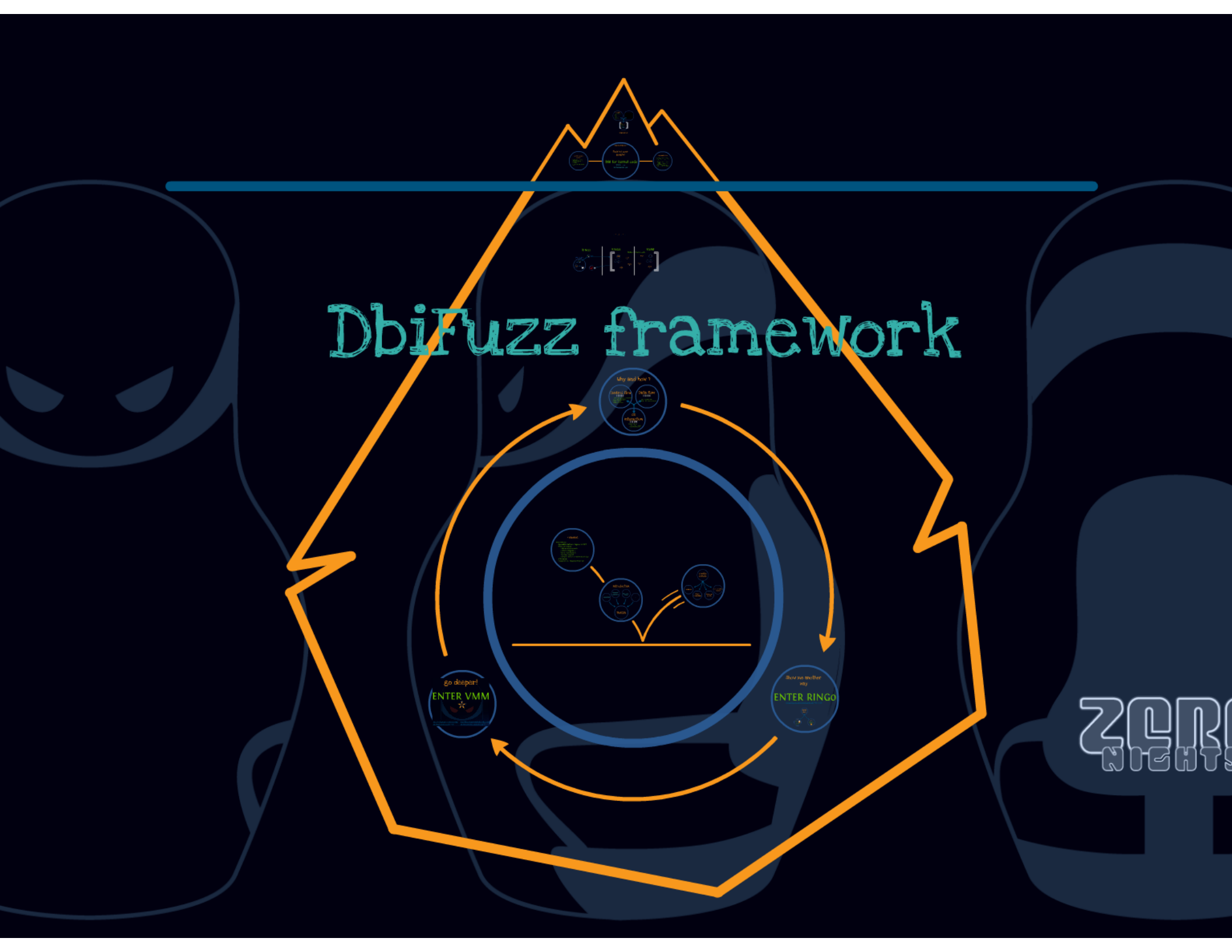
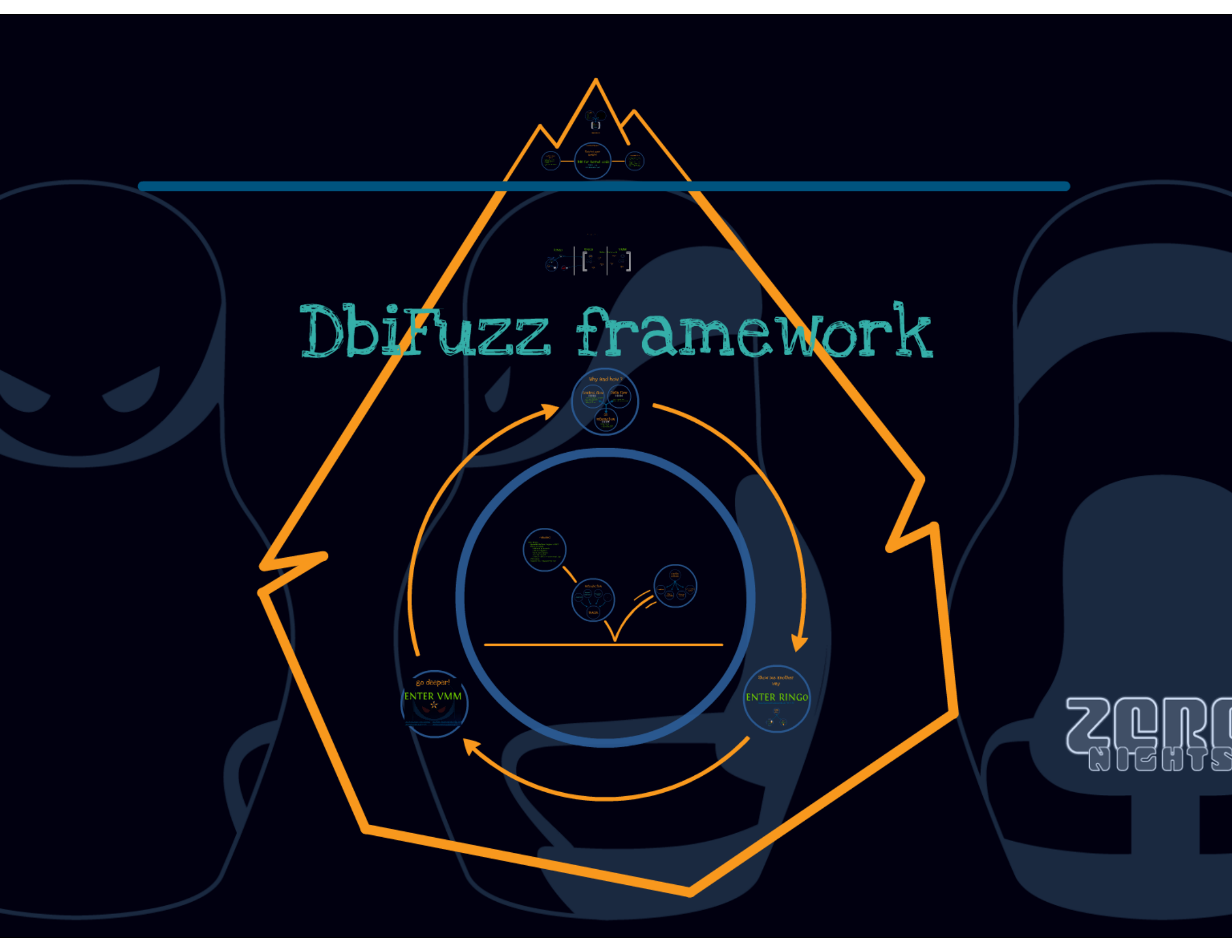

DbiFuzz framework
Why and how ?
go deeper!
ENTER VMM
ENTER RING0

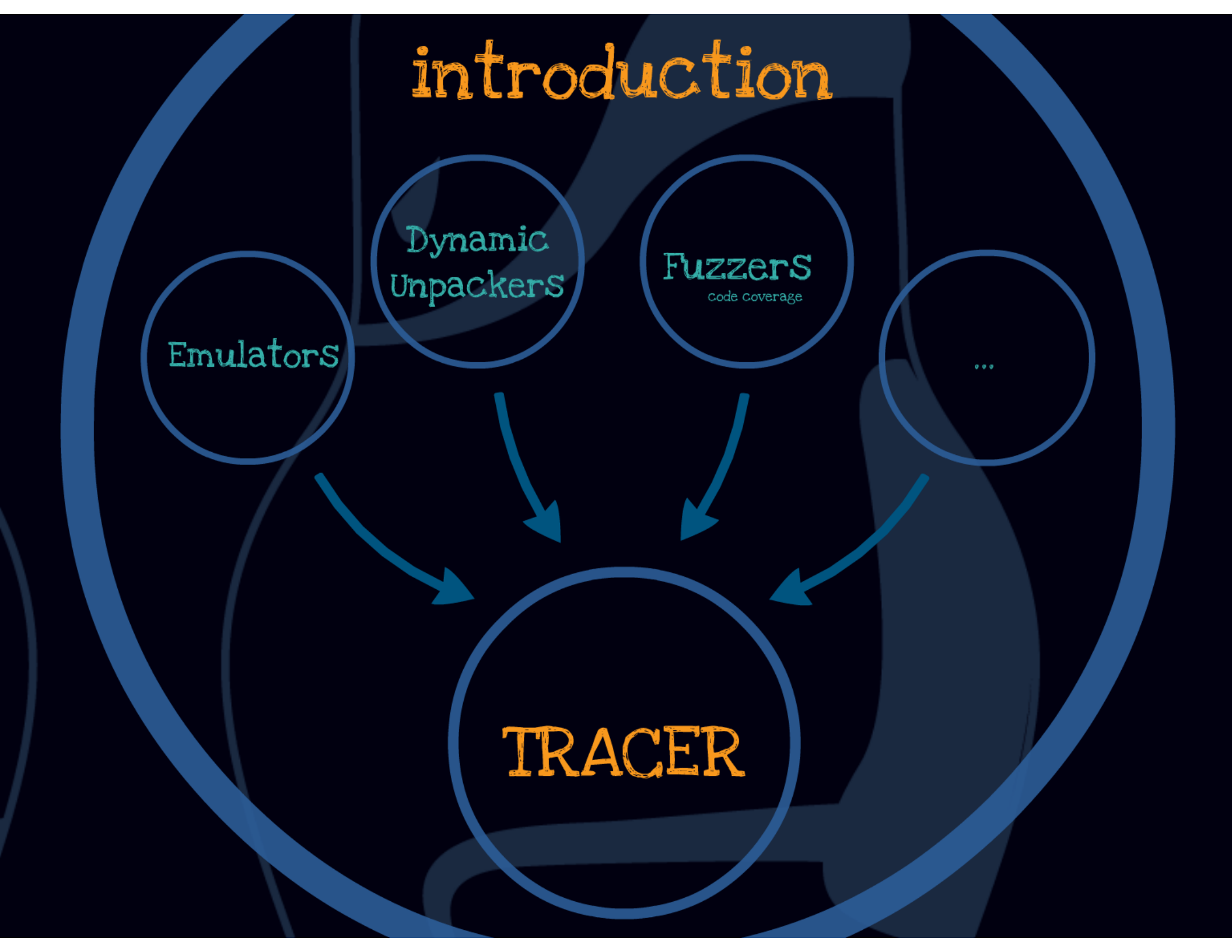
DbiFuzz framework

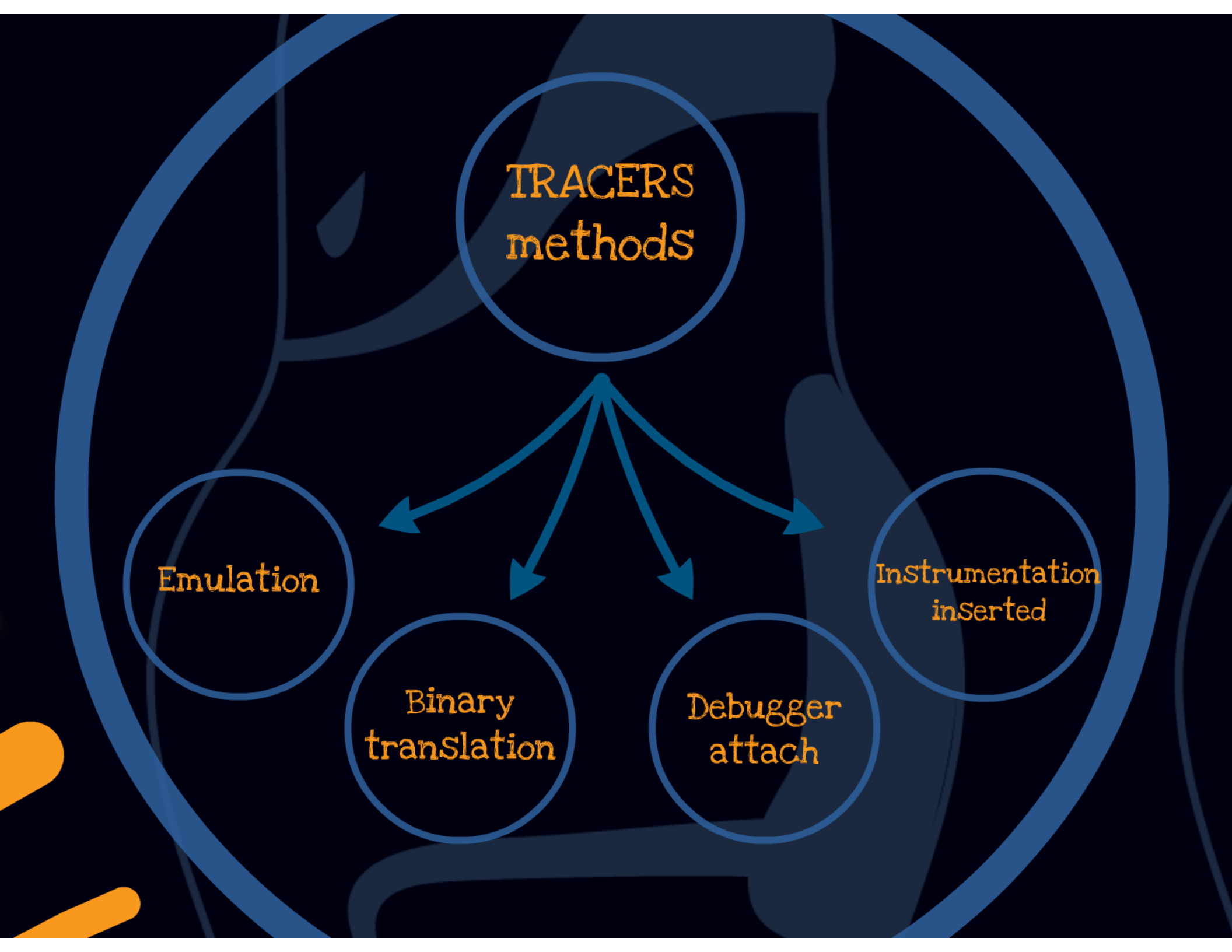


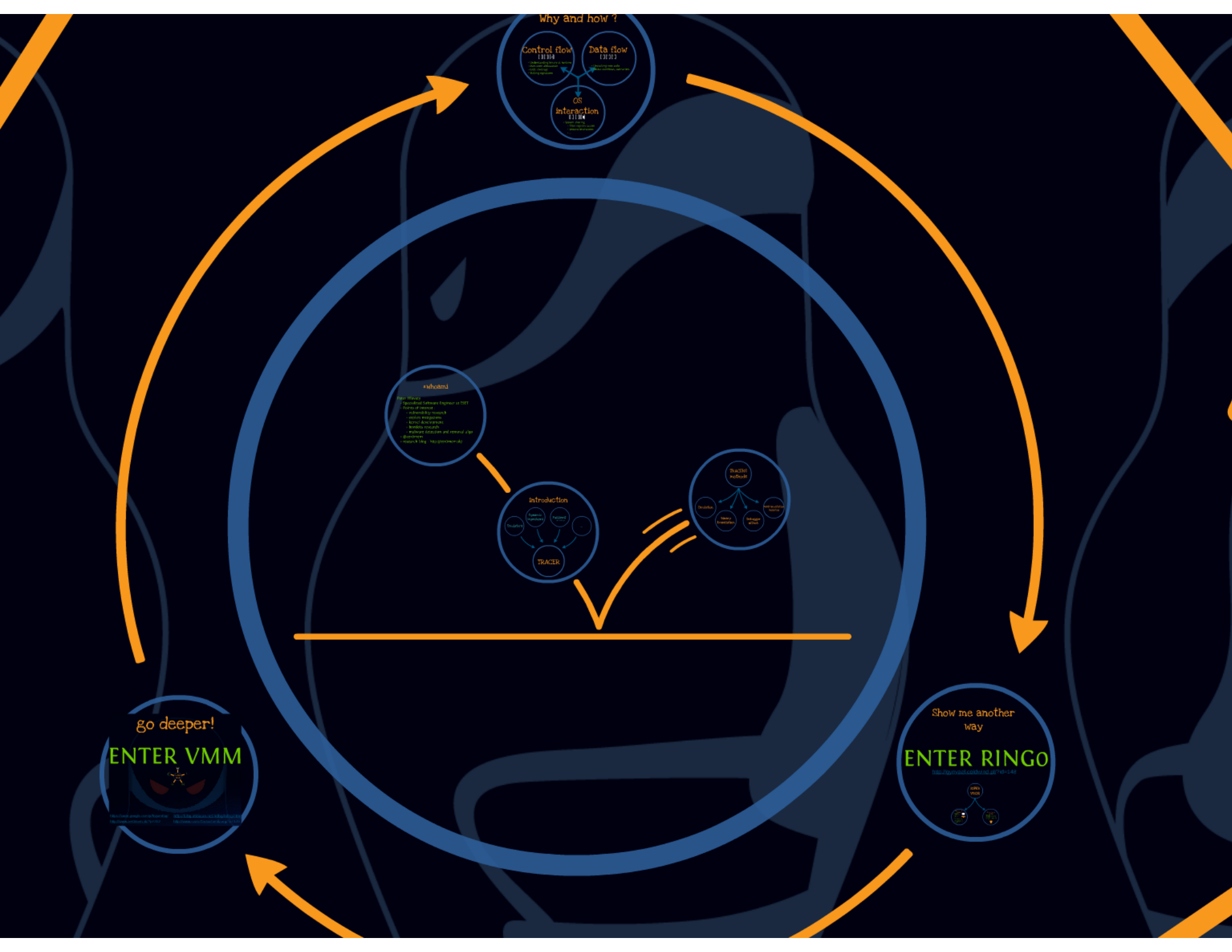
#whoami

Peter Hlavaty

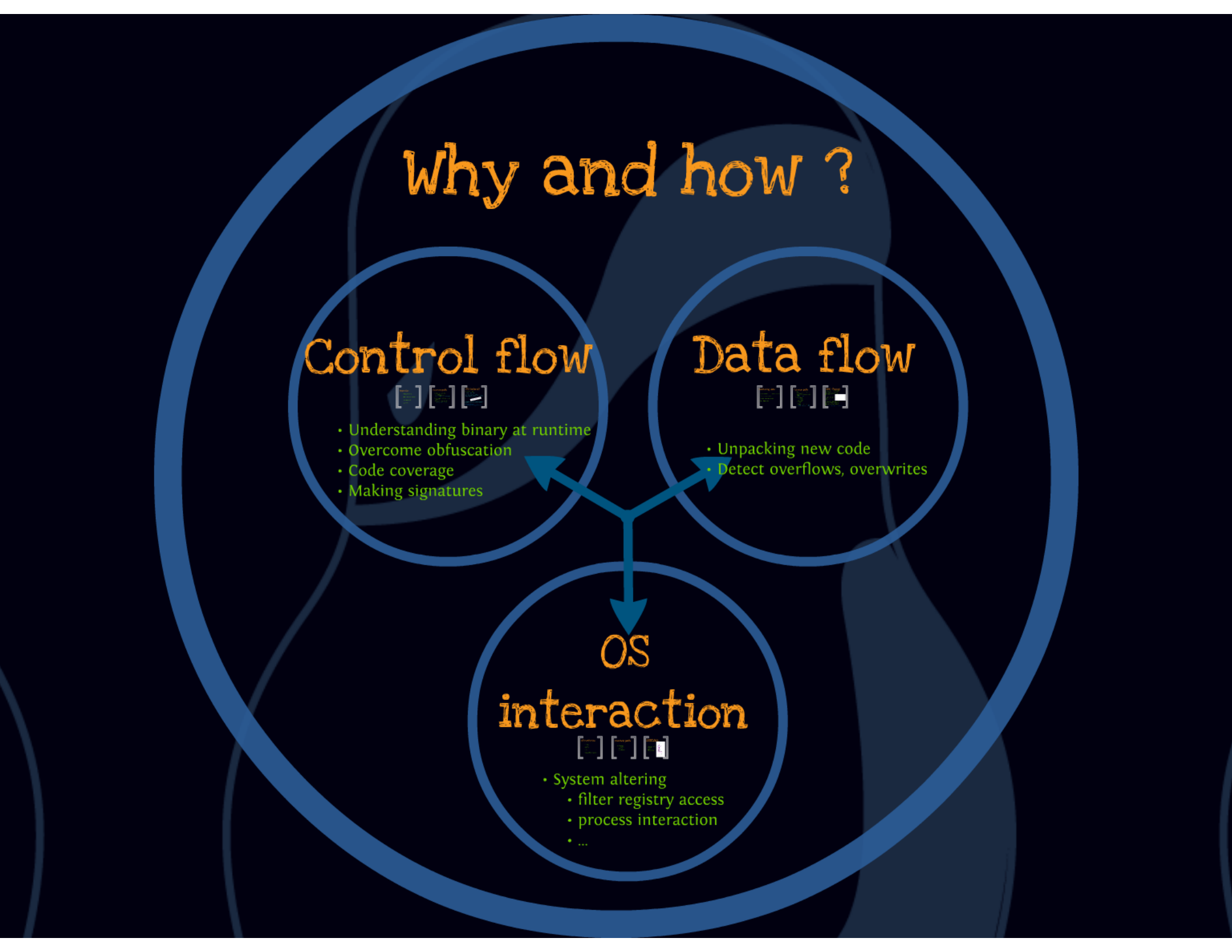
- Specialized Software Engineer at ESET
- Points of interest :
  - vulnerability research
  - exploit mitigations
  - kernel development
  - bootkits research
  - malware detection and removal algo
- @zer0mem
- research blog : http://zer0mem.sk/


introduction
Dynamic Unpackers
Fuzzers
code coverage
Emulators
...
TRACER


TRACERS
methods
Emulation
Binary
translation
Debugger
attach
Instrumentation
inserted


Why and how ?
Control flow
Data flow
OS interaction
#whoami
Introduction
TRACER
go deeper!
ENTER VMM
Show me another way
ENTER RINGO

Why and how ?


Control flow
• Understanding binary at runtime
• Overcome obfuscation
• Code coverage
• Making signatures
Data flow
• Unpacking new code
• Detect overflows, overwrites
OS interaction
• System altering
• filter registry access
• process interaction
• ...

Control flow

- Understanding binary at runtime
- Overcome obfuscation
- Code coverage
- Making signatures

trace for :

- every intruction
- based on basic blocks
- function calls
- api calls

Common path

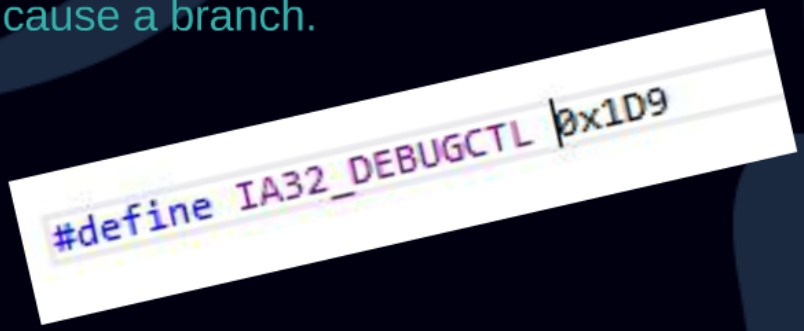
- Insert pre / post instrumentation
  - per instruction
  - per analysed Basic Block
  - ... a lot of hooks, a lot of preprocessing ...
- Based on debugger – handling trap exception
  - run debuger
  - set SEH
  - hook ntdll!KiExceptionDispatcher
  - ... every time invoked exception ...

CPU features!

Intel MSR BTF flag

17.4.3 Single-Stepping on Branches

When software sets both the BTF flag (bit 1) in the IA32_DEBUGCTL MSR and the TF flag in the EFLAGS register, the processor generates a single-step debug exception only after instructions that cause a branch.



The processor clears the BTF flag when it generates a debug exception. The debugger must set the BTF flag before
resuming program execution to continue single-stepping on branches.

http://pedramamini.com/blog/2006.12.13/
https://groups.google.com/forum/#!topic/linuxkernelnewbies/HksIHK_LNSg
http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf

Data flow

- Unpacking new code
- Detect overflows, overwrites

Monitoring data

- Monitoring RW access (selfmodify code, data)
- Checking structs integrity
- Monitor input parsing process
- Save / Restore state

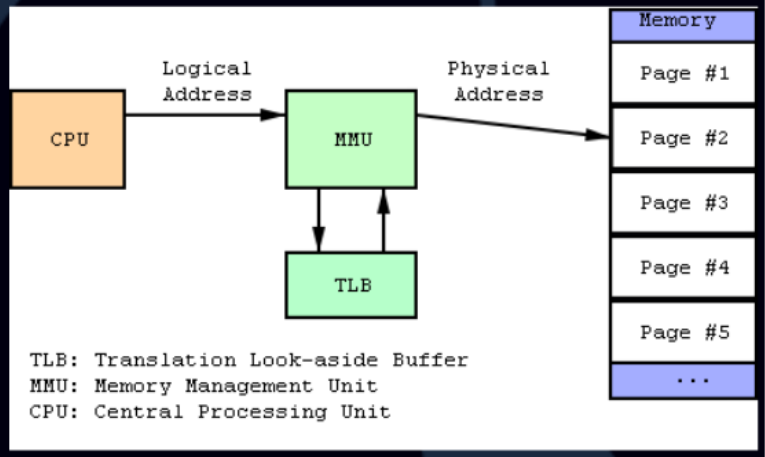
Common path

- Disasm
  - Analyse all instructions – find LOAD / STORE instructions
  - Parse LOAD / STORE instruction, get targeted memory
  - … analyse each instruction …
- Virtual Memory Protection
  - call VirtualProtect
    - PAGE_NOACCESS
    - PAGE_READONLY
    - PAGE_READWRITE
    - PAGE_EXECUTE
  - Handle access by
    - Debugger
    - SEH
    - Ntdll!KiExceptionHandler hook
  - … every time invoked exception handling …

MMU - Paging!

Most MMUs use an in-memory table of items called a "page table," containing one "page table entry" (PTE) per page, to map virtual page numbers to physical page numbers in main memory.

The physical page number is combined with the page offset to give the complete physical address.



A PTE may also include information about whether the page has been written to (the "dirty bit"), when it was last used (the "accessed bit"), what kind of processes (user mode or supervisor mode) may read and write it, and whether it should be cached.

http://en.wikipedia.org/wiki/Memory_management_unit

OS interaction

- System altering
  - filter registry access
  - process interaction
  - ...

API monitoring

- Process
- Registry
- Files
- targeted binary specific

Common path

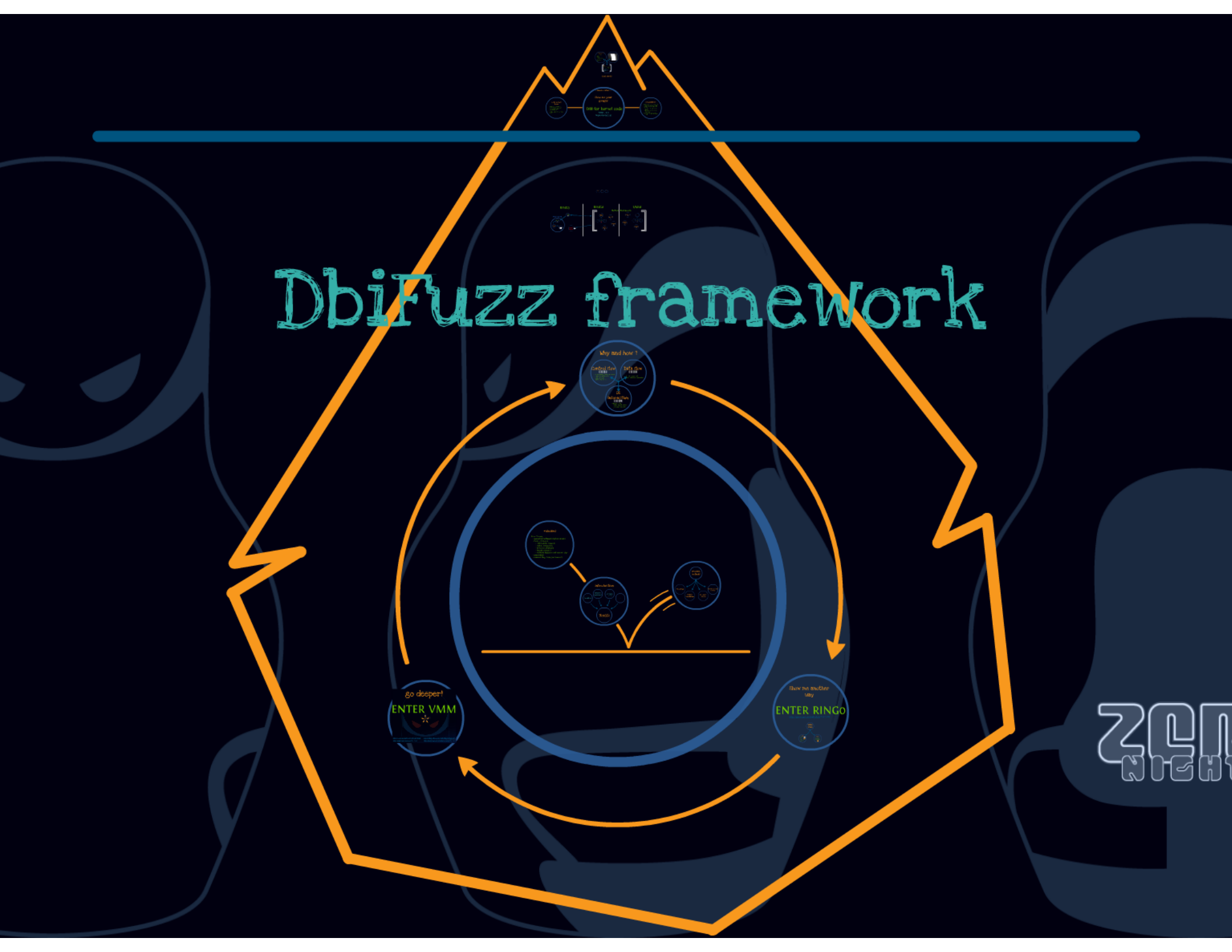
API Hooking

- trampolines
- imports
- breakpoints

SYSCALL

- Applications alter OS via syscalls
- SYSCALL invoke nt!KiSystemCall64

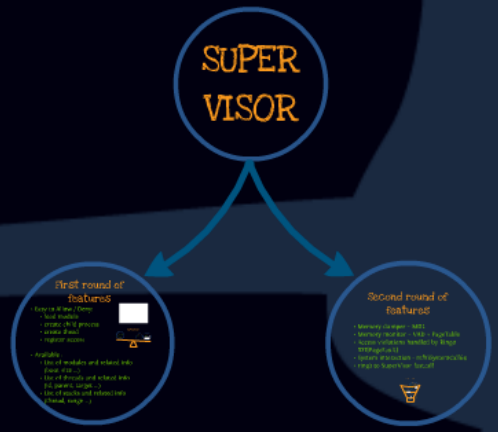
```
//win8
enum
{
    ntdll_NtWorkerFactoryWorkerReady = 0,
    ntdll_NtMapUserPhysicalPagesScatter,
    ntdll_ZwWaitForSingleObject,
    ntdll_ZwCallbackReturn,
    ntdll_NtReadFile,
    ntdll_NtDeviceIoControlFile,
    ntdll_NtWriteFile,
    ntdll_ZwRemoveIoCompletion,
    ntdll_ZwReleaseSemaphore,
    ntdll_NtReplyWaitReceivePort,
    ntdll_NtReplyPort,
    ntdll_ZwSetInformationThread,
    ntdll_NtSetEvent,
    ntdll_NtClose,
    ntdll_NtQueryObject,
    ntdll_ZwQueryInformationFile,
    ntdll_ZwOpenKey,
    ntdll_NtEnumerateValueKey,
    ntdll_NtFindAtom,
    ntdll_NtQueryDefaultLocale,
    ntdll_ZwQueryKey,
    ntdll_NtQueryValueKey,
    ntdll_NtAllocateVirtualMemory,
    ntdll_ZwQueryInformationProcess,
    ntdll_ZwWaitForMultipleObjects32,
    ntdll_NtWriteFileGather,
    ntdll_NtSetInformationProcess,
    ntdll_NtCreateKey,
    ntdll_ZwFreeVirtualMemory,
    ntdll_ZwImpersonateClientOfPort,
    ntdll_NtReleaseMutant,
    ntdll_NtQueryInformationToken,
    ntdll_NtRequestWaitReplyPort,
    ntdll_ZwQueryVirtualMemory,
    ntdll_ZwOpenThreadToken,
    ntdll_NtQueryInformationThread,
    ntdll_NtOpenProcess,
```


DbiFuzz framework
Why and how ?
go deeper!
ENTER VMM
show me another way
ENTER RINGO

Show me another way

ENTER RINGo

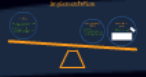
http://gynvael.coldwind.pl/?id=148



SUPER VISOR

First round of features

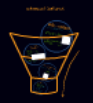
- Easy to Allow / Deny:
  - load module
  - create child process
  - create thead
  - register access



- Available :
  - List of modules and related info (base, size ...)
  - List of threads and related info (id, parent, target ...)
  - List of stacks and related info (thread, range ...)

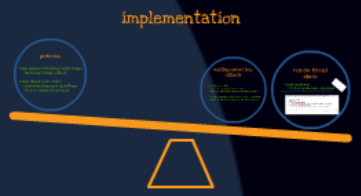
Second round of features

- Memory dumper – MDL
- Memory monitor – VAD + PageTable
- Access violations handled by Ring0 IDT[PageFault]
- System interaction - nt!KiSystemCall64
- ring3 to SuperVisor fastcall



First round of features

- Easy to Allow / Deny:
  - load module
  - create child process
  - create thead
  - register access

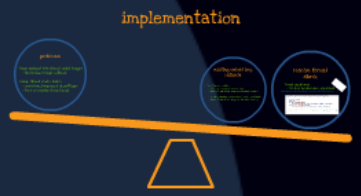


- Available :
  - List of modules and related info (base, size ...)
  - List of threads and related info (id, parent, target ...)
  - List of stacks and related info (thread, range ...)

```cpp
template<class TYPE>
class CProcessMonitor
{
//prohibit outside init
    void operator=(const CProcessMonitor&);

public:
    CProcessMonitor()
    {
        m_processWorker = new CProcessCtxWorker<TYPE>;
        if (!m_processWorker)
            return;

        //registry callback
        UNICODE_STRING altitude;
        RtlInitUnicodeString(&altitude, L"360055");//FSFilter Activity Monitor
        {
            CPassiveLvl irql;

            NTSTATUS status;
            status = PsSetCreateProcessNotifyRoutineEx(ProcessNotifyRoutineEx, FALSE);
            ASSERT(STATUS_SUCCESS == status);

            status = PsSetLoadImageNotifyRoutine(ImageNotifyRoutine);
            ASSERT(STATUS_SUCCESS == status);

            status = PsSetCreateThreadNotifyRoutine(ThreadNotifyRoutine);
            ASSERT(STATUS_SUCCESS == status);

            status = CmRegisterCallbackEx(RegisterCallback, &altitude, gDriverObject, NULL, &m_cookie, NULL);
            ASSERT(STATUS_SUCCESS == status);
        }
    }
```

First round of features

- Easy to Allow / Deny:
  - load module
  - create child process
  - create thead
  - register access



- Available :
  - List of modules and related info (base, size ...)
  - List of threads and related info (id, parent, target ...)
  - List of stacks and related info (thread, range ...)

}

implementation

problems

- Keep updated info about loaded images
  - No Unload Image callback
- Know thread stacks limits
  - avoid touching stack GuardPages
  - how to resolve thread stack

missing unload img callback

- List of loaded modules
  - Each LoadImg register new image
  - throw all overlaped images (unloaded already!)
  - per ThreadNofity resolve thread stack [hard limits]
  - throw all overlaped images (unloaded already!)

resolve thread stack

Thread stack limits

- TEB [DeallocationStack, StackBase]

problems

- Keep updated info about loaded images
  - No Unload Image callback
- Know thread stacks limits
  - avoid touching stack GuardPages
  - how to resolve thread stack

missing unload img callback

- List of loaded modules
  - Each LoadImg register new image
  - throw all overlaped images (unloaded already!)

  - per ThreadNofity resolve thread stack [hard limits]
  - throw all overlaped images (unloaded already!)

reSolve thread stack

Thread stack limits

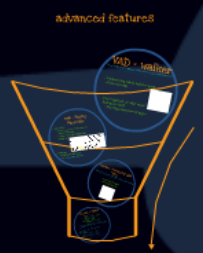
- TEB [DeallocationStack, StackBase]

http://doxygen.reactos.org/d8/d6b/ldrinit_8c_source.html#l02139 [LdrpInit]

```
if (!Id && NT_SUCCESS(PsLookupThreadByThreadId(m_threadId, &Id)))
{
    CPassiveLvl irql;
    void* teb;
    if (teb = GetWow64Teb(Id))
        ResolveThreadLimits<NT_TIB32>(reinterpret_cast<NT_TIB32*>(teb));
    else if (teb = PsGetThreadTeb(Id))
        ResolveThreadLimits<NT_TIB>(reinterpret_cast<NT_TIB*>(teb));

    //DbgPrint("\nstack boundaries : %p %p\n", m_stack.Begin(), m_stack.End());
    return true;
}
```

Second round of features

- Memory dumper – MDL
- Memory monitor – VAD + PageTable
- Access violations handled by Ring0 IDT[PageFault]
- System interaction - nt!KiSystemCall64
- ring3 to SuperVisor fastcall



advanced features

VAD - walker

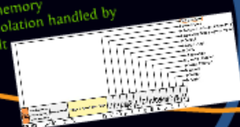
"The VAD tree: A process-eye view of physical memory"

- Enumerating whole address space of user process

- First PageFault on addr cause lookup to VAD!!
  - Altering protection of pages

MMU - Paging PageTable

- PageTables
  - Direct access to protection of related memory
  - Access violation handled by PageFault



- PageFault
  - IDT[PageFault] hook (PatchGuard problem!)

memory descriptor list (MDL)

- Memory dumper
- Patching binary

SYSCALL / SYSRET

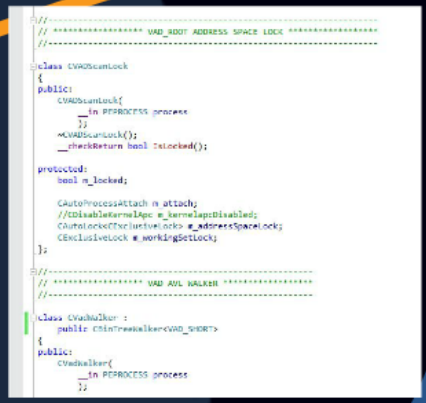
- nt!KiSystemCall64
- MSR[IA64_SYSENTER_EIP] hook
  - (PatchGuard problem!)
- ring3-ring0 FastCall
  - re-implement own syscall

VAD - walker

"The VAD tree: A process-eye view of physical memory"

http://www.dfrws.org/2007/proceedings/p62-dolan-gavitt.pdf

- Enumerating whole address space of user process



```
//---------------------------------------------------------------
// ***************** VAD_ROOT ADDRESS SPACE LOCK *****************
//---------------------------------------------------------------

class CVADScanLock
{
public:
    CVADScanLock(
        __in PEPROCESS process
        );
    ~CVADScanLock();
    __checkReturn bool IsLocked();

protected:
    bool m_locked;

    CAutoProcessAttach m_attach;
    //CDisableKernelApc m_kernelApcDisabled;
    CAutoLockedExclusiveLock m_addressSpaceLock;
    CExclusiveLock m_workingSetLock;
};

//---------------------------------------------------------------
// ***************** VAD AVL WALKER *****************
//---------------------------------------------------------------

class CVadWalker :
    public CBinTreeWalker<VAD_SHORT>
{
public:
    CVadWalker(
        __in PEPROCESS process
        );
```

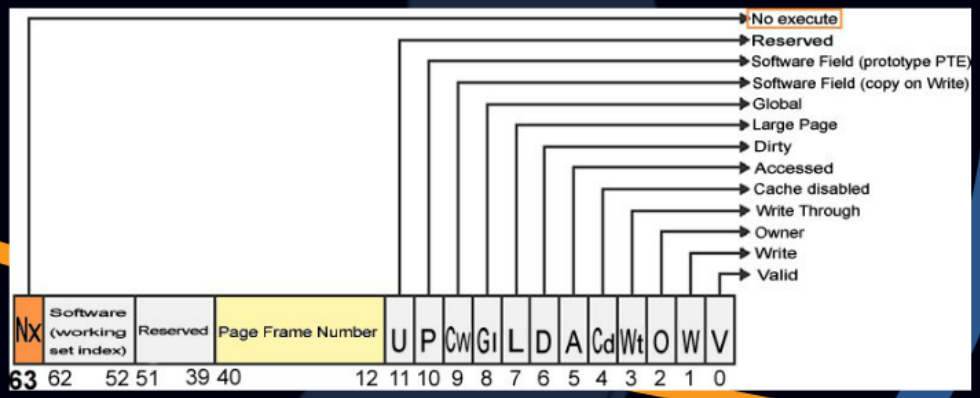
- First PageFault on addr cause lookup to VAD!!
  - Altering protection of pages

Paging Table

```
//-------------------------------------------------------------------
// ****************** VAD_ROOT ADDRESS SPACE LOCK ******************
//-------------------------------------------------------------------

class CVADScanLock
{
public:
    CVADScanLock(
        __in PEPROCESS process
        );
    ~CVADScanLock();
    __checkReturn bool IsLocked();

protected:
    bool m_locked;

    CAutoProcessAttach m_attach;
    //CDisableKernelApc m_kernelapcDisabled;
    CAutoLock<CExclusiveLock> m_addressSpaceLock;
    CExclusiveLock m_workingSetLock;
};

//-------------------------------------------------------
// ****************** VAD AVL WALKER ******************
//-------------------------------------------------------

class CVadWalker :
    public CBinTreeWalker<VAD_SHORT>
{
public:
    CVadWalker(
        __in PEPROCESS process
        );
```

MMU - Paging PageTable

- PageTables
  - Direct access to protection of related memory
  - Access violation handled by PageFault



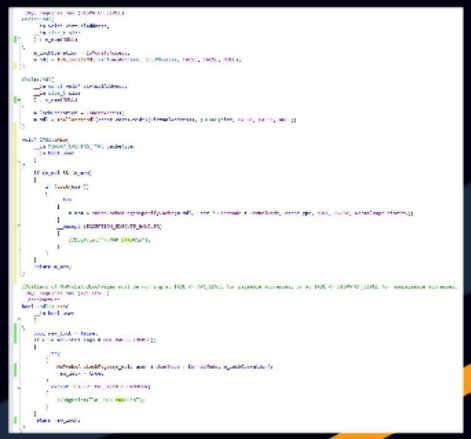
- PageFault
  - IDT[PageFault] hook (PatchGuard problem!)

memory descriptor list (MDL)

"An MDL describes the layout of a virtual memory buffer in physical memory."

http://msdn.microsoft.com/en-us/library/windows/hardware/ff554414(v=vs.85).aspx

- Memory dumper
- Patching binary



```
_IRQL_requires_max_(DISPATCH_LEVEL)
CMdl::CMdl(
    __in void* virtualAddress,
    __in size_t size
    ) : m_mem(NULL)
{
    m_lockOperation = IoModifyAccess;
    m_mdl = IoAllocateMdl(virtualAddress, (ULONG)size, FALSE, FALSE, NULL);
}

CMdl::CMdl(
    __in const void* virtualAddress,
    __in size_t size
    ) : m_mem(NULL)
{
    m_lockOperation = IoReadAccess;
    m_mdl = IoAllocateMdl(const_cast<void*>(virtualAddress), (ULONG)size, FALSE, FALSE, NULL);
}

void* CMdl::Map(
    __in MEMORY_CACHING_TYPE cacheType,
    __in bool user
    )
{
    if (m_mdl && !m_mem)
    {
        if (Lock(user))
        {
            __try
            {
                m_mem = MmMapLockedPagesSpecifyCache(m_mdl, user ? UserMode : KernelMode, cacheType, NULL, FALSE, NormalPagePriority);
            }
            __except (EXCEPTION_EXECUTE_HANDLER)
            {
                //DbgPrint("\nMAP ERROR\n");
            }
        }
    }
    return m_mem;
}

//Callers of MmProbeAndLockPages must be running at IRQL <= APC_LEVEL for pageable addresses, or at IRQL <= DISPATCH_LEVEL for nonpageable addresses.
_IRQL_requires_max_(APC_LEVEL)
__checkReturn
bool CMdl::Lock(
    __in bool user
    )
{
    bool new_lock = false;
    if (!(m_mdl->MdlFlags & MDL_PAGES_LOCKED))
    {
        __try
        {
            MmProbeAndLockPages(m_mdl, user ? UserMode : KernelMode, m_lockOperation);
            new_lock = true;
        }
        __except (EXCEPTION_EXECUTE_HANDLER)
        {
            //DbgPrint("\n LOCK ERROR\n");
        }
    }
    return new_lock;
}
```

SYSCALL / SYSRET

- nt!KiSystemCall64
- MSR[IA64_SYSENTER_EIP] hook
  - (PatchGuard problem!)
- ring3->ring0 FastCall
  - do-implement own syscall

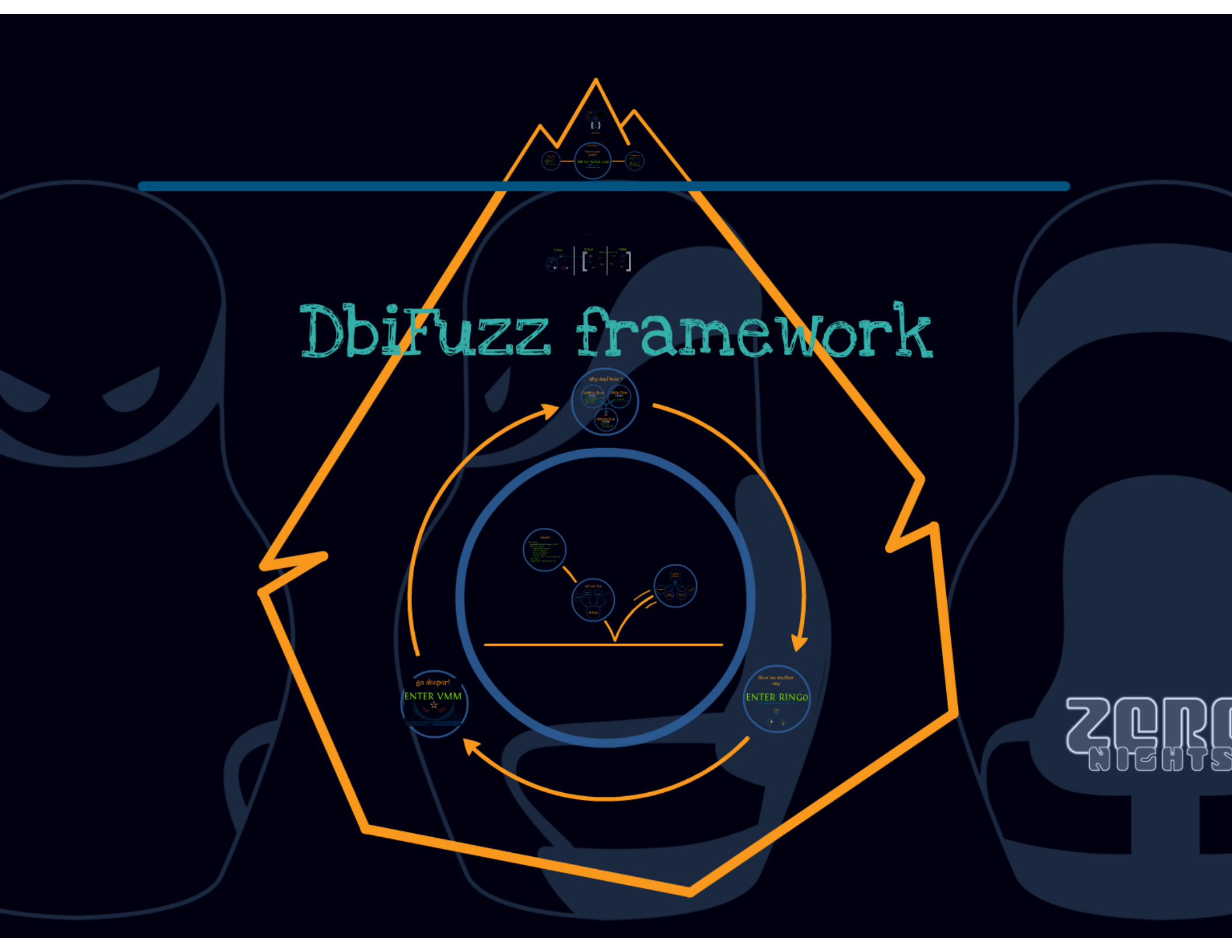
- Operation
- IF (CS.L ≠ 1) or (IA32_EFER.LMA ≠ 1) or (IA32_EFER.SCE ≠ 1)
  (* Not in 64-Bit Mode or SYSCALL/SYSRET not enabled in IA32_EFER *)
  - THEN #UD;
- FI;

- RCX ← RIP; (* Will contain address of next instruction *)
- RIP ← IA32_LSTAR;
- R11 ← RFLAGS;
- RFLAGS ← RFLAGS AND NOT(IA32_FMASK);

- CS.Selector ← IA32_STAR[47:32] AND FFFCH
  (* Operating system provides CS; RPL forced to 0 *

http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf

DbiFuzz framework



go deeper!

ENTER VMM

https://code.google.com/p/hyperdbg/

http://www.zer0mem.sk/?p=302

http://fdbg.x86asm.net/hdbg/hdbg.html

http://www.ivanlef0u.tuxfamily.org/?p=120

HYPER VISOR

Cons

- Virtualize whole OS, not just single app ...
- VMM Exit cost
- Instructions executed per VMM Exit

Pros

- Layer above to ringo
- Callbacks on VMM Exit points
- VMM can be really minimalistic

Overall features

- MultiProcess
- MultiThreading
- x64 binaries
- Fast calls for switching CPL (UM – SV – HV)
- Modul for USER-MODE tracer
- Keeping target binary untouched by DBI as much as possible

another round of features

TRAPS

PageFaults + EPT

CPUID

Cons of current implementation

- EPT is not implemented yet
  - IDT[PageFault] hook implemented
- implemented for win8CP ... but ...
  - all system dependency is in Undoc.hpp
  - check CUndoc::Init() for compatibilty
- Lack of deeper knowledge of VMM
  - Per BTF-VMM Exit is invoked also PageFault
  - Piece of rcode 'injected' to target binary address space

TRAP processing

handle subset of exceptions :
- VMX_VMCS32_GUEST_INTERRUPTIBILITY_STATE

set TRAP to subset :
- VMX_VMCS_CTRL_EXCEPTION_BITMAP :- TRAP_debug

load & store 'debug' informations to container
- RIP, RFLAGS, RSP, branch-src

if BTF & LBR in MSR(VMX_VMCS_GUEST_DEBUGCTL_FULL)
- walking trough LBR_STACK
- pair src + dst

Freeze monitored thread & invoke user-mode tracer
- lack of deeper knowledge of VMM forced me to use ohacks :
  - vmwrite(VMX_VMCS64_GUEST_RIP, kernel_container)
  - in usermode is imidiatelly invoked PageFault
  - handle this PageFault, and set RIP to r) injected code
  - r) injected code is some kind of KeWaitForObjEvent

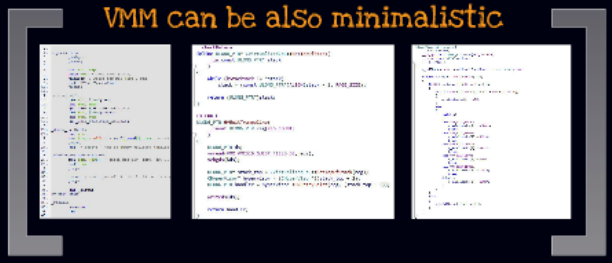
Cons

- Virtualize whole OS, not just single app ...
- VMM Exit cost
- Instructions executed per VMM Exit

Pros

- Layer above to ring0
- Callbacks on VMM Exit points
- VMM can be really minimalistic

VMM can be also minimalistic

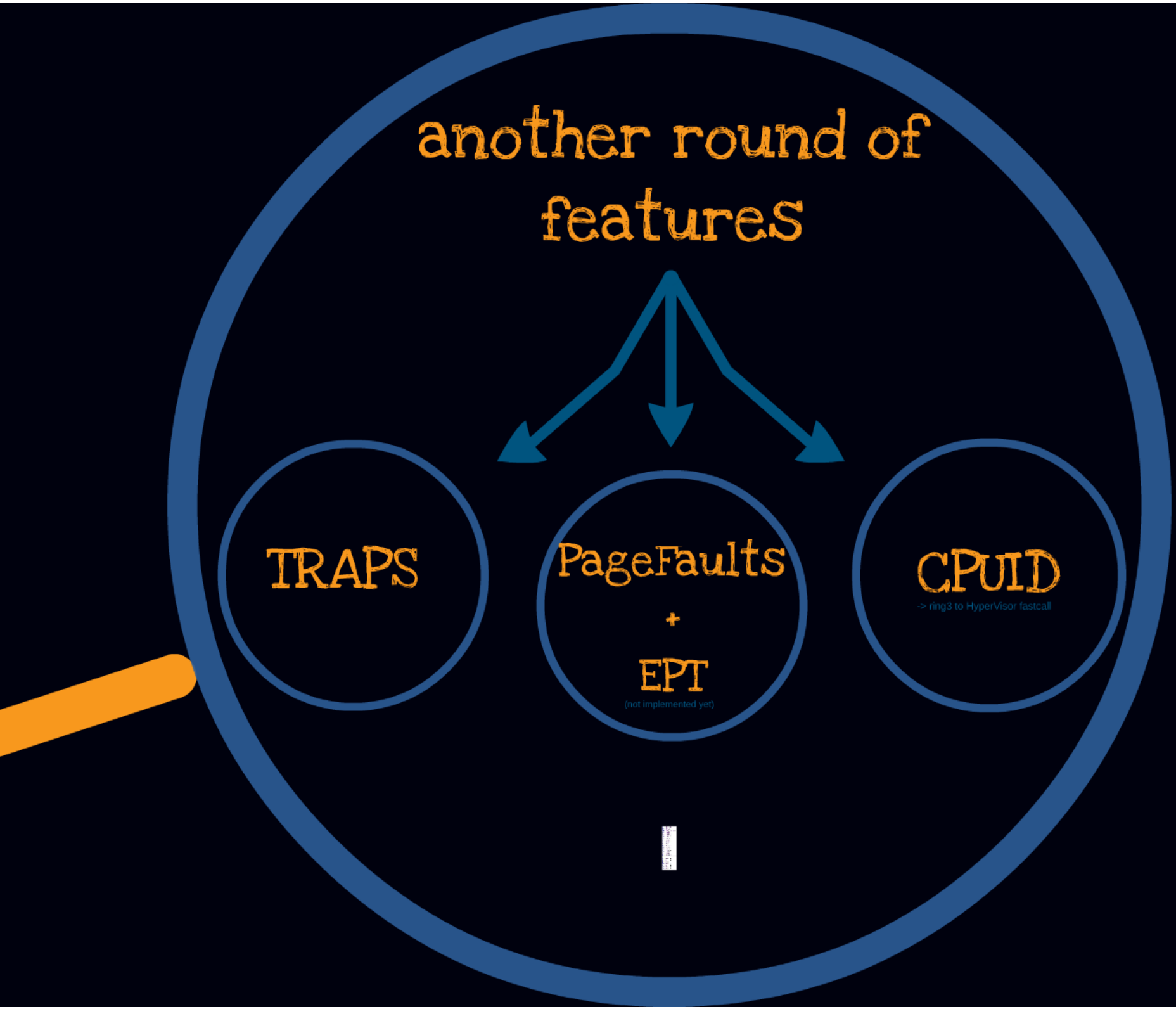


VMM can be also minimalistic

```
hv_exit proc
        pushptr
        pushaq

        mov rcx, rsp
        push rcx  ; save &reg ptr ;)
        pushptr  ; space for mov [arg], rcx
        call HVExitTrampoline
        popptr

;check handler
        lea rcx, @dummy_end
        cmp rax, rcx
        ja _user_specified_callback
        lea rcx, @dummy_start
        cmp rax, rcx
        jb _user_specified_callback

_dummy_callback:
        pop rcx
        mov [rsp + 010h * sizeof(qword)], rax ;ret ->
        popaq
        ret  ; popptr + jmp to dummy_callback handler

_user_specified_callback:
        mov rcx, [rsp]  ; space for mov [arg], rcx ..
        call rax
        popptr

        popaq  ; cause pop all (volatile && non..) reg
        popptr

        jmp _resume
hv_exit endp

_resume:
        vmresume
        ret
```

```
__checkReturn
inline ULONG_PTR* CVirtualizedCpu::GetTopOfStack(
    __in const ULONG_PTR* stack
    )
{
    while (kStackMark != *stack)
        stack = (const ULONG_PTR*)ALIGN(stack - 1, PAGE_SIZE);

    return (ULONG_PTR*)stack;
}

EXTERN_C
ULONG_PTR HVExitTrampoline(
    __inout ULONG_PTR reg[REG_COUNT]
    )
{
    ULONG_PTR ds;
    vmread(VMX_VMCS16_GUEST_FIELD_DS, &ds);
    xchgds(&ds);

    ULONG_PTR* stack_top = CVirtualizedCpu::GetTopOfStack(reg);
    CHyperVisor* hypervisor = (CHyperVisor*)(stack_top + 2);
    ULONG_PTR handler = hypervisor->HVEntryPoint(reg, (stack_top + 1));

    writeds(ds);

    return handler;
}
```

```
CHyperVisor::CHyperVisor(
    __in BYTE coreId,
    __in_opt const ULONG_PTR traps[MAX_CALLBACK],
    __in_opt const VOID* callback
    ) : m_coreId()
{
    m_callback = NULL != callback ? callback : (const VOID*)DUMMY;

    for(int i = 0; i < MAX_CALLBACK; i++)
    {
        if(NULL == traps || NULL == traps[i])
        {
            if ((i > VMX_EXIT_VMCALL) && (i <= VMX_EXIT_VMXON))
            {
                m_hvCallbacks[i] = VMX;
            }
            else
            {
                switch(i)
                {
                case VMX_EXIT_VMCALL:
                    m_hvCallbacks[i] = VMCALL;
                    break;
                case VMX_EXIT_RDMSR:
                    m_hvCallbacks[i] = RDMSR;
                    break;
                case VMX_EXIT_WRMSR:
                    m_hvCallbacks[i] = WRMSR;
                    break;
                case VMX_EXIT_INVD:
                    m_hvCallbacks[i] = INVD;
                    break;
                case VMX_EXIT_CPUID:
                    m_hvCallbacks[i] = CPUID;
                    break;
                default:
                    m_hvCallbacks[i] = DUMMY;
                }
            }
        }
        else
        {
            m_hvCallbacks[i] = traps[i];
        }
    }
}
```


another round of features
TRAPS
PageFaults
+
EPT
(not implemented yet)
CPUID
-> ring3 to HyperVisor fastcall

```
//VMX Basic Exit Reasons.
#define     VMX_EXIT_APIC_ACCESS   44
#define     VMX_EXIT_CPUID   10
#define     VMX_EXIT_CRX_MOVE   28
#define     VMX_EXIT_DRX_MOVE   29
#define     VMX_EXIT_EPT_MISCONFIG   49
#define     VMX_EXIT_EPT_VIOLATION   48
#define     VMX_EXIT_ERR_INVALID_GUEST_STATE   33
#define     VMX_EXIT_ERR_MACHINE_CHECK   41
#define     VMX_EXIT_ERR_MSR_LOAD   34
#define     VMX_EXIT_EXCEPTION   0
#define     VMX_EXIT_EXTERNAL_IRQ   1
#define     VMX_EXIT_HLT   12
#define     VMX_EXIT_INIT_SIGNAL   3
#define     VMX_EXIT_INVALID   -1
#define     VMX_EXIT_INVD   13
#define     VMX_EXIT_INVEPT   50
#define     VMX_EXIT_INVPG   14
#define     VMX_EXIT_INVVPID   53
#define     VMX_EXIT_IO_SMI_IRQ   5
#define     VMX_EXIT_IRQ_WINDOW   7
#define     VMX_EXIT_MONITOR   39
#define     VMX_EXIT_MWAIT   36
#define     VMX_EXIT_PAUSE   40
#define     VMX_EXIT_PORT_IO   30
#define     VMX_EXIT_PREEMPTION_TIMER   52
#define     VMX_EXIT_RDMSR   31
#define     VMX_EXIT_RDPMC   15
#define     VMX_EXIT_RDTSC   16
#define     VMX_EXIT_RSM   17
#define     VMX_EXIT_SIPI   4
#define     VMX_EXIT_SMI_IRQ   6
#define     VMX_EXIT_TASK_SWITCH   9
#define     VMX_EXIT_TPR   43
#define     VMX_EXIT_TR_ACCESS   47
#define     VMX_EXIT_TRIPLE_FAULT   2
#define     VMX_EXIT_VMCALL   18
#define     VMX_EXIT_VMCLEAR   19
#define     VMX_EXIT_VMLAUNCH   20
#define     VMX_EXIT_VMPTRLD   21
#define     VMX_EXIT_VMPTRST   22
#define     VMX_EXIT_VMREAD   23
#define     VMX_EXIT_VMRESUME   24
#define     VMX_EXIT_VMWRITE   25
#define     VMX_EXIT_VMXOFF   26
#define     VMX_EXIT_VMXON   27
#define     VMX_EXIT_WBINVD   54
#define     VMX_EXIT_WRMSR   32
#define     VMX_EXIT_XDTR_ACCESS   46
#define     VMX_EXIT_XSETBV   55

/*
 * Trap/fault mnemonics.
 */
#define TRAP_divide_error      0
#define TRAP_debug             1
#define TRAP_nmi               2
#define TRAP_int3              3
#define TRAP_overflow          4
#define TRAP_bounds            5
#define TRAP_invalid_op        6
#define TRAP_no_device         7
#define TRAP_double_fault      8
#define TRAP_copro_seg         9
#define TRAP_invalid_tss      10
#define TRAP_no_segment       11
#define TRAP_stack_error      12
#define TRAP_gp_fault         13
#define TRAP_page_fault       14
#define TRAP_spurious_int     15
#define TRAP_copro_error      16
#define TRAP_alignment_check  17
#define TRAP_machine_check    18
#define TRAP_simd_error       19
```

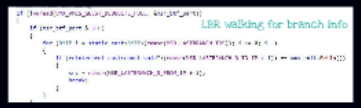
TRAP processing

handle subset of exceptions :
- VMX_VMCS32_GUEST_INTERRUPTIBILITY_STATE

set TRAP to subset :
- VMX_VMCS_CTRL_EXCEPTION_BITMAP <- TRAP_debug

load & store 'debug' informations to container
- RIP, RFLAGS, RSP, branch-src

if BTF & LBR in MSR[VMX_VMCS_GUEST_DEBUGCTL_FULL]
- walking trough LBR_STACK
- pair src + dst



Freeze monitored thread & invoke user-mode tracer
- lack of deeper knowledge of VMM forced me to use ohacks :
  - vmwrite(VMX_VMCS64_GUEST_RIP, kernel_container)
  - in usermode is imidiatelly invoked PageFault
  - handle this PageFault, and set RIP to r3 injected code
  - r3 injected code is some kind of KeWaitForDbiEvent

EST_DEBUGCTL_F

```
if (!vmread(VMX_VMCS_GUEST_DEBUGCTL_FULL, &msr_btf_part))
{
    if (msr_btf_part & BTF)
    {
        for (BYTE i = static_cast<BYTE>(rdmsr(MSR_LASTBRANCH_TOS)); i >= 0; i--)
        {
            if (reinterpret_cast<const void*>(rdmsr(MSR_LASTBRANCH_0_TO_IP + i)) == vmm_exit.GetIp())
            {
                src = rdmsr(MSR_LASTBRANCH_0_FROM_IP + i);
                break;
            }
        }
```

LBR walking for branch info

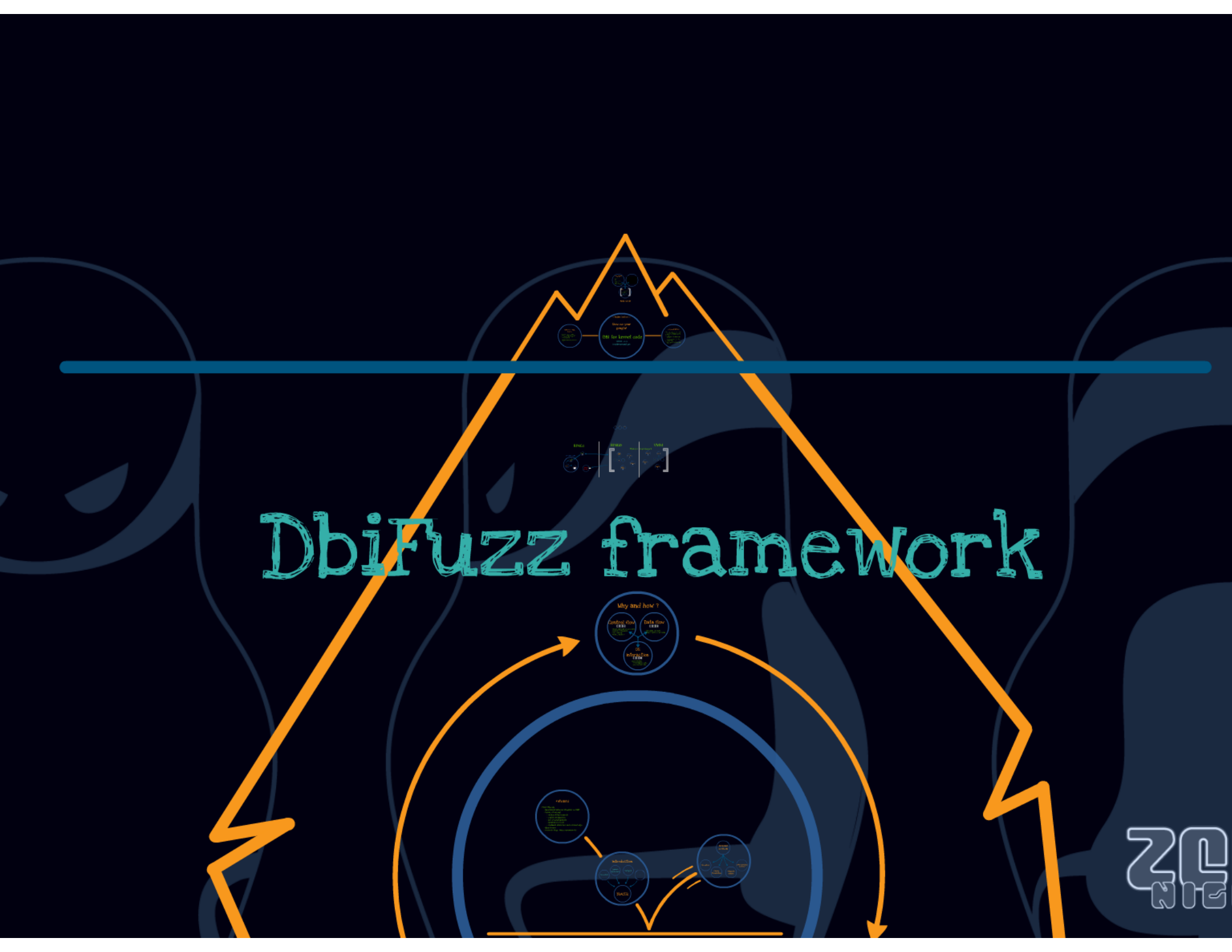
ser-mode tracer

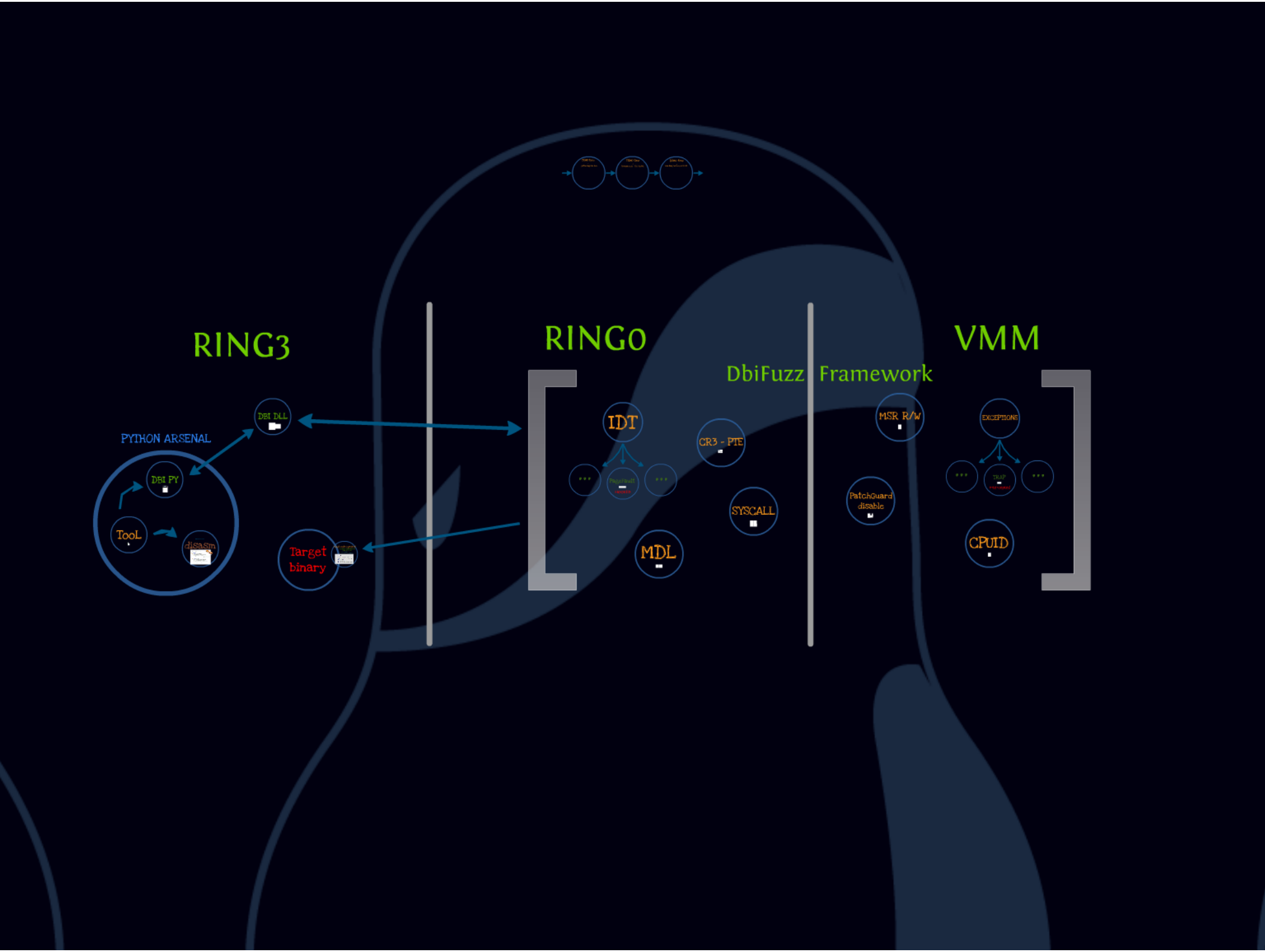
Cons of current implementation

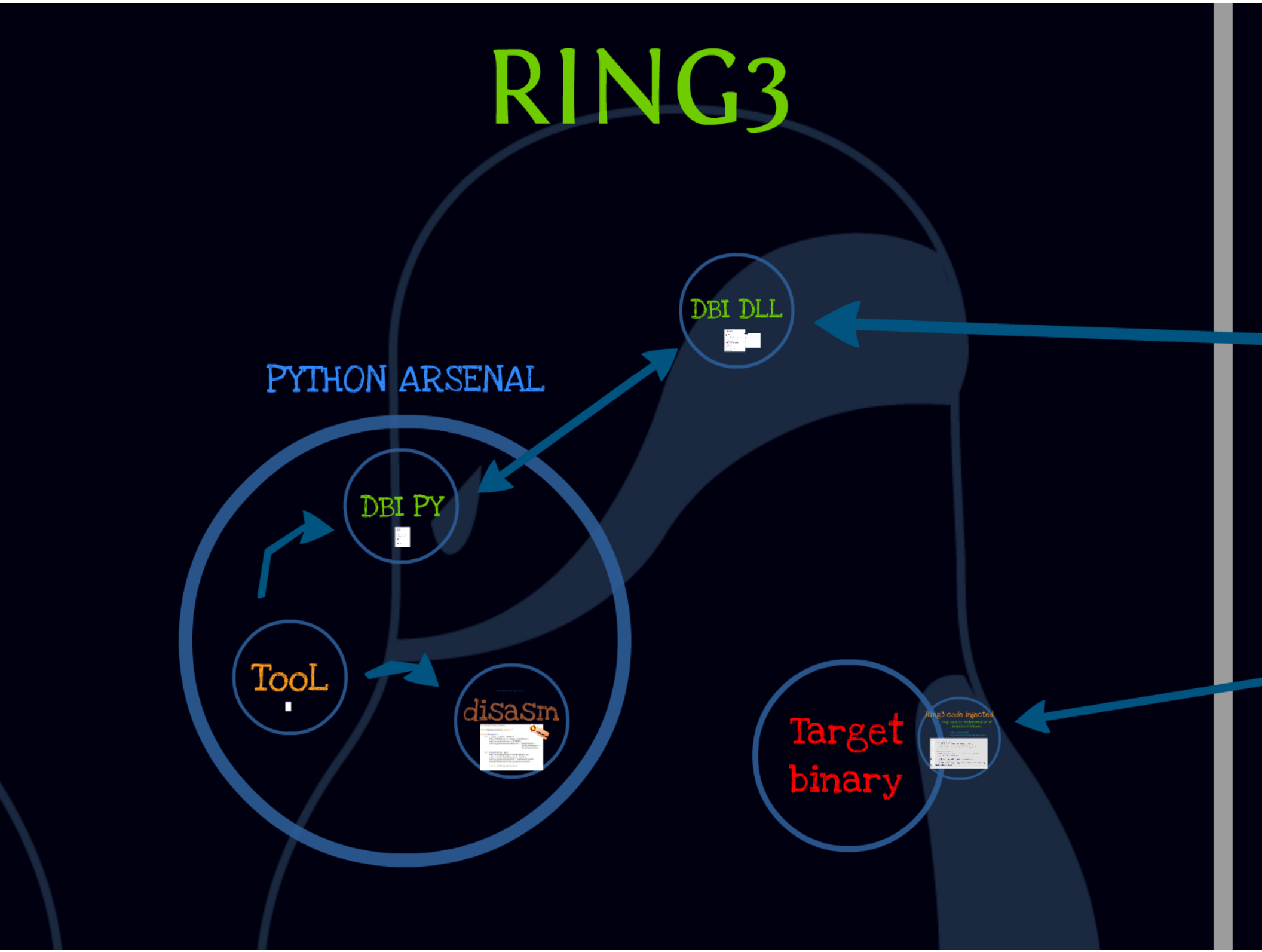
- EPT is not implemented yet
  - IDT[PageFault] hook implemented
- implemented for win8CP ... but ...
  - all system dependency is in Undoc.hpp
  - check CUndoc::Init() for compatibilty
- Lack of deeper knowledge of VMM
  - Per BTF-VMM Exit is invoked also PageFault
  - Piece of r3code 'injected' to target binary address space

Overall features

- MultiProcess
- MultiThreading
- x64 binaries
- Fast calls for switching CPL (UM – SV – HV)
- Modul for USER-MODE tracer
- Keeping target binary untouched by DBI as much as possible


DbiFuzz framework
Why and how ?
Control flow
Data flow
DBI for kernel code


RING3
RING0
VMM
DbiFuzz Framework
PYTHON ARSENAL
DBI DLL
DBI PY
TooL
disasm
Target binary
IDT
CR3 - PTE
SYSCALL
MDL
MSR R/W
PatchGuard disable
EXCEPTIONS
CPUID


RING3
DBI DLL
PYTHON ARSENAL
DBI PY
TooL
disasm
Target
binary

Target binary

Ring3 code inje

ring3 code as implemer
KeWaitForDbiEv

- only 5 instructions
- lazy injection method App

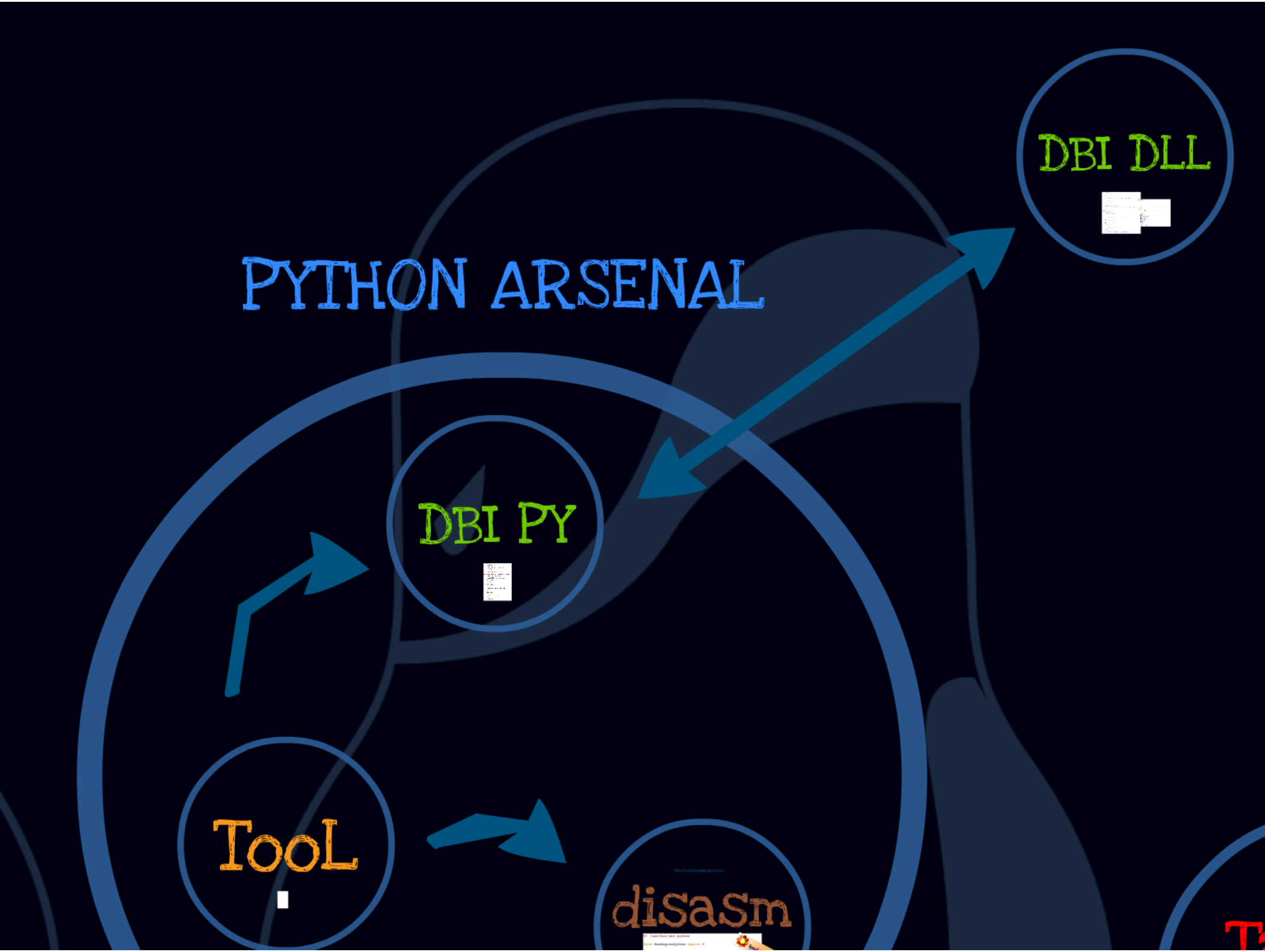
```
1.
2.  ExtTrapTrace proc
3.      ;sub rsp, 5 * sizeof(qword) ; IRETQ
4.      ;sub rsp, 010h * sizoef(qword) ; popaq
5.      ;sub rsp, 1 * sizeof(qword) ; semaphore
6.
7.  _WaitForFuzzEvent:
8.      cmp byte ptr[rsp], 0    ; thread friend
9.      jz _WaitForFuzzEvent
10.
11.     add rsp, sizeof(qword) ; semaphore
12.     popaq ; load context
13.     iretq ; set rip & rsp and continue with
14. ExtTrapTrace endp
15.
```

Ring3 code injected

ring3 code as implementation of KeWaitForDbiEvent

- only 5 instructions
- lazy injection method AppInit_DLLs

```
1.
2.  ExtTrapTrace proc
3.      ;sub rsp, 5 * sizeof(qword) ; IRETQ
4.      ;sub rsp, 010h * sizoef(qword) ; popaq
5.      ;sub rsp, 1 * sizeof(qword) ; semaphore
6.
7.  _WaitForFuzzEvent:
8.      cmp byte ptr[rsp], 0    ; thread friendly :P
9.      jz _WaitForFuzzEvent
10.
11.     add rsp, sizeof(qword) ; semaphore
12.     popaq ; load context
13.     iretq ; set rip & rsp and continue with tracing
14. ExtTrapTrace endp
15.
```


PYTHON ARSENAL
DBI DLL
DBI PY
TooL
disasm

DBI DLL

```
void SmartTrace(
    __in HANDLE procId,
    __in HANDLE threadId,
    __inout DBI_OUT_CONTEXT* dbiOut
    )
{
    FastCallMonitorWait(SYSCALL_TRACE_FLAG, procId, threadId, dbiOut);
}

EXTERN_C __declspec(dllexport)
void GetNextFuzzThread(
    __inout CID_ENUM* cid
    )
{
    HANDLE proc_id = cid->ProcId.Value;
    HANDLE thread_id = cid->ThreadId.Value;
    FastCallMonitor(SYSCALL_ENUM_THREAD, cid->ProcId.Value, cid->ThreadId.Value, cid);
}

EXTERN_C __declspec(dllexport)
void Init(
    __in HANDLE procId,
    __in HANDLE threadId,
    __inout DBI_OUT_CONTEXT* dbiOut
    )
{
    FastCallMonitor(SYSCALL_INIT, procId, threadId, dbiOut);
}

EXTERN_C __declspec(dllexport)
void DbiDumpMemory(
    __in HANDLE procId,
    __in_bcount(size) const void* src,
    __in_bcount(size) void* dst,
    __in size_t size
    )
{
    PARAM_MEMCOPY mem_cpy;
    RtlZeroMemory(&mem_cpy, sizeof(mem_cpy));
    mem_cpy.Src.Value = src;
    mem_cpy.Dst.Value = dst;
    mem_cpy.Size.Value = size;
    FastCallMonitor(SYSCALL_DUMP_MEMORY, procId, 0, &mem_cpy);
}
```

```
fast_call_monitor_wait proc
    push rbx
    mov rbx, 0666h ;

    xor rax, rax
    push rax
    lea rax, [rsp]

    xchg rax, rcx ; rcx is volatile to syscall end is with r11 automaticly rewritten
    syscall

_WaitForFuzzEvent:
    cmp byte ptr[rsp], 0    ; thread friendly waitforevent
    jz _WaitForFuzzEvent

    pop rax

    pop rbx
    ret
fast_call_monitor_wait endp

fast_call_monitor proc
    push rbx
    mov rbx, 0666h ;

    mov rax, rcx
    syscall

    pop rbx
    ret
fast_call_monitor endp
```

DBI PY

```
    def Go(self, ip):
        self.__Context__().TraceInfo.StateInfo.IRet.Flags &= ~0x100
        self.__Context__().TraceInfo.StateInfo.IRet.Return = ip
        self.__Context__().TraceInfo.Btf = 0
        return self.__Step()

    def SingleStep(self, ip):
        self.__Context__().TraceInfo.StateInfo.IRet.Flags |= 0x100
        self.__Context__().TraceInfo.StateInfo.IRet.Return = ip
        self.__Context__().TraceInfo.Btf = 0
        return self.__Step()

    def BranchStep(self, ip):
        self.__Context__().TraceInfo.StateInfo.IRet.Flags |= 0x100
        self.__Context__().TraceInfo.StateInfo.IRet.Return = ip
        self.__Context__().TraceInfo.Btf = 1
        return self.__Step()

#thread non-specific == affect all threads! -> should be implemented as thread specific!!!
    def SetAddressBreakpoint(self, ip):
        hook = PARAM_HOOK(ip)
        self.DbiSetHook(self.m_cid.ProcId, pointer(hook))

    def SetMemoryBreakpoint(self, mem, size):
        mem2watch = PARAM_MEM2WATCH(mem, size)
        self.DbiWatchMemoryAccess(self.m_cid.ProcId, pointer(mem2watch))

#thread non-specific -> memory access
    def NextMemory(self, mem):
        mem_enum = MEMORY_ENUM(mem, 0, 0)
        self.DbiEnumMemory(self.m_cid.ProcId, pointer(mem_enum))
        if (mem_enum.Begin == mem):
            return None
        return mem_enum

    def ReadMemory(self, mem, size):
        buff = BYTE_BUFFER()
        self.DbiDumpMemory(self.m_cid.ProcId, mem, pointer(buff), size)
        return buff.Bytes

    def WriteMemory(self, mem, buff, size):
        bbuff = BYTE_BUFFER(buff)
        self.DbiPatchMemory(self.m_cid.ProcId, mem, pointer(bbuff), size)

#thread non-specific -> modules
    def GetModule(self, moduleName):
        discovered = []
        moduleName = moduleName.lower()

        img = self.NextModule(0)
        while (img.Begin not in discovered):
            discovered.append(img.Begin)
            if (moduleName in img.ImageName.lower()):
                return img
            img = self.NextModule(img.Begin)lf.m_cid.ProcId, pointer(proc))
        return proc.ApiAddr

    def ReadPtr(self, mem):
        buff = self.ReadMemory(mem, 0x100)
        ptr = 0
        for i in range(0, 0x8):
            ptr |= (buff[i] << (8 * i))
        return ptr
```

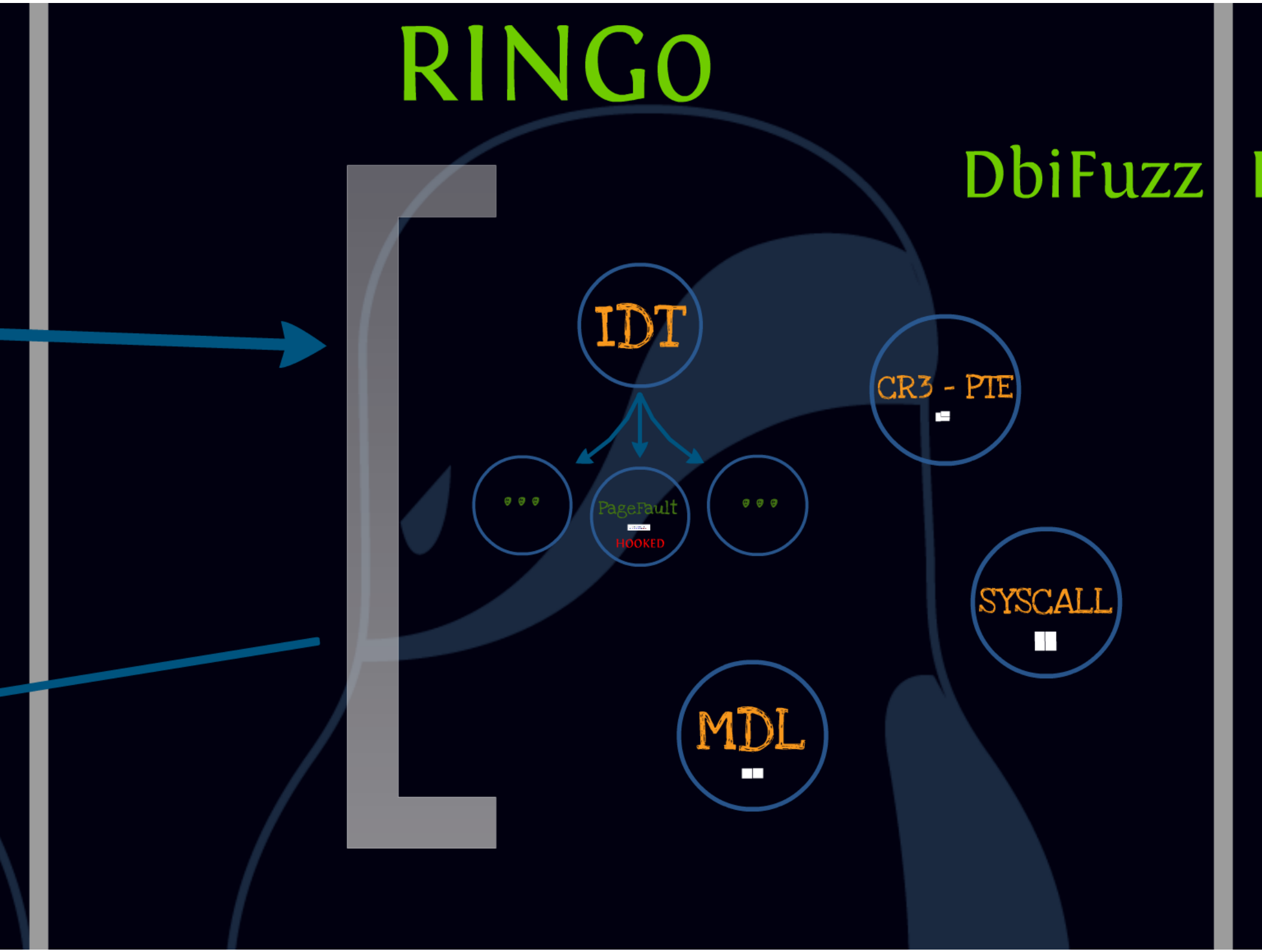
http://www.beaengine.org/

disasm

```
#! /usr/bin/env python

from BeaEnginePython import *

class CDisasm():
    def __init__(self, dumper):
        self.ReadMemory = dumper.ReadMemory
        self.m_instruction = DISASM()
        self.m_instruction.Options = MasmSyntax +
                                     SuffixedNumeral +
                                     ShowSegmentRegs

    def Disasm(self, ip):
        self.m_instruction.VirtualAddr = ip
        code = self.ReadMemory(ip, 0x100)
        self.m_instruction.EIP = addressof(code)
        Disasm(addressof(self.m_instruction))

        return self.m_instruction
```

ToOL

```
#! /usr/bin/env python

from DbiFuzzTracer import *
from Disasm import *

def main(pid):
    print("main start {")

    tracer = CDbiFuzzTracer(pid)

    tid = 0
    for i in range(0, 0x10):
        tid = tracer.GetNextThread(tid)
        if (not tid):
            print("no thread for trace")
            return
        break

    mem = tracer.NextMemory(0)
    for i in range(0, 0x10):
        if (mem.Begin > 0xFFFFFFFF):
            break

        if (None == tracer.GetModuleByAddr(mem.Begin)):
            print("non module mem : ", hex(mem.Begin), " ",
                  hex(mem.Size), " ", hex(mem.Flags))

            tracer.SetMemoryBreakpoint(mem.Begin, mem.Size)
            mem = tracer.NextMemory(mem.Begin)
        else:
            break
    return

    dis = CDisasm(tracer)
    target = tracer.GetModule("codecoverme.exe").Begin
    for i in range(0, 50):

        if (target.Begin > tracer.GetIp() or
            target.Begin + target.Size < tracer.GetIp()):

            tracer.SetAddressBreakpoint(tracer.ReadPtr(tracer.GetRsp()))
            tracer.Go(tracer.GetIp())
            print("HOOKED")

        inst = dis.Disasm(tracer.GetIp())
        print(hex(inst.VirtualAddr), " : ", inst.CompleteInstr)

        tracer.BranchStep(tracer.GetIp())

        if (MemoryAccess == tracer.GetReason()):
            print("not in module memory accesed : ",
                  tracer.GetMemoryAccesInfo().Memory)
            break

    return

main(0xef8)
```

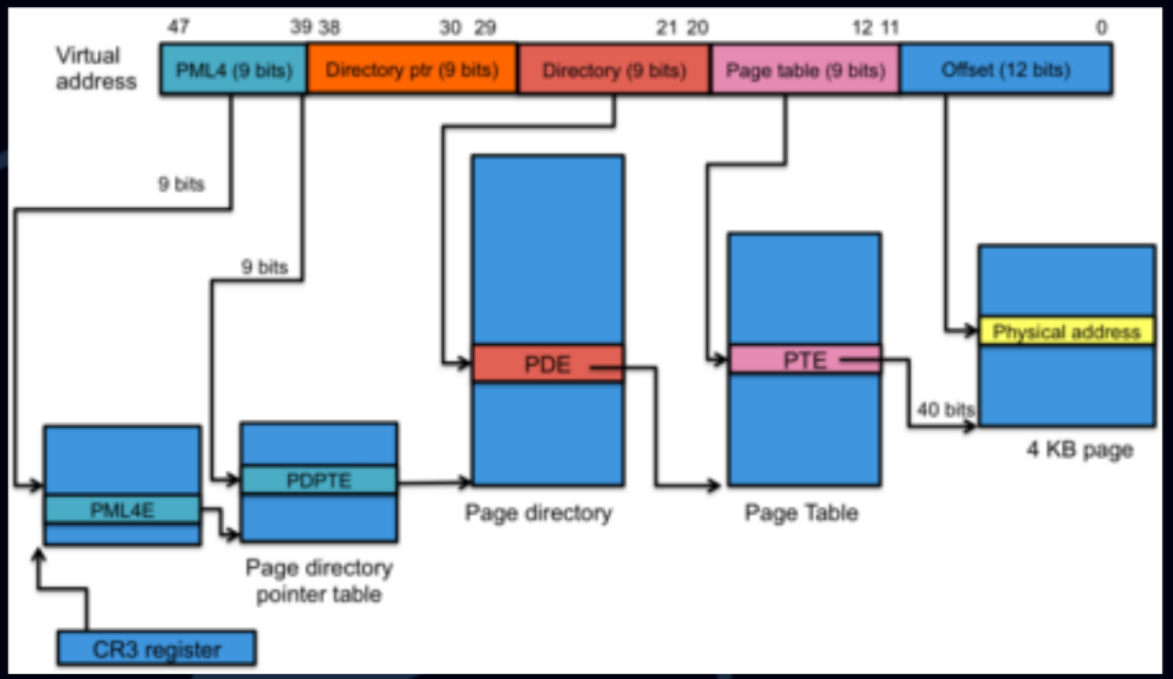

RINGo
DbiFuzz
IDT
CR3 - PTE
PageFault
HOOKED
SYSCALL
MDL

PageFault

HOOKED

```
    // } ************************** HANDLE HV TRAP EXIT **************************

    // { ************************** HANDLE PROTECTED MEMORY ACCESS **************************
    CMemoryRange* mem;
    if (m_mem2watch.Find(CMemoryRange(faultAddr, sizeof(ULONG_PTR)), &mem))
    {
        fuzz_thread->RegisterMemoryAccess(reg, faultAddr, mem, pf_iret);
        pf_iret->IRet.Return = const_cast<void*>(m_extRoutines[ExtWaitForDbiEvent]);
        return true;
    }
    // } ************************** HANDLE PROTECTED MEMORY ACCESS **************************
}
```

CR3 - PTE




```
struct VIRTUAL_ADDRESS
{
    union
    {
        void* Address;
        struct
        {
            ULONG_PTR ByteOffset : PAGE_SHIFT;
            ULONG_PTR PTESelector : 9;
            ULONG_PTR PTSelector : 9;
            ULONG_PTR PDPSelector : 9;
            ULONG_PTR PML4Selector : 9;
        } Selector;
        struct
        {
            ULONG_PTR ExtendedByteOffset : 21;
            ULONG_PTR PTSelector : 9;
            ULONG_PTR PDPSelector : 9;
            ULONG_PTR PML4Selector : 9;
        } LPSelector;
    };
};

struct PAGE_TABLE_ENTRY
{
    ULONG_PTR Valid : 1;
    ULONG_PTR Write : 1;
    ULONG_PTR Owner : 1;
    ULONG_PTR WriteTrough : 1;
    ULONG_PTR CacheDisabled : 1;
    ULONG_PTR Accessed : 1;
    ULONG_PTR Dirty : 1;
    ULONG_PTR LargePage : 1;
    ULONG_PTR Global : 1;
    ULONG_PTR SWCopyOnWrite : 1;
    ULONG_PTR SWPrototypePTE : 1;
    ULONG_PTR SWWrite : 1;
    ULONG_PTR PageFrameNumber : 28;
    ULONG_PTR Reserved : 12;
    ULONG_PTR SWWorkingSetIndex : 11;
    ULONG_PTR NoExecute : 1;
};
```

```
CMMU(
    __in const void* address
    ) : m_va(*reinterpret_cast<const VIRTUAL_ADDRESS*>(&address)),
        m_pml4(readcr3() + m_va.Selector.PML4Selector * sizeof(void*), sizeof(PAGE_TABLE_ENTRY)),
        m_pdp(GetNextTable(PML4(), m_va.Selector.PDPSelector), sizeof(PAGE_TABLE_ENTRY)),
        m_pt(GetNextTable(PDP(), m_va.Selector.PTSelector), sizeof(PAGE_TABLE_ENTRY)),
        m_pte(GetNextTable(PT(), m_va.Selector.PTESelector), sizeof(PAGE_TABLE_ENTRY))
{
}

__forceinline
__checkReturn
const void* GetNextTable(
    __in const PAGE_TABLE_ENTRY* table,
    __in size_t selector
    )
{
    if (!table)
        return NULL;

    return reinterpret_cast<const void*>((table->PageFrameNumber << PAGE_SHIFT) + selector * sizeof(void*));
}
```

http://www.cs.rutgers.edu/~pxk/416/notes/images/09a-ia32e-paging-4K.png

MDL

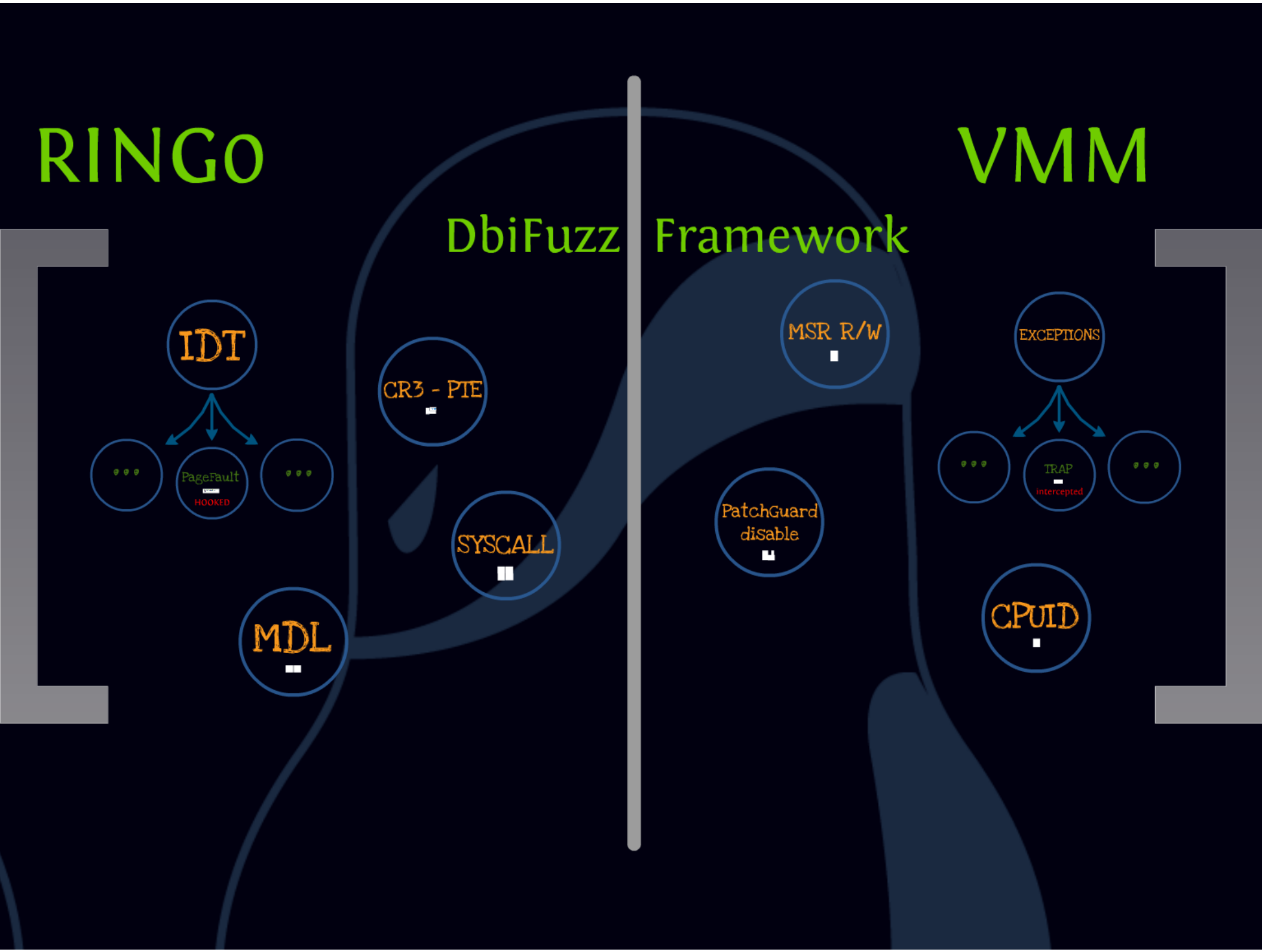
```
__checkReturn
bool CProcess2Fuzz::DbiDumpMemory(
    __inout ULONG_PTR reg[REG_COUNT]
    )
{
    PARAM_MEMCOPY params;
    if (ReadParamBuffer<PARAM_MEMCOPY>(reg, &params))
    {
        CApcLvl irql;
        CMdl mdl_dbg(params.Dst.Value, params.Size.Value);
        void* dst = mdl_dbg.WritePtr();
        if (dst)
        {
            CEProcess eprocess(m_processId);
            CAutoEProcessAttach attach(eprocess);
            if (eprocess.IsAttached())
            {
                CMdl mdl_mntr(params.Src.Value, params.Size.Value);
                const void* src = mdl_mntr.ForceReadPtrUser();
                if (src)
                {
                    memcpy(dst, src, params.Size.Value);
                    return true;
                }
            }
        }
    }
    return false;
}
```

```
__checkReturn
bool CProcess2Fuzz::DbiPatchMemory(
    __inout ULONG_PTR reg[REG_COUNT]
    )
{
    PARAM_MEMCOPY params;
    if (ReadParamBuffer<PARAM_MEMCOPY>(reg, &params))
    {
        CApcLvl irql;
        CMdl mdl_mntr(params.Src.Value, params.Size.Value);
        const void* src = mdl_mntr.ReadPtr();
        if (src)
        {
            CEProcess eprocess(m_processId);
            CAutoEProcessAttach attach(eprocess);
            if (eprocess.IsAttached())
            {
                CMdl mdl_dbg(params.Dst.Value, params.Size.Value);
                void* dst = mdl_dbg.ForceWritePtrUser();
                if (dst)
                {
                    memcpy(dst, src, params.Size.Value);
                    return true;
                }
            }
        }
    }
    return false;
}
```

SYSCALL

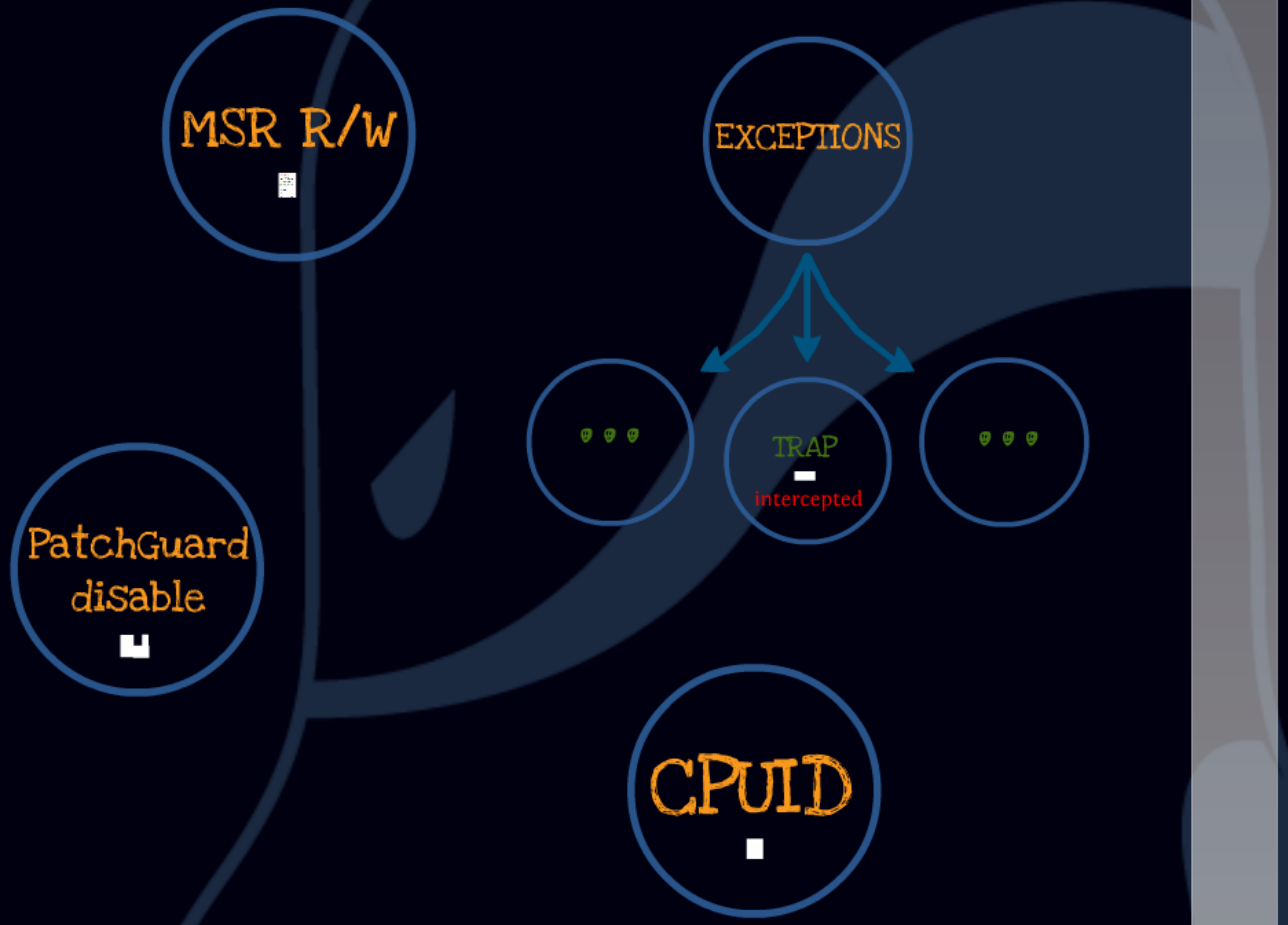
```cpp
class CSYSCALL
{
public:
    __checkReturn
    virtual bool Syscall(
        __inout ULONG_PTR reg[REG_COUNT]
        )
    {
        ULONG_PTR ring0rsp = reg[RSP];
        //-2 == simulating push ebp, pushfq to copy state as in reg[R
        reg[RSP] = reinterpret_cast<ULONG_PTR>(get_ring3_rsp() - 2);

        bool status = false;
        switch (static_cast<ULONG>(reg[RAX]))
        {
        case ntdll_NtAllocateVirtualMemory:
            status = NtAllocateVirtualMemory(reg);
            break;
        case ntdll_ZwFreeVirtualMemory:
            status = ZwFreeVirtualMemory(reg);
            break;
        case ntdll_ZwQueryVirtualMemory:
            status = ZwQueryVirtualMemory(reg);
            break;
        case ntdll_NtWriteVirtualMemory:
            status = NtWriteVirtualMemory(reg);
            break;
        case ntdll_NtReadVirtualMemory:
            status = NtReadVirtualMemory(reg);
            break;
        case ntdll_NtProtectVirtualMemory:
            status = NtProtectVirtualMemory(reg);
            break;
        case ntdll_NtFlushVirtualMemory:
            status = NtFlushVirtualMemory(reg);
            break;
        case ntdll_NtLockVirtualMemory:
            status = NtLockVirtualMemory(reg);
            break;
        case ntdll_ZwSetInformationVirtualMemory:
            status = ZwSetInformationVirtualMemory(reg);
            break;
        case ntdll_ZwUnlockVirtualMemory:
            status = ZwUnlockVirtualMemory(reg);
            break;
        default:
            break;
        }

        reg[RSP] = ring0rsp;
        return status;
    }

protected:
    virtual bool NtAllocateVirtualMemory(
        __inout ULONG_PTR reg[REG_COUNT]
    )
    {
        return false;
    }
```

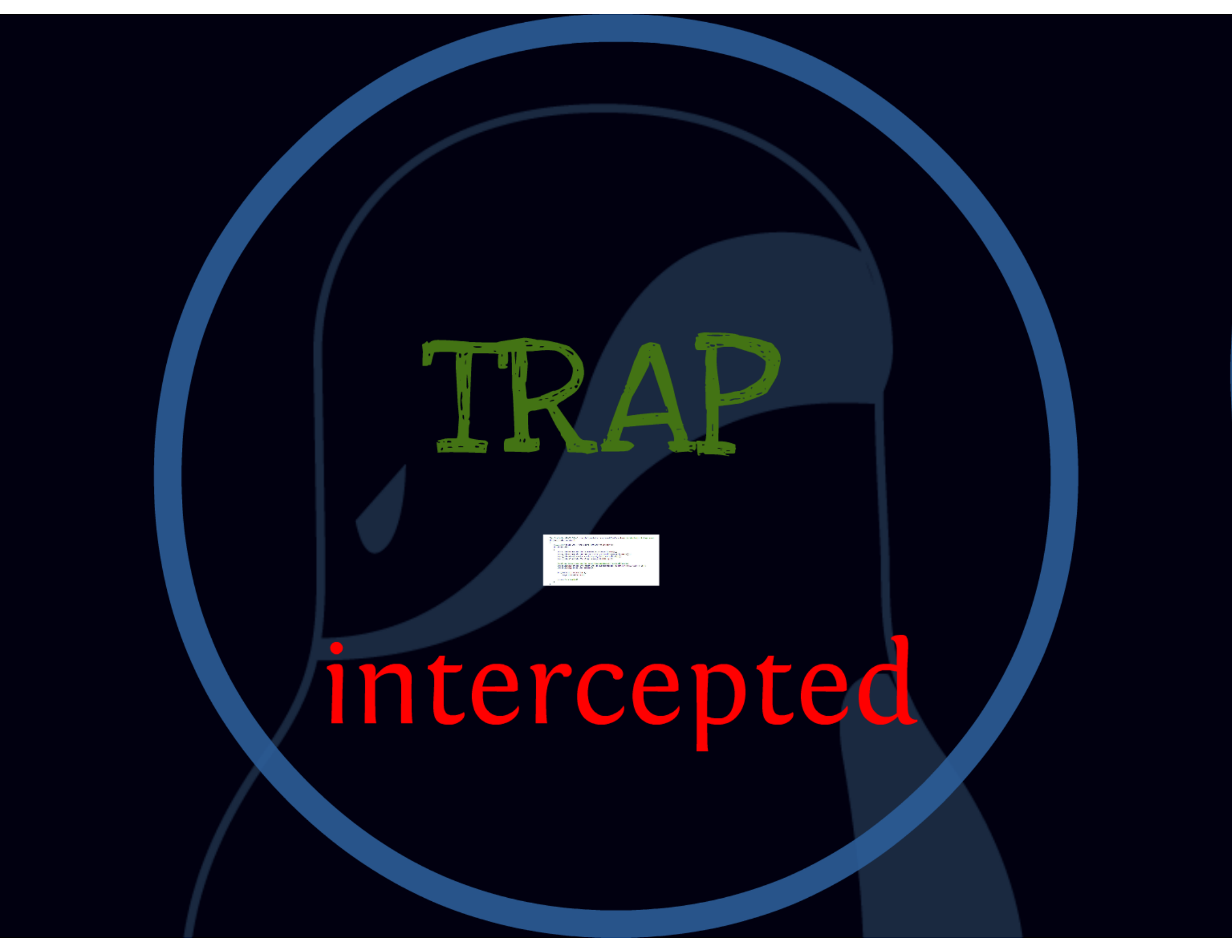
```cpp
class CSyscallCallbacks :
    public CSYSCALL
{
protected:
    __checkReturn
    virtual bool VirtualMemoryCallback(
        __in void* memory,
        __in size_t size,
        __in bool write,
        __inout ULONG_PTR reg[REG_COUNT],
        __inout_opt BYTE* buffer = NULL
        )
    {
        return false;
    }

protected:

//imeplemntation of Virtual Memory syscalls callbacks
    virtual bool NtAllocateVirtualMemory(
        __inout ULONG_PTR reg[REG_COUNT]
    ) override
    {
        return VirtualMemoryCallback(
            reinterpret_cast<void*>(*reinterpret_cast<ULONG_PTR*>(reg[RDX])),
            *reinterpret_cast<size_t*>(reg[R9]),
            !!((ULONG)*PPARAM(reg, 6) & PAGE_WR_MASK),
            reg
            );
    }
    virtual bool ZwFreeVirtualMemory(
        __inout ULONG_PTR reg[REG_COUNT]
    ) override
    {
        return VirtualMemoryCallback(
            reinterpret_cast<void*>(*reinterpret_cast<ULONG_PTR*>(reg[RDX])),
            *reinterpret_cast<size_t*>(reg[R8]),
            false,
            reg
            );
    }
    virtual bool ZwQueryVirtualMemory(
        __inout ULONG_PTR reg[REG_COUNT]
    ) override
    {
        return VirtualMemoryCallback(reinterpret_cast<void*>(reg[RDX]),
            0,
            false,
            reg
            );
    }
    virtual bool NtWriteVirtualMemory(
        __inout ULONG_PTR reg[REG_COUNT]
    ) override
    {
        return VirtualMemoryCallback(
            reinterpret_cast<void*>(reg[RDX]),
            (size_t)reg[R9],
            true,
            reg,
            reinterpret_cast<BYTE*>(reg[R8])
            );
    }
```


RINGo
VMM
DbiFuzz Framework
IDT
PageFault
HOOKED
...
...
CR3 - PTE
SYSCALL
MDL
MSR R/W
PatchGuard disable
EXCEPTIONS
TRAP
intercepted
...
...
CPUID

VMM

z Framework




TRAP
intercepted

```
CAutoTypeMalloc<TRACE_INFO>* trace_info_container = m_branchInfoQueue.Pop();//interlocked NonPage queue
if (trace_info_container)
{
    TRACE_INFO* trace_info = trace_info_container->GetMemory();
    if (trace_info)
    {
        trace_info->StateInfo.IRet.StackPointer = vmm_exit.GetSp();
        trace_info->StateInfo.IRet.Return = const_cast<void*>(vmm_exit.GetIp());
        trace_info->PrevEip.Value = reinterpret_cast<const void*>(src);
        trace_info->StateInfo.IRet.Flags = vmm_exit.GetFlags();

        //set eip to non-exec mem for quick recognization by PageFault handler
        vmm_exit.SetSp(&trace_info->StateInfo.IRet.StackPointer[-(IRetCount + REG_COUNT + 1)]);
        vmm_exit.SetIp(trace_info_container);

        if (vmm_exit.IsTrapActive())
            vmm_exit.DisableTrap();

        return;//is handled!
    }
}
```

MSR R/W

```
__checkReturn
bool CDbiMonitor::SetVirtualizationCallbacks()
{
    DbgPrint("CSysCall::SetVirtualizationCallbacks\n");

    if (!CCRonos::SetVirtualizationCallbacks())
        return false;

    m_traps[VMX_EXIT_RDMSR] = (ULONG_PTR)HookProtectionMSR;
    m_traps[VMX_EXIT_WRMSR] = (ULONG_PTR)WrMsrSpecialBTF;
    m_traps[VMX_EXIT_EXCEPTION] = (ULONG_PTR)TrapHandler;
    m_traps[VMX_EXIT_DRX_MOVE] = (ULONG_PTR)AntiPatchGuard;

    //m_traps[VMX_EXIT_EPT_VIOLATION] = (ULONG_PTR)PageFaultHandler;

    return RegisterCallback(m_callbacks, CPUIDCALLBACK);
}

//-------------------------------------------------------------
// ***************** HYPERVISOR CALLBACKS *****************
//-------------------------------------------------------------

void CDbiMonitor::WrMsrSpecialBTF(
    __inout ULONG_PTR reg[REG_COUNT]
    )
{
    //special handling for our wrmsr - enable / disable BTF
    if (IA32_DEBUGCTL == reg[RCX])
    {
        //handle just dword-low
        ULONG_PTR msr_btf_part;
        vmread(VMX_VMCS_GUEST_DEBUGCTL_FULL, &msr_btf_part);

        if (reg[RAX] & BTF)
            reg[RAX] |= msr_btf_part;//enable BTF + LBR
        else
            reg[RAX] &= msr_btf_part;//disable BTF + LBR

        vmwrite(VMX_VMCS_GUEST_DEBUGCTL_FULL, reg[RAX]);
        vmread(VMX_VMCS_GUEST_DEBUGCTL_HIGH, &reg[RDX]);
    }

    wrmsr((ULONG)reg[RCX], (reg[RDX] << 32) | (ULONG)reg[RAX]);
}
```

PatchGuard disable

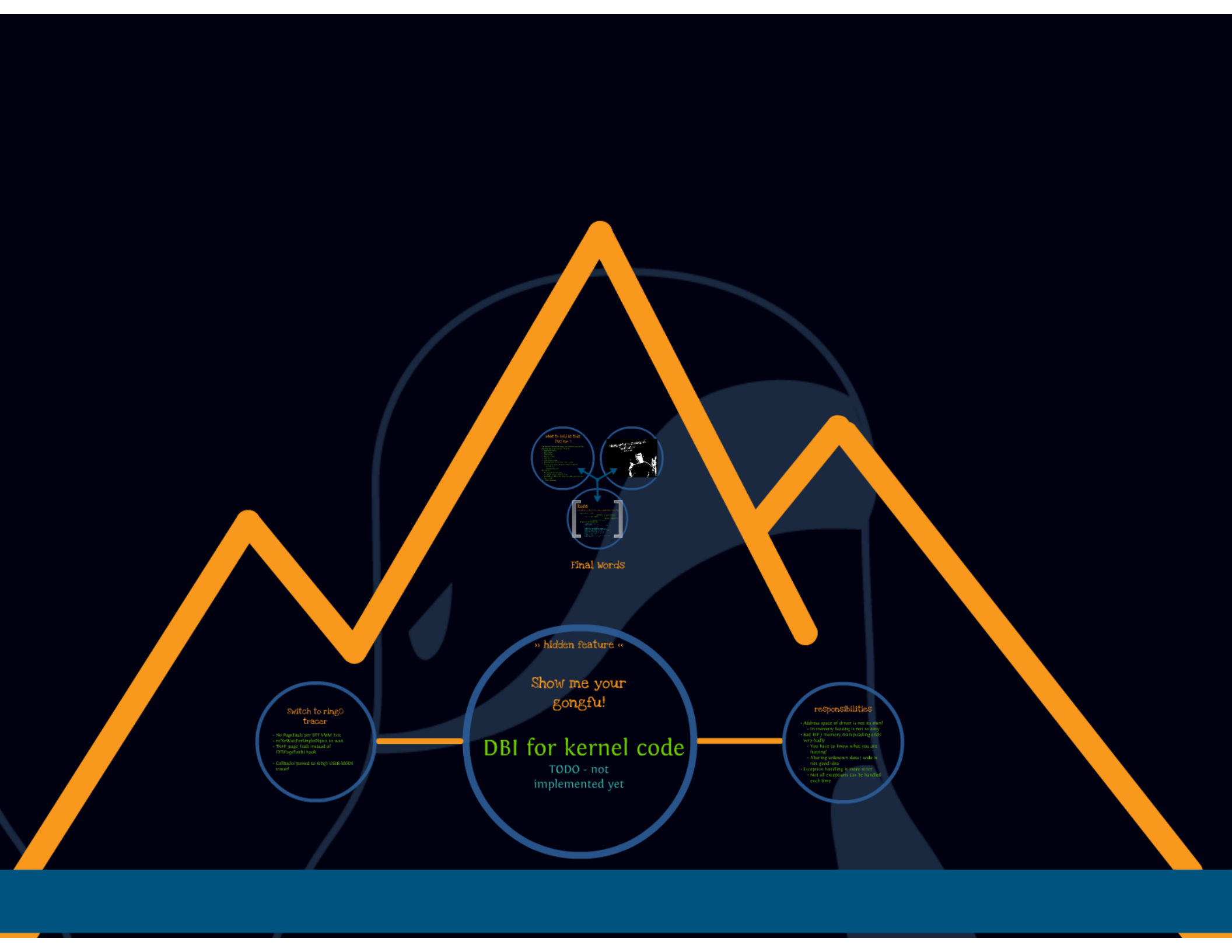
```
__checkReturn
bool CDbiMonitor::SetVirtualizationCallbacks()
{
    DbgPrint("CSysCall::SetVirtualizationCallbacks\n");

    if (!CCRonos::SetVirtualizationCallbacks())
        return false;

    m_traps[VMX_EXIT_RDMSR] = reinterpret_cast<ULONG_PTR>(VMMRDMSR);
    m_traps[VMX_EXIT_WRMSR] = reinterpret_cast<ULONG_PTR>(VMMWRMSR);
    m_traps[VMX_EXIT_EXCEPTION] = reinterpret_cast<ULONG_PTR>(VMMEXCEPTION);

    //disable patchguard
    RegisterCallback(m_callbacks, AntiPatchGuard);

    return RegisterCallback(m_callbacks, VMMCPUID);
}

//-----------------------------------------------------------------------
// ***************** HYPERVISOR PATCHGUARD DISABLING *****************
//-----------------------------------------------------------------------

void CDbiMonitor::AntiPatchGuard(
    __inout ULONG_PTR reg[REG_COUNT]
    )
{
    CVMMAutoExit vmm_exit;
    switch (vmm_exit.GetReason())
    {
    case VMX_EXIT_RDTSC:
        if (CUndoc::IsPatchGuardContextOnRTDSC(reg))
        {
            reg[RSI] = reinterpret_cast<ULONG_PTR>(vmm_exit.GetIp());
            vmm_exit.SetIpFromCallback(patchguard_hook);//.asm wrapper to PatchGuardHook
        }
        break;
    default:
        return;
    }
}

//PatchGuard - alice in wonderland  [ virtualization-based hooks ]
//-----------------------------------------------------------------------
// ***************** SUPERVISOR PATCHGUARD DISABLING *****************
//-----------------------------------------------------------------------

//I. disable patchguard
EXTERN_C size_t PatchGuardHook(
    __inout ULONG_PTR reg[REG_COUNT]
    )
{
    DbgPrint("\n >>>>>> PatchGuardHook %p [%p] %p\n\n", reg[RCX], reg, reg[RSI]);
    KeBreak();
    return CUndoc::PatchGuardContextStackTopDelta();
}
//II. hide (even for PatchGuard) SYSENTER interception
EXTERN_C void* RdmsrHook(
    __inout ULONG_PTR reg[REG_COUNT]
    )
{
    void* ret = reinterpret_cast<void*>(reg[RCX]);
    DbgPrint("\n\n---> RdmsrHook; PG active at : %p\n", ret);
    KeBreak();
    reg[RCX] = IA64_SYSENTER_EIP;
    return ret;
}
```

```
;PATCHGUARD related
rdmsr_hook proc
    ENTER_HOOK_PROLOGUE
    ENTER_HOOK RdmsrHook
    ENTER_HOOK_EPILOGUE
    ret
rdmsr_hook endp

patchguard_hook proc
    ENTER_HOOK_PROLOGUE
    ENTER_HOOK PatchGuardHook
    ENTER_HOOK_EPILOGUE

    pop r11
    add r11, rsp
    mov rbx, [r11+38h]
    mov rsi, [r11+40h]
    mov rdi, [r11+48h]
    mov rsp, r11
    pop r15
    pop r14
    pop r13
    pop r12
    pop rbp

    add rsp, 050h
    ret
patchguard_hook endp
```

```
__forceinline
static
KTIMER* PatchGuardGetPKTIMER(
    __in ULONG_PTR contextAddr
    )
{
    return *reinterpret_cast<KTIMER**>(contextAddr + m_pKTIMER);//0x330
}

__forceinline
static
size_t PatchGuardContextStackTopDelta()
{
    return m_patchGuardContextStackTopDelta;//0x7D0
}

__forceinline
static
bool IsPatchGuardContextOnRTDSC(
    __in const ULONG_PTR reg[REG_COUNT]
    )
{
    return (reg[RSI] == m_patchGuardMagic);//0x7010008004002001
}
```

CPUID

```
//__in_opt CALLBACK** callbacks == ptr to m_callbacks
void CCRonos::HVCallback(__inout ULONG_PTR reg[REG_COUNT], __in_opt CALLBACK** callbacks)
{
    if (!callbacks)
        return;

    CALLBACK* t_callbacks = *callbacks;
    if (!t_callbacks)
        return;

    while (t_callbacks->Callback != NULL)
    {
        (*(void (*)(ULONG_PTR*))(t_callbacks->Callback))(reg);
        t_callbacks = t_callbacks->Next;
    }
}

void CCRonos::HVCpuid( __inout ULONG_PTR reg[REG_COUNT] )
{
    reg[RAX] = kCpuidMark;
}


//extended hypevisor callbacks mechanism

void HVCallback1( __inout ULONG_PTR reg[REG_COUNT] )
{
    ULONG_PTR ExitReason;
    vmread(VMX_VMCS32_RO_EXIT_REASON, &ExitReason);

    if (VMX_EXIT_CPUID == ExitReason)
        reg[RDX] = kCpuidMark3;
}

void HVCallback2( __inout ULONG_PTR reg[REG_COUNT] )
{
    ULONG_PTR ExitReason;
    vmread(VMX_VMCS32_RO_EXIT_REASON, &ExitReason);

    if (VMX_EXIT_CPUID == ExitReason)
        reg[RCX] = kCpuidMark2;
}

void HVCallback3( __inout ULONG_PTR reg[REG_COUNT] )
{
    ULONG_PTR ExitReason;
    vmread(VMX_VMCS32_RO_EXIT_REASON, &ExitReason);

    if (VMX_EXIT_CPUID == ExitReason)
        reg[RBX] = kCpuidMark1;
}
```


Final words
›› hidden feature ‹‹
Show me your gongfu!
DBI for kernel code
TODO - not implemented yet
Switch to ring0 tracer
responsibilities

Final words

›› hidden feature ‹‹

Show me your gongfu!

DBI for kernel code

TODO - not implemented yet

Switch to ring0 tracer

- No PageFault per BTF-VMM Exit
- nt!KeWaitForSingleObject to wait
- TRAP_page_fault instead of IDT[PageFault] hook

- Callbacks passed to Ring3 USER-MODE tracer!

responsibilities

- Address space of driver is not its own!
  - In-memory fuzzing is not so easy
- Bad RIP / memory manipulating ends very badly
  - You have to know what you are fuzzing!
  - Altering unknown data / code is not good idea
- Exception handling is more strict
  - Not all exceptions can be handled each time


›› hidden feature ‹‹
Show me your gongfu!
DBI for kernel code
TODO - not implemented yet

Switch to ring0 tracer

- No PageFault per BTF-VMM Exit
- nt!KeWaitForSingleObject to wait
- TRAP_page_fault instead of IDT[PageFault] hook

- Callbacks passed to Ring3 USER-MODE tracer!

responsibilities

- Address space of driver is not its own!
  - In-memory fuzzing is not so easy
- Bad RIP / memory manipulating ends very badly
  - You have to know what you are fuzzing!
  - Altering unknown data / code is not good idea
- Exception handling is more strict
  - Not all exceptions can be handled each time

What to hell is this PoC for ?

- Not finished, and may be buggy. Not really for use yet, but ..
- Implemented various classes – features
  - PageTable walker
  - VAD walker
  - MDL worker
  - Process monitor
  - AutoLocks
  - Lightweight VMM
  - Implemented CPU features -> Trap + BTF
  - Various easy-to-use containers based on msdn
    - Locked AVL
    - Interlocked Stack
- IDEA behind
  - Dbi based in kernel mode
  - as module and accesible from .py
  - Possibility to link with IDA-python, other python(other lng) arsenal
  - + hidden feature!

Refs

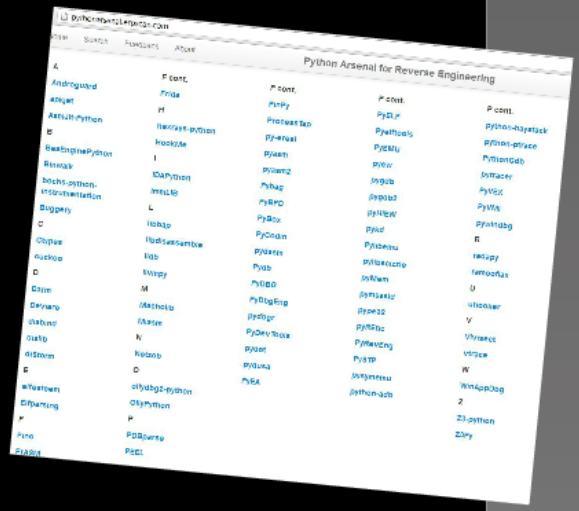
https://github.com/zerosum/ShowMeYourGong3u/tree/xoiuserland

- @gynvael [ExcpHook]
- @Ivanlefou [Hypervisor Abyss]
- @IntelLabs [manuals]
- @pedramamini [BTF]
- @aionescu – VAD related
- @reactos [VAD, OS internals]

Final Words


“Simplicity is the key to brilliance”
Bruce Lee
Photo By Thomas Duchnicki :: Location Scout

just as module



- DbiFuzz as tracer
- IDA Pro and python as Binary info
  - Graph info
  - Disasm
  - Functions, params frame, vars frame
- dbghelp.dll as symbols
- Clang as LLVM
- Python / c++ ... based

Python binding to Z3

This is a Python binding to the SMT solver Z3. Since it is based on Python's dynamic foreign function interface ctypes, no compilation is required. For additions, corrections or suggestions, please contact me.

5- How to use the library with Python ?

BeaEngine can be used under Python thanks to the ctypes library.

```
from BeaEnginePython import *

Instruction = DISASM()
```

BeaEngine

IDAPython in a Nutshell

IDAPython is an IDA Pro plugin that integrates the Python programming language, allowing scripts to run in IDA Pro. These programs have access to IDA Plugin API, IDC and all modules available for Python. The power of IDA Pro and Python provides a platform for easy prototyping of reverse engineering and other research tools.

What to hell is this PoC for ?

- Not finished, and may be buggy. Not really for use yet, but ..
- Implemented various classes – features
  - PageTable walker
  - VAD walker
  - MDL worker
  - Process monitor
  - AutoLocks
  - Lightweight VMM
  - Implemented CPU features -> Trap + BTF
  - Various easy-to-use containers based on msdn
    - Locked AVL
    - Interlocked Stack
- IDEA behind
  - Dbi based in kernel mode
  - as module and accesible from .py
  - Possibility to link with IDA-python, other python(other lng) arsenal
  - + hidden feature!

Refs

https://github.com/zer0mem/ShowMeYourGongFu/tree/x64userland

- @gynvael [ExcpHook]
- @Ivanlef0u [Hypervisor Abyss]
- @IntelLabs [manuals]
- @pedramamini [BTF]
- @aionescu – VAD related
- @reactos [VAD, OS internals]

http://gynvael.coldwind.pl/?id=148
http://www.ivanlef0u.tuxfamily.org/?p=120
http://www.openrce.org/blog/view/535/Branch_Tracing_with_Intel_MSR_Registers

http://linux.linti.unlp.edu.ar/images/f/f1/Vtx.pdf
http://download.intel.com/products/processor/manual/326019.pdf
http://download.intel.com/products/processor/manual/253669.pdf
http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf
http://www.dfrws.org/2007/proceedings/p62-dolan-gavitt.pdf
http://technet.microsoft.com/en-us/sysinternals/bb963901.aspx
http://fdbg.x86asm.net/hdbg/hdbg.html
http://code.google.com/p/hyperdbg/
http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf
http://beatrix2004.free.fr/BeaEngine/index1.php