基于Unicorn-Engine的开源Windows可执行文件沙盒实现解析

王伟波 | Comodo反病毒引擎研发技术负责人，FreeBuf年度作者

FIT · FREEBUF

目 录

虚拟化软件的比较以及各自的特点

- 沙箱,是一个比较宽泛的概念，总的来说沙箱的特性就是,在一个系统中隔离运行不可信的代码(例如, Sandboxie, Shade sandbox).
- 虚拟机，提供了一个虚拟出来的硬件环境，在宿主机之上运行客户操作系统(例如,VirtualBox, VMware, QEMU).
- CPU模拟器,利用软件来模拟目标CPU指令运行,比如unicorn-engine,libemu等.
- 不管是沙箱，虚拟机，模拟器都可以用在反恶意软件的战斗中.

下图是典型的虚拟机和CPU模拟器软件的一个实例的比较:

	Approach	Performance	Execution granularity	System fidelity
VirtualBox	HardWareVirtualization (VT/AMD-V)	Quick ,heavily	N/A	good
Bochs	Full emulation	Slow, lightweight	Instruction level	good
Uniron-engine	Emulation/dynamic translation	Quick, lightweight	Instruction level	N/A

虚拟化软件的比较以及各自的特点

在处理样本时,沙箱,虚拟机的缺点:

- 沙箱(sandboxie)适合普通用户使用, 不能观察到样本的具体执行过程。
- 虚拟机运行需要花费很多的资源, 每次运行都还原操作系统，代价很大.
- 在资源有限的情况下,对于处理大批量样本, 虚拟机显得捉襟见肘.
- 沙箱,虚拟机执行恶意样本, 本身不能获取样本指令级别的行为

获取样本指令级行为有何用?

- 比如说对于数据挖掘,获取样本指令执行序列来判断,分类样本是很有用.

可参考:

<<DATA MINING METHODS FOR MALWARE DETECTION USING INSTRUCTION SEQUENCES >>

虚拟化软件的比较以及各自的特点

如何能够利用少量资源,批量获取样本运行行为?
如何能更全面的获取样本的行为?

idea: 一个CPU模拟器+windows仿真环境.

unicorn-engine介绍

●unicorn-engine是一个基于QEMU的cpu模拟执行框架,支多平台(WINDOWS,*NIX),
多指令集(ARM, ARM64, M68K, MIPS, SPARC, and X86 (16, 32, 64-bit)).
●除过C语言本地直接调用之外,支持很多种语言的接口调用(如:VB,Perl,Rust,Python,Go).
●基于JIT编译技术,所以运行速度很快.
●相比其他CPU模拟器相比,有很大的优势.

http://www.unicorn-engine.org/

unicorn-engine介绍

Why unicorn-engine?

- 接口很方便.
- 长期维护.
- 性能很好.
- 指令支持的非常全.

Features	libemu	PyEmu	IDA-x86emu	libCPU	Unicorn
Multi-arch	X	X	X	X	✓
Updated	X	X	X	X	✓
Independent	X	X	X	✓	✓
JIT	X	X	X	✓	✓

- Multi-arch: existing tools only support X86
- Updated: existing tools do not supports X86_64

结论: wxemu == unicorn-engine+windows 仿真环境 ☺

unicorn-engine介绍

unicorn-engine 提供了:

- 内存读写功能（uc_mem_map.uc_mem_map_ptr, uc_mem_read,uc_mem_write).
- cpu寄存器读写功能.
- 指令执行hook,内存读写hook,代码基本块执行hook,内存错误,指令执行错误hook等,中断/系统调用指令hook.

显然,已经提供的这些功能很容易用来执行shellcode,
但是作为一个window的模拟器的很重要的基础设施,unicorn-engine 需要做一些处理.

Hack unicorn-engine

1.移除一些无关的代码

unicorn-engine 目前支持好几种架构的指令集,wxemu主要是处理X86平台的文件,所以其他几种指令执行模块可以移除掉. 这样不仅可 以减小文件大小,而且会加快执行速度.

2.增加地址转换api

- Unicorn-engine 提供了简单的内存管理接口, uc_mem_write, uc_mem_map_ptr,
- 这两个api比较适合,较大段内存的写入.
- 对于只知道起始地址,又需要进行连续的内存写操作,相对来说麻烦的多.
- 初始化目标程序运行环境的时候这类操作又非常的多.
- 所以必须增加一个guest地址到host地址的转换,和host地址到guest地址的转换的api.
- 这样就可以直接操作guest内存,就像操作host内存一样了(getraddr,getvaddr).

处理前	处理后
3000KB	~1800KB

Hack unicorn-engine

Unicorn/qemu/exec.c

●函数qemu_ram_alloc_from_ptr能够获得申请的内存块host起始地址(new_block->host)

●通过在结构体MemoryRegion中增加一个成员hostaddr,将host内存块起始值传给它.

●从而通过相对内存偏移+host内存块起始位置的方式来获取host地址.

```
struct MemoryRegion {
    Object parent_obj;
     ....
    NotifierList iommu_notify;
    struct uc_struct *uc;
    uint32_t perms;   //all perms, partially redundant with
readonly
    uint64_t end;
    uint64_t hostaddr;
};
```

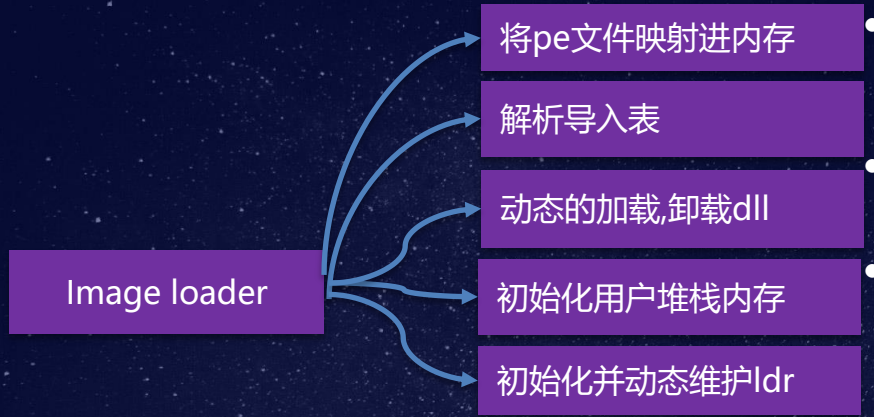
```
1083  // return -1 on error
1084  ram_addr_t qemu_ram_alloc_from_ptr(ram_addr_t size, void *host,
1085          MemoryRegion *mr, Error **errp)
1086  {
1087      RAMBlock *new_block;
1088      ram_addr_t addr;
1089      Error *local_err = NULL;
1090
1091      size = TARGET_PAGE_ALIGN(size);
1092      new_block = g_malloc0(sizeof(*new_block));
1093      if (new_block == NULL)
1094          return -1;
1095
1096      new_block->mr = mr;
1097      new_block->length = size;
1098      new_block->fd = -1;
1099      new_block->host = host;
1100      if (host) {
1101          new_block->flags |= RAM_PREALLOC;
1102      }
1103      addr = ram_block_add(mr->uc, new_block, &local_err);
1104      if (local_err) {
1105          mr->hostaddr=NULL;
1106          g_free(new_block);
1107          error_propagate(errp, local_err);
1108          return -1;
1109      }
1110    mr->hostaddr=new_block->host;
1111      return addr;
1112  }
1113
```

wxemu的设计与实现

如何围绕可执行文件,构造windows仿真环境?

- 一个image loader.
- 内存管理.
- 对象管理.
- 文件系统.
- 注册表系统.
- 异常处理.
- 实现大量的,常见的win32 api.

wxemu的设计与实现-image loader



- 加载系统dll,可以用系统本身的dll的吗?
  No.这里加载的是虚拟win32 dll.是在初始化的过程中生成的.
  真正的api实现部分是在wxhelper中的.

- 为何只生成API stub,而不直接在所在原始文件中模拟api?
  1.方便
  2.为了host 层可以对api的执行可控

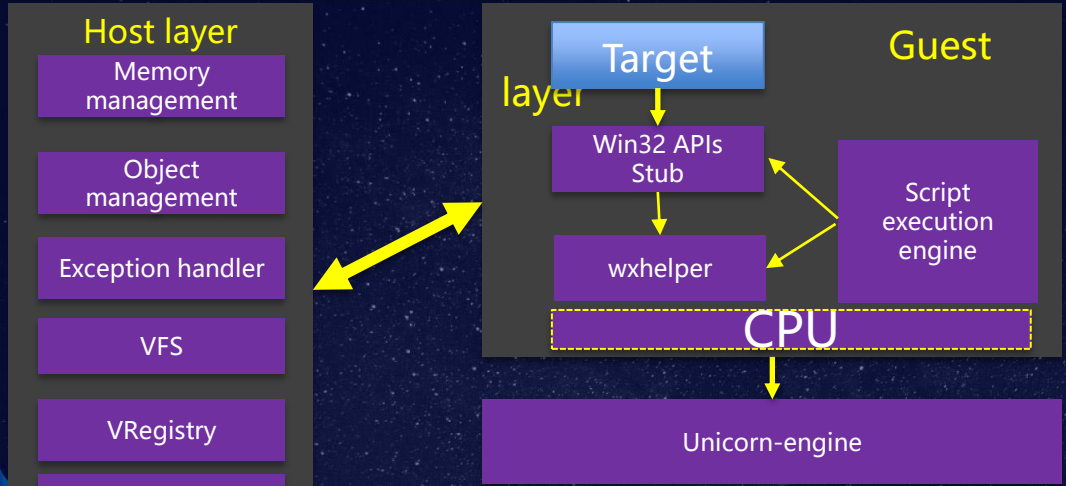
- 当动态加载一个dll的时候,该怎么更新Ldr结构体的数据?
  更新ldr结构,这里涉及读写guest内存操作,而且是链表结构,
  如果只用unicorn-engine提供的接口,效率会很低.所以这里会获取
  ldr链表结构host地址,然后直接进行链表插入.

wxemu的设计与实现-image loader

```
if (strcmp(filename, "C:\\Users\\Maldiohead\\Maldiohead.exe"))
{
    ent->Flags = LDRP_IMAGE_DLL;
    vlistinserttail(0x7ffd0000+ ININITORD,&lldr->InInitializationOrderModuleList, &ent->InInitializationOrderModuleList, vent + ENT_ININITORD);
}
vlistinserttail(0x7ffd0000+ INLOADORD,&lldr->InLoadOrderModuleList, &ent->InLoadOrderLinks, vent + ENT_INLOADORD);
vlistinserttail(0x7ffd0000+ INMEMORD,&lldr->InMemoryOrderModuleList, &ent->InMemoryOrderModuleList,vent+ ENT_INMEMORD);
```

```
__INLINE__
void vlistinserttail(uint32_t vlisthead, PLIST_ENTRY rlisthead, PLIST_ENTRY entry, uint32_t ventry)
{
    PLIST_ENTRY OldBlink;
    PLIST_ENTRY ROldBlink;
    OldBlink = rlisthead->Blink;/*OldBlink is vaddr*/
    entry->Flink = vlisthead;
    entry->Blink = OldBlink;
    wxgetraddr(OldBlink, &ROldBlink);
    ROldBlink->Flink = ventry;
    ROldBlink->Blink = ventry;
}
```

wxemu的设计与实现-架构



Wxemu 架构图

wxemu的设计与实现-对象管理

进程对象
线程对象
文件对象
注册表对象

对象管理的目的:
1.遵循面向对象的编程原则
2.提供一个公用且统一的对象操作界面.
3.提供快速,安全,简单的对象创建,访问,释放机制(利用句柄).

```
enum wxobject_type {
        PS,
        THRD,
        FILE,
        REGISTRY,
        DIR,
};


  struct wxobject
  {
        char*   objname;
        enum    wxobject_type type;
        int     object_size;
        struct wxobject_ops *ops;
  };
```

wxemu的设计与实现-对象管理

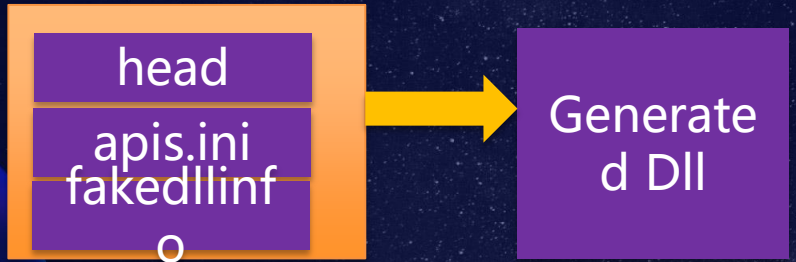
wxemu进程对象:
1.拥有对象的基本属性.
2.拥有Win32 进程对象的一些基本元素(pid,peb,线程对象等).
3.包含虚拟注册表对象,虚拟文件系统对象.

- 在host layer的代码中,process对象是一个非常核心的对象
- 生存周期和样本运行周期一样.

```
struct  _process
{
    struct wxobject;
    struct _subprocess* subprocess;
    uint32_t   pid;
    uint32_t   ppid;
    PEB32      *peb;
    uint64_t   inscount;
    uint8_t    *builtindll;
    uint8_t    *cur_dir;
    bool_t     (*psfree)();
    struct  list modulelist;
    struct  list threadlist;
    struct  thread *running_thread;
    struct  regvfs *regvfs;
    struct  vfs *vfs;
    #define GETALLAPISEQ   1
    #define GETMAINAPISEQ  2
    #define GETMEMMAPINFO  3
    #define GETEXECUTETIME 4
    #define GETINSCOUNT    5
    void(*getprocessinfo)(uint32_t infotype);
    char* targetname;
    int  pause;
    uint32_t* apimap;
    bool  islogmode;
    FILE* logfp;
    FILE* errlog;
}pprocess;
```

wxemu的设计与实现-对象管理

- FAKEDLINFO结构中包含,要生成win32 dll的一些基本信息.
- head 是PE头数据块.
- apis.ini是具体的dll所包含的api信息(api名,参数,返回值信息)
- 生成的dll,包含系统dll的基本属性,最重要的是生成了API stub代码




```
struct FAKEDLLINFO
{
    uint32_t dllindex;
    uint32_t specail;
    uint32_t baseaddr;
    uint32_t baseofcode;
    uint32_t hostaddr;
    uint32_t exportva;
    uint32_t exportsize;
    uint32_t importva;
    uint32_t importsize;
    uint32_t imagesize;
    uint8_t* head;
    uint8_t* exportdir;
    uint32_t exportnum;
    uint32_t exportstart;
    uint32_t exportnamerva;
    uint32_t exportordal;
    uint32_t gap;
    uint8_t* protobuf;
    uint32_t protosize;
    char* filename;
}fakedll[] =
{
    { 0,0, 0x7c800000,0,0,0,0,0,0,0,kernel32head,kernel32exportdir,954,0,0x4b9e, 0x3820,0xa000,0,0,"kernel32.dll" },
    { 0,0, 0x7c920000,0,0,0,0,0,0,0,ntdll,ntdllexport,1317,0,0x678f, 0x5d44,0x10000,0,0,"ntdll.dll" },
    { 0,0, 0x77da0000,0,0,0,0,0,0,0,advapi32head,advapi32exportdir,676,0,0x3141, 0x1fec,0xa000,0,0,"advapi32.dll" },
    { 0,1, 0x71a20000,0,0,0,0,0,0,0,ws32head,ws32exportdir,117,0,0x1EC5, 0x1DD0,0xa000,0,0,"ws2_32.dll" },
    { 0,0, 0x77d10000,0,0,0,0,0,0,0,user32head,user32exportdir,734,0,0x55d8, 0x501c,0xa000,0,0,"user32.dll" },
```

wxemu的设计与实现-对象管理

线程对象:

1.包含线程执行的一些基本信息与统计信息.

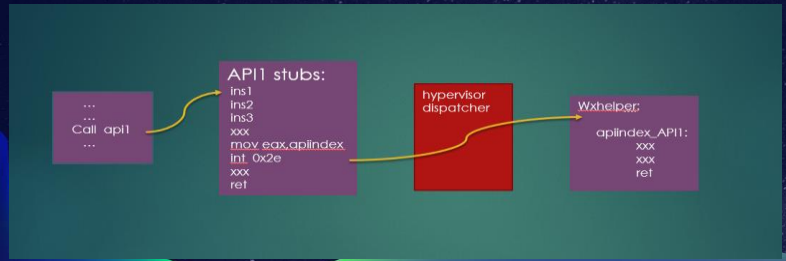
2.最主要的是包含api调用的一些信息.为API 的调用和trace起到很大的作用.

```
struct  _thread
{
    struct  wxobject;
    uint32_t thrdid;
    uint32_t ep;
    enum thrdtype thrdtype;
    struct  apicall   *api_call;
    uint32_t apicallcnt;
    uint64_t last_apicallpc;
    uint64_t lastapicall_stack;
};

struct apicall {
    char *apiname;
    char* argtype;
    int  namelen;
    uint32_t lastmain_pc;
    uint32_t retvalue;
    char* modulename;
    uint32_t last_sysindex;
    void(*apiproc)(struct _process* process, unsigned int eax, uint32_t address, uint32_t size);
};
```

wxemu的设计与实现-api调用执行的过程

- 在pe加载和初始化的过程中,wxemu会解析IAT,然后根据target文件的导入函数名,来装填函数地址.这里写入的是generated dll 的API Stubs的地址.
- UC_HOOK_INTR 可以用来监控int中断或者syscall指令的执行,所以我们可以利用这个中断来跳转到真正的api地址来执行.
- UC_HOOK_CODE 用来监控,trace 每条指令的执行.可以用来收集指令执行序列,反汇编工作.
- wxemu利用int 0x2e 来执行中断,调用真是api,当然这里可以用sysenter来实现.




```
typedef enum uc_hook_type {
    // Hook all interrupt/syscall events
    UC_HOOK_INTR = 1 << 0,
    // Hook a particular instruction - only a very small subset of instructions supported here
    UC_HOOK_INSN = 1 << 1,
    // Hook a range of code
    UC_HOOK_CODE = 1 << 2,
    // Hook basic blocks
    UC_HOOK_BLOCK = 1 << 3,
    // Hook for memory read on unmapped memory
    UC_HOOK_MEM_READ_UNMAPPED = 1 << 4,
    // Hook for invalid memory write events
    UC_HOOK_MEM_WRITE_UNMAPPED = 1 << 5,
    // Hook for invalid memory fetch for execution events
    UC_HOOK_MEM_FETCH_UNMAPPED = 1 << 6,
    // Hook for memory read on read-protected memory
    UC_HOOK_MEM_READ_PROT = 1 << 7,
    // Hook for memory write on write-protected memory
    UC_HOOK_MEM_WRITE_PROT = 1 << 8,
    // Hook for memory fetch on non-executable memory
    UC_HOOK_MEM_FETCH_PROT = 1 << 9,
    // Hook memory read events.
    UC_HOOK_MEM_READ = 1 << 10,
    // Hook memory write events.
    UC_HOOK_MEM_WRITE = 1 << 11,
    // Hook memory fetch for execution events
    UC_HOOK_MEM_FETCH = 1 << 12,
    // Hook memory read events, but only successful access.
    // The callback will be triggered after successful read.
    UC_HOOK_MEM_READ_AFTER = 1 << 13,
} uc_hook_type;
```

wxemu的设计与实现-api调用执行的过程

GetProcessIoCounters的API stub

- eax中保存的是函数的编号,然后host利用这个编号来查找在真正的函数地址实现地址(类似SSDT).
- 找到之后就会跳转到过去执行,最后返回函数调用点.

```
EEEB
EEEB                    ; BOOL __stdcall GetProcessIoCounters(HANDLE hProcess, PIO_COUNTERS lpIoCounters)
EEEB                            public GetProcessIoCounters
EEEB                    GetProcessIoCounters proc near          ; DATA XREF: .text:off_7C802654↑o
EEEB
EEEB                    hProcess        = dword ptr  4
EEEB                    lpIoCounters    = dword ptr  8
EEEB
EEEB 8B FF                              mov     edi, edi
EEED 90                                 nop
EEEE 90                                 nop
EEEF 90                                 nop
EEF0 90                                 nop
EEF1 90                                 nop
EEF2 90                                 nop
EEF3 90                                 nop
EEF4 B8 9D 01 00 00                     mov     eax, 19Dh
EEF9 90                                 nop
EEFA 90                                 nop
EEFB 90                                 nop
EEFC 90                                 nop
EEFD 90                                 nop
EEFE CD 2E                              int     2Eh             ; DOS 2+ internal - EXECUTE COMMAND
EEFE                                                            ; DS:SI -> counted CR-terminated command string
EF00 90                                 nop
EF01 90                                 nop
EF02 90                                 nop
EF03 90                                 nop
EF04 90                                 nop
EF05 C2 04 00                           retn    4
EF05                    GetProcessIoCounters endp
EF05
EF05                    ; ---------------------------------------------------------------------------
```

wxemu的设计与实现-内存管理

Unicorn-engine memory api:

```
uc_mem_write      // write data to virtual address.
uc_mem_read       //read data from virtual address.
uc_mem_map        //Map memory in for emulation.
uc_mem_map_ptr    //Map existing host memory in for emulation
```

1.动态内存申请,释放.
2.内存地址转换(getraddr,getvaddr).
3.满足动态申请，释放内存，堆内存管理等.

wxemu的设计与实现-内存管理

VirtualAlloc的实现

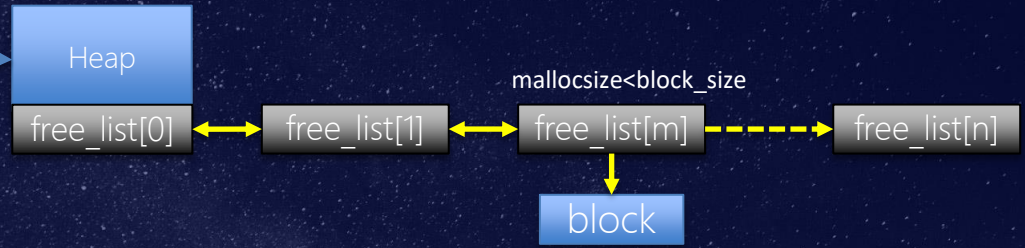

```
300
301  void* WXVritualAlloc(uint32_t address, uint32_t size, uint32_t prot,uint32_t alloc_type)
302  {
303      uint32_t hostaddr;
304      int err;
305      uint32_t value;
306      if (address)
307      {
308          if (!wx_mmap(address, size, prot, MAP_FIXED, &hostaddr))
309          {
310              return address;
311          }
312          else return NULL;
313      }
314      else
315          address = wx_mmap(address, size, prot, WX_MAP_HEAP, &hostaddr);
316      return address;
317  }
```

```
//this means the memory will mapped anywhere .
if (!(internalflag&MAP_FIXED))
{
    uint32_t allocpage;
    if (internalflag&WX_MAP_HEAP)
        allocpage = find_free_pages(GET_PAGE(ALIGN_TO_PAGE(size)), ADDRESS_HEAP_LOW, ADDRESS_HEAP_HIGH);
    else
        allocpage = find_free_pages(GET_PAGE(ALIGN_TO_PAGE(size)), ADDRESS_ALLOCATION_LOW, ADDRESS_ALLOCATION_HIGH);
    if (!allocpage)
    {
        printf("cannot' find free page\n");
        return -ENOMEM;
    }
    addr = GET_PAGE_ADDRESS(allocpage);
}
```

wxemu的设计与实现-内存管理

堆内存的管理
HeapAlloc
HeapFree

```
#define _HEAP_BASE ADDRESS_STACK_TOP+0x100000
    a = wx_mmap(ADDRESS_STACK_TOP, 0x100000, WX_PROT_ALL, MAP_FIXED, &hostaddr);
    a = wx_mmap(_HEAP_BASE, 0x200000, WX_PROT_ALL, MAP_FIXED, &hostaddr);
```



wxemu的设计与实现-VFS的实现

VFS需要实现的功能:
1.将目标文件加载到guest内存.
2.将文件对象写入到虚拟文件系统.
3.必要时，能dump文件到真实环境中.
4.不能影响真实环境.

●wxemu的文件系统都是在内存模拟实现的，所以文件的写入,创建,不会真的影响磁盘。这样主要是为了速度，还有不影响真实环境，执行完病毒， 不会对host产生任何影响.
● wxemu提供接口，如果想获取病毒所创建的文件，是可以获取的.
● wxemu文件系统预先模拟了一些系统的文件，如果需要，很容易增加进去新的文件对象.

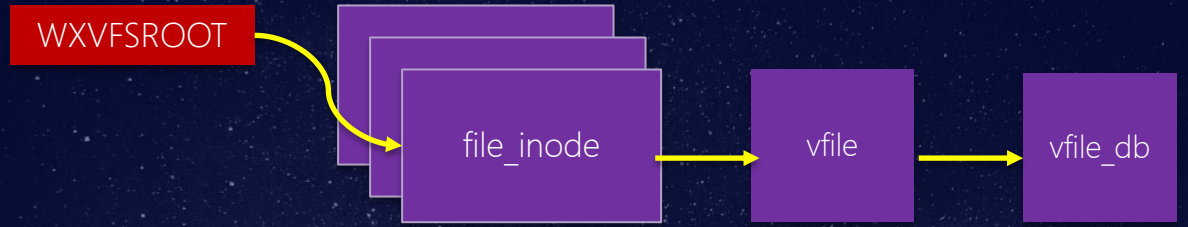
wxemu的设计与实现-VFS的实现

●通过三个不同链表,same_level,sub_level,parent_level来组织整个文件系统的文件结构.

```
struct file_inode
 {
        struct wxobject;
        struct list_head same_level;
        struct list_head sub_level;
        struct list_head *parent_level;
        struct file_inode next_inode;
        struct inode_operations *vnodeops;
};
```

```
struct inode_operations
{
        struct vfile * (*open)(struct file_inode *);
        const char * (*get_pathname)(struct file_inode*);
        ...
        ...
};
```

wxemu的设计与实现-VFS的实现



VFS结构

```
struct  vfile {
    struct wxobject;
    struct  vfile_db *vdb;
};

struct vfile_db
{
    uint8_t* vfile_data;
    uint32_t cur_offset;
    enum     data_type;
    enum     vfile_status;
    struct vfile_ops* vfile_op;
}
```

wxemu的设计与实现-VFS的实现

```
{"C:",            0,0,4,0,0,0,&(builtinode[0].next),NULL},
{"WINDOWS",       0,0,4,0,0,0,&builtinode[6],&builtinode[2],&builtinode[1].next,&builtinode[0]},
{"SYSTEM32",      0,0,4,0,0,0,&builtinode[3],&builtinode[2].next,&builtinode[1]},
{"kernel32.dll",  0,0,2,0,0,0,&builtinode[4],0,&builtinode[3].next,&builtinode[2]},
{"ntdll.dll",     0,0,2,0,0,0,&builtinode[5],0,&builtinode[4].next,&builtinode[2]},
{"wxhelper.dll",  0,0,2,0,0,0,&builtinode[3],0,&builtinode[5].next,&builtinode[2]},
{"Users",         0,0,4,0,0,0,&builtinode[6].next,&builtinode[0] },
{"Maldiohead",    0,0,4,0,0,0,&builtinode[6].next,&builtinode[6]},
{"PhysicalDrive0",0,0,2,0,0,0,&(builtinode[0].next),NULL},
{"Drivers",       0,0,2,0,0,0,&builtinode[3],0,&builtinode[9].next,&builtinode[2]},
{"maldiohead.exe",0,0,2,0,0,0,&builtinode[3],0,&builtinode[10].next,&builtinode[7]},
{"temp",          0,0,4,0,0,0,&builtinode[3],&builtinode[2].next,&builtinode[1]},
```

虚拟出的文件节点

wxemu的设计与实现-VRegistry的实现

1.注册表的模拟主要是实现注册表的创建键,键值读写.
2.windows中最重要的三个hive是HKEY_LOCAL_MAHINCE,HKEY_CURRENT_USER.HKEY_CLASS_ROOT.所以这里主要是对这三个的一个实现.

```
struct regvfs
{
        struct wxobject regvfsobj;
        char* regbuf;
        uint32_t registry_size;
        struct list_head keynode;
        struct list_head virtualkey;
}_regvfs;
```

```
struct key
{
    struct object       obj;
    struct list_head    sibling;
    char                *buf;
     ....
    unsigned short      bufsize;
    int                 last_subkey;
    int                 nb_subkeys;
    struct key          **subkeys;
    struct key_value    *values;
} key;
```

wxemu的设计与实现- Win32 api的模拟

- win32 api 模拟一部分是在wxhelper中实现的,另外一部分是在wxemu内核中实现的.
- Api的模拟分为两种,一种简单的模拟，另一种需要"内核"的帮助（系统级模拟）.
- 简单模拟一般指那种，直接可以返回一个值，或者所有的操作都是内存读写，不需要调用文件读写，注册表读写，句柄访问的函数等.
- 系统级模拟往往需要模拟器内核参与，比如创建文件，就需要VFS的参与，载入动态库也是如此.内存申请，释放需要unicron-engine的参与.
- api代码可参考wine/reactos ☺

简单模拟	系统级模拟
GetCurrentProcessId	CreateFile*
GetModuleHandleA	LoadLibrary*
GetCurrentProcess	Reg*Key
GetProcessHeap	VirtualAlloc*

wxemu的功能

- API trace.
- 内存字符串dump.
- 虚拟调试器.
- 脱壳器.

wxemu 的功能-api trace

```
0x00402f2f: kernel32.dll!InterlockedExchange(0x0040924c, 0x77d10000)=>0x0
0x00402fc1: kernel32.dll!GetProcAddress(0x77d10000, 0x004057fa :"wsprintfA")=>0x77d245b8
0x00402873: user32.dll!wsprintfA(0x00108bac, 0x004084e0 :"%c%c%c%c%c.exe", 0x00000076 :"?")=>0xa
0x0040288a: kernel32.dll!lstrcatA(0x00108d0c :"C:\Windows\System32\", 0x004084dc :"\")=>0x108d0c
0x0040289a: kernel32.dll!lstrcatA(0x00108d0c :"C:\Windows\System32\vmjjeo.exe", 0x00108bac :"vmjjeo.exe")=>0x108d0c
0x004028af: kernel32.dll!CopyFileA(0x00108c04 :"C:\Users\Maldiohead\Maldiohead.exe", 0x00108d0c :"C:\Windows\System32\vmjjeo.exe", 0x00000000)=>0x1
0x004028d2: kernel32.dll!lstrcpyA(0x00108c04 :"C:\Windows\System32\vmjjeo.exe", 0x00108d2b :"  ")=>0x108c04
0x00402fc1: kernel32.dll!GetProcAddress(0x77da0000, 0x0040670e :"OpenSCManagerA")=>0x77db0857
0x004028fe: advapi32.dll!OpenSCManagerA(NULL, NULL, 0x000f003f)=>0x1
0x00402fc1: kernel32.dll!GetProcAddress(0x77da0000, 0x004057e8 :"CreateServiceA")=>0x77dac4e9
0x0040293c: advapi32.dll!CreateServiceA(0x00000001, 0x00408014 :"Nationalreo", 0x00408034 :"Nationalntm Instruments Domain Service", 0x000f01ff, 0x00000110, 0x00000002, 0x00000000, 0x00108c04 :"C:\Windows\System32\vmjjeo.exe", NULL, 0x00000000)=>0x1
0x00402fc1: kernel32.dll!GetProcAddress(0x77da0000, 0x004057d8 :"StartServiceA")=>0x77db270b
0x00402984: advapi32.dll!StartServiceA(0x00000001, 0x00000000, 0x00000000)=>0x1
0x0040299a: kernel32.dll!lstrcpyA(0x00108e10 :"SYSTEM\CurrentControlSet\Services\", 0x004084cf :"")=>0x108e10
0x004029a8: kernel32.dll!lstrcatA(0x00108e10 :"SYSTEM\CurrentControlSet\Services\Nationalreo", 0x00408014 :"Nationalreo")=>0x108e10
0x00402fc1: kernel32.dll!GetProcAddress(0x77da0000, 0x004057ca :"RegOpenKeyA")=>0x77db13ca
0x004029c1: advapi32.dll!RegOpenKeyA(0x80000002, 0x00108e10 :"SYSTEM\CurrentControlSet\Services\Nationalreo", 0x00108d08)=>0x1
0x004029cb: kernel32.dll!lstrlenA(0x004080b4 :"Providesfht a domain server for NI security.")=>0x2c
0x00402fc1: kernel32.dll!GetProcAddress(0x77da0000, 0x004057b8 :"RegSetValueExA")=>0x77db18c2
0x004029e3: advapi32.dll!RegSetValueExA(0x00000000, 0x004084d0 :"Description", 0x00000000, 0x00000001, 0x004080b4, 0x0000002c)=>0x0
0x00402fc1: kernel32.dll!GetProcAddress(0x77da0000, 0x004057a2 :"CloseServiceHandle")=>0x77dabd0b
0x00402a17: advapi32.dll!CloseServiceHandle(0x00000001)=>0x1
0x00402a22: advapi32.dll!CloseServiceHandle(0x00000001)=>0x1
0x00401f6d: kernel32.dll!GetModuleFileNameA(NULL, 0x00108c30 :"C:\Users\Maldiohead\Maldiohead.exe", 0x00000022)=>0x22
0x00401f8a: kernel32.dll!GetShortPathNameA(0x00108c30 :"C:\Users\Maldiohead\Maldiohead.exe", 0x00108c30 :"C:\Users\Maldiohead\Maldiohead.exe", 0x00000104)=>0x99
0x00401faa: kernel32.dll!GetEnvironmentVariableA(0x00408498 :"COMSPEC", 0x00108e38 :"C:\Windows\system32\cmd.exe", 0x00000104)=>0x1b
0x00401fc5: kernel32.dll!lstrcpyA(0x00108d34 :" /c del ", 0x00408495 :"")=>0x108d34
0x00401fda: kernel32.dll!lstrcatA(0x00108d34 :" /c del C:\Users\Maldiohead\Maldiohead.exe", 0x00108c30 :"C:\Users\Maldiohead\Maldiohead.exe")=>0x108d34
0x00401fe9: kernel32.dll!lstrcatA(0x00108d34 :" /c del C:\Users\Maldiohead\Maldiohead.exe > nul", 0x00408484 :" > nul")=>0x108d34
0x00401ffb: kernel32.dll!lstrcatA(0x00108e38 :"C:\Windows\system32\cmd.exe /c del C:\Users\Maldiohead\Maldiohead.exe > nul", 0x00108d34 :" /c del C:\Users\Maldiohead\Maldiohead.exe > nul")=>0x108e38
0x0040203c: kernel32.dll!GetCurrentProcess()=>0xffffffff
0x00402045: kernel32.dll!SetPriorityClass(0xffffffff, 0x00000100)=>0x1
0x0040204f: kernel32.dll!GetCurrentThread()=>0xfffffffe
0x00402058: kernel32.dll!SetThreadPriority(0xfffffffe, 0x0000000f)=>0x1
0x0040207e: kernel32.dll!CreateProcessA(NULL, 0x00108e38 :"C:\Windows\system32\cmd.exe /c del C:\Users\Maldiohead\Maldiohead.exe > nul", 0x00000000, 0x00000000, 0x00000000, 0x0000000c, 0x00000000, NULL, 0x00108bec, 0x00108bdc)=>0x1
0x0040208b: kernel32.dll!SetPriorityClass(0x00000000, 0x00000040)=>0x1
```

wxemu 的功能-我是如何发现Xshell使用的是DGA域名的

- 8月份,Xshell后门事件中使用的DGA域名，利用的是GetSystemTime函数来计算的.
- wxemu跑出来的域名和当时最新公开的域名不一样(8月份的DGA域名是nylalobghyhirgh.com,模拟这个api的时间设置的是一月份，见下图).
- 出现域名字符串的上面调用了获取时间的函数，所以怀疑是DGA域名.

然后尝试修改当前时间为8月份，结果和公开的域名一样，最终确定是DGA算法生成的域名的.

```
LPSYSTEMTIME WINAPI __GetSystemTime(LPSYSTEMTIME lpSystemTime)
{

    lpSystemTime->wYear = 2017;
    lpSystemTime->wMonth = 1;
    lpSystemTime->wDay = 1;;
    lpSystemTime->wHour = 22;
    lpSystemTime->wMinute = 22;
    lpSystemTime->wSecond = 22;
    lpSystemTime->wMilliseconds = 2222;
    lpSystemTime->wDayOfWeek = 2;

    return lpSystemTime;
}
```

Demo

目前存在缺点与开发计划

●目前还不能真正支持多线程(目前是将线程当作一个回调函数来执行,执行完成之后,返回调用点继续执行)
●脚本引擎仍在开发当中.
●目前不支持VB/.Net程序的运行.
●目前不支持x86_64bit.

计划:
● 会在半年内开源请关注： https://github.com/maldiohead
● 由于unicorn-engine的没有提供支持多线程运行的能 力,所以后期计划在unicorn-engine基础之上,增加这一特性.
●尽快支持VB .Net程序.

Question?