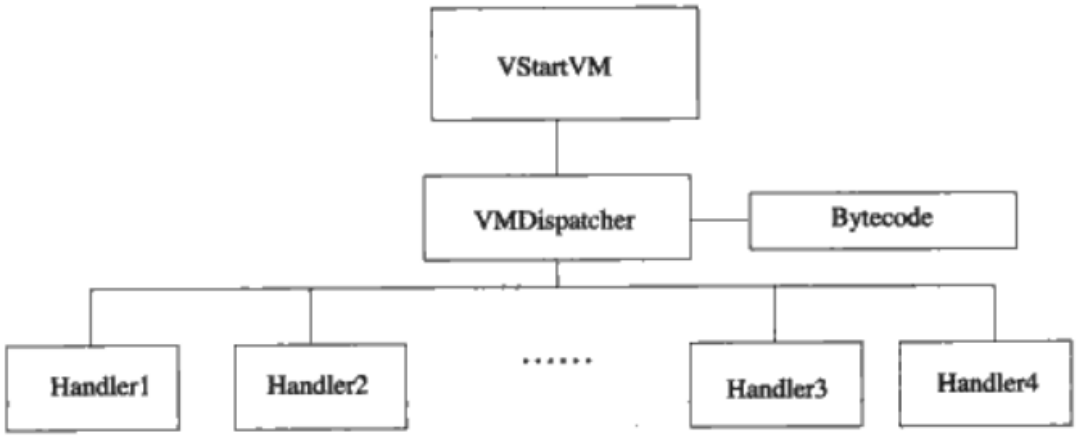
VMP原理

虚拟机保护技术就是将基于x86汇编系统的可执行代码转换为字节码指令系统的代码，以达到保护原有指令不被轻易逆向和修改的目的

https://www.cnblogs.com/LittleHann/p/3344261.html

大致结构如图：



- `VStartVM`部分初始化虚拟机
- `VMDispatcher`负责调度这些Handler

- VMP的句柄（`Handler`）：某一类的机器码的解释执行代码
- `Bytecode`就是虚拟伪指令
- 在程序中，`VMDispatcher`往往是一个类while结构，不断的循环读取伪指令，然后执行

虚拟机的分类

- 栈式虚拟机

  所有操作数通过出栈入栈进行操作，中间码体积小，效率低

  代表：`java`，`lua5.0`以前
- 寄存器式虚拟机

  所有操作数通过内部寄存器操作，中间码的体积大，效率高

  有一块内存， 相当于自己的寄存器，`reg0 - reg16`

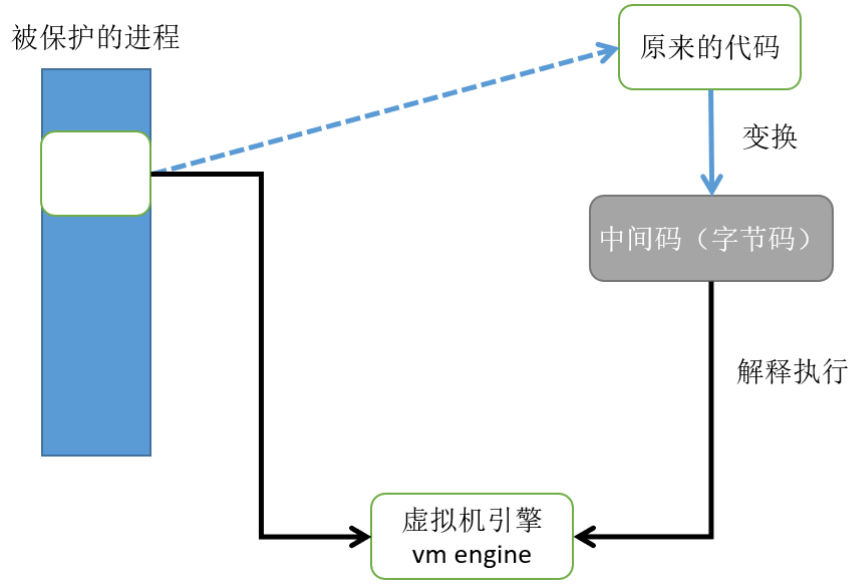
  代表：`lua5.0`以后，`davik(android 5.0)`
- `jit - just in time`，即时编译

  将中间码翻译成机器码执行

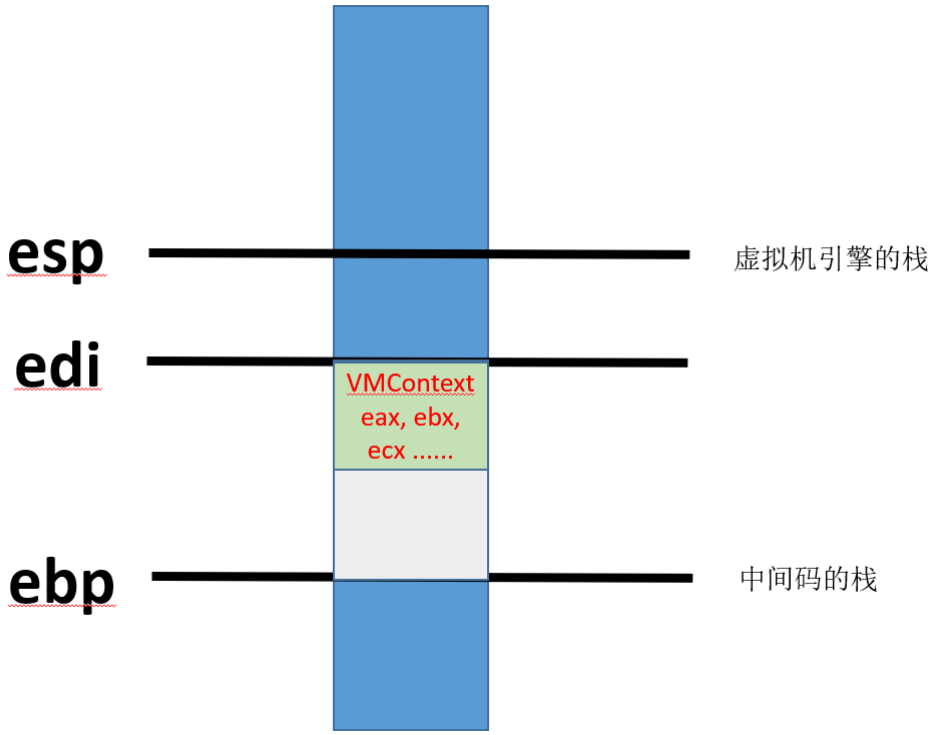
  代表：`.net`， `android art`， `java`现在版本

VMP

VMP虚拟引擎使用真实的寄存器，VMP字节码使用内存空间当寄存器，全部执行完成后将内存的数据赋值给真实的寄存器



栈结构



即

```
edi = VMContext
esi = 当前字节码地址
ebp = 真实堆栈

在整个虚拟机代码执行过程中，必须要遵守一个事实。
1. 不能将edi,esi,ebp寄存器另做他用
2. edi指向的VMContext存放在栈中而没有存放在其他固定地址或者申请的堆空间中，是因为考虑到多线程程序的兼容
```

VMContext

注意：对于不同的虚拟机不同的版本中Context成员的顺序可能不同

```
VMContext:
    +0  fd
    +4  dx
    +10 esp
    +18 esi
    +1c ebx
    +20 eax
    +24 edi
    +28 ecx
    +2c ebp
```

其他对抗手段

1. 反调试

   思路是利用进程在调试状态下和非调试状态下的区别来进行反调试

   手段：https://bbs.pediy.com/thread-257600.htm

2. 混淆

   - 代码膨胀

     利用的IDA反编译，代码全部拷贝到VS中，利用VS的F7功能可去膨胀

   - 流程混淆

     可利用x64dbg的绘制图功能，绘制执行流程可去混淆

   - IAT混淆

     IAT表中跳往混淆代码，通过混淆代码最后跳往API处