Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

污点分析及其关键技术

网络安全组 王沛冉

2019年05月26日

内容提要

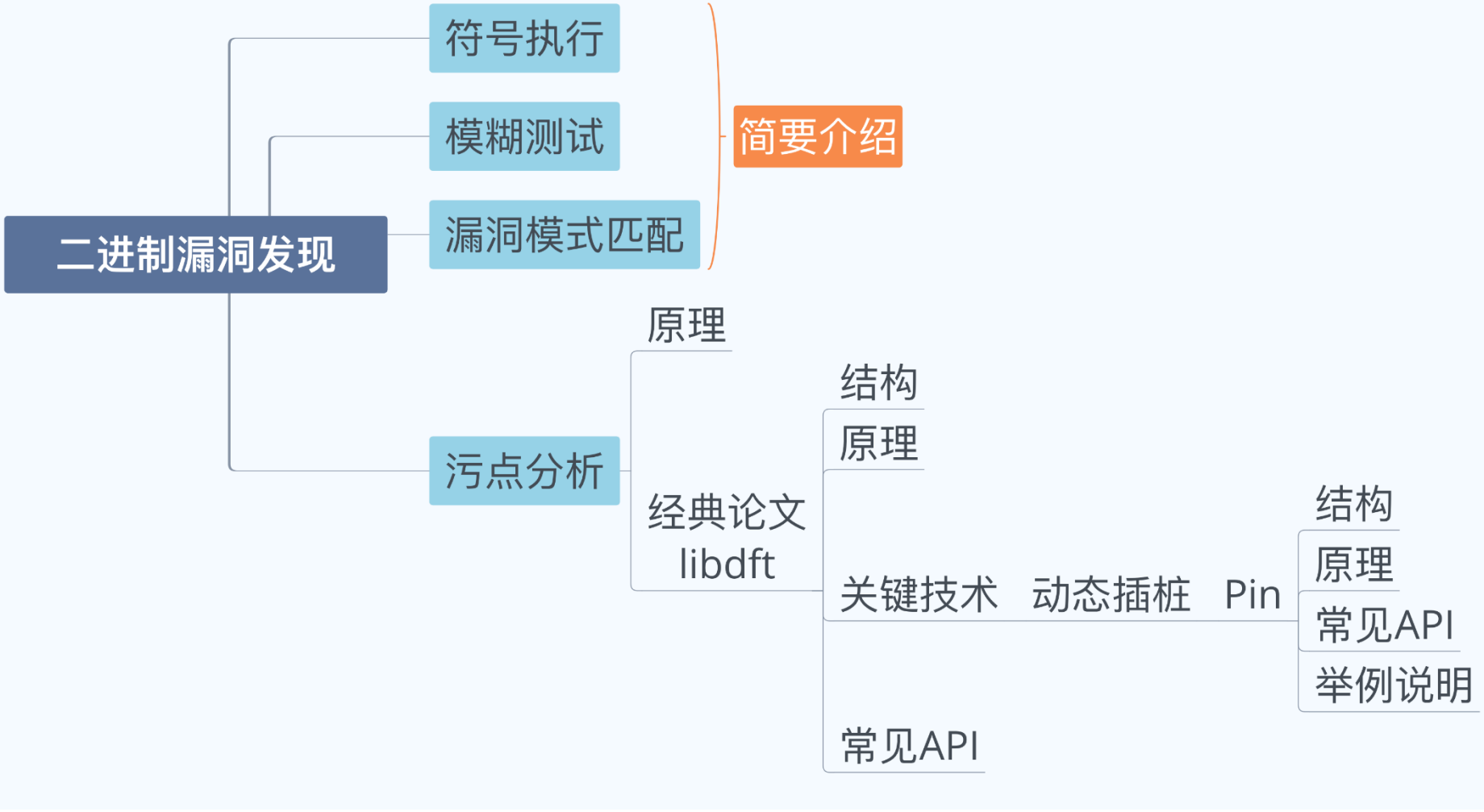
- 二进制漏洞挖掘概要
- 污点分析技术简介及原理
- 污点分析框架Libdft简介及原理
- 二进制动态插桩框架Pin原理

预期收获

- 网络安全组：
  - 1.了解二进制漏洞发现主流研究方向
  - 2.深入了解污点分析原理
  - 3.libdft工具原理
  - 4.插桩技术与Pin工具原理和实践

- 文本安全组及数据挖掘组：
  - 1.对二进制程序漏洞发现研究概要有一定了解
  - 2.充实自己的技术栈

思维导图


二进制漏洞发现
符号执行
模糊测试
漏洞模式匹配
简要介绍
污点分析
原理
结构
原理
经典论文
libdft
关键技术
动态插桩
Pin
结构
原理
常见API
举例说明
常见API

符号执行

- 符号执行技术的核心思想是使用符号值来表示程序的输入数据，并将程序的运算过程逐指令或逐语句地转换为数学表达式，在控制流图的基础上生成符号执行树，并为每一条路径建立一系列以输入数据为变量的符号表达式。
- 把程序的输入改为一个符号，则程序的输出为该符号的函数。

T	追踪控制流信息，进行漏洞挖掘
I	二进制可执行程序或源码
P	通过符号执行引擎执行目标程序或静态分析源码
O	符号执行树

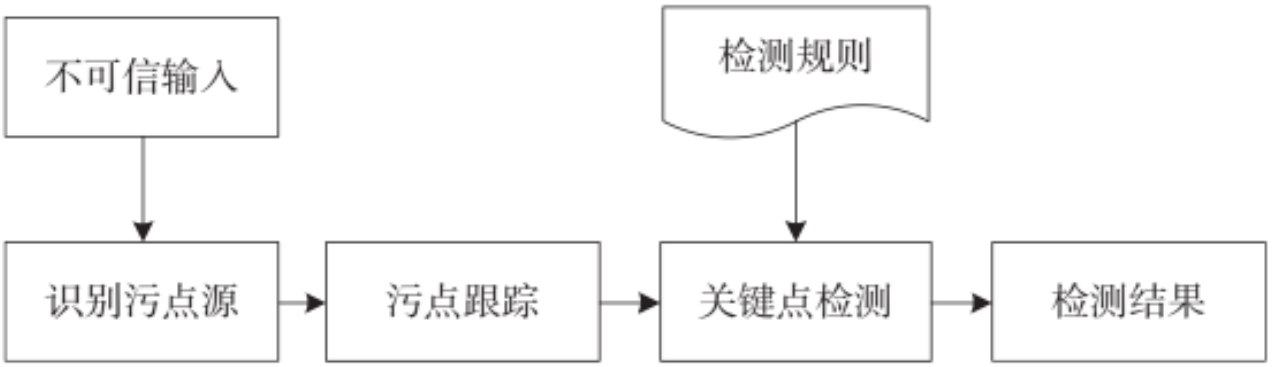
模糊测试

- 模糊测试（fuzz testing）介于完全的手工测试和完全的自动化测试之间。使用工具基于规则（手动设置）生成大量测试样例，并将测试样例输入到被测试系统中，（自动）检测系统是否响应正常。

T	通过半自动生成模糊测试样例对系统进行黑盒测试
I	待测试系统、样例生成规则
P	1.模糊测试工具根据规则生成大量测试样例 2.将测试样例导入系统中进行黑盒测试 3.检测系统响应 4.根据系统响应判断是否存在异常
O	系统响应分析结果

污点分析-定义

- 污点分析就是分析程序中由污点源引入的数据是否能够不经无害处理,而直接传播到污点汇聚点.如果不能,说明系统是数据流安全的;否则,说明系统产生了**隐私数据泄露**或**危险数据操作**等安全问题.



污点分析-定义

- TIPO

T	通过追踪数据流信息，检测是否发生了敏感信息泄露或危险数据操作等行为，进而实现漏洞挖掘
I	1.污点源和汇聚点标识范围 2.污点传播逻辑 3.无害化处理 4.汇聚点检测回调函数
P	通过对数据流追踪，进行污点传播分析
O	污点检测触发结果和回调函数运行结果

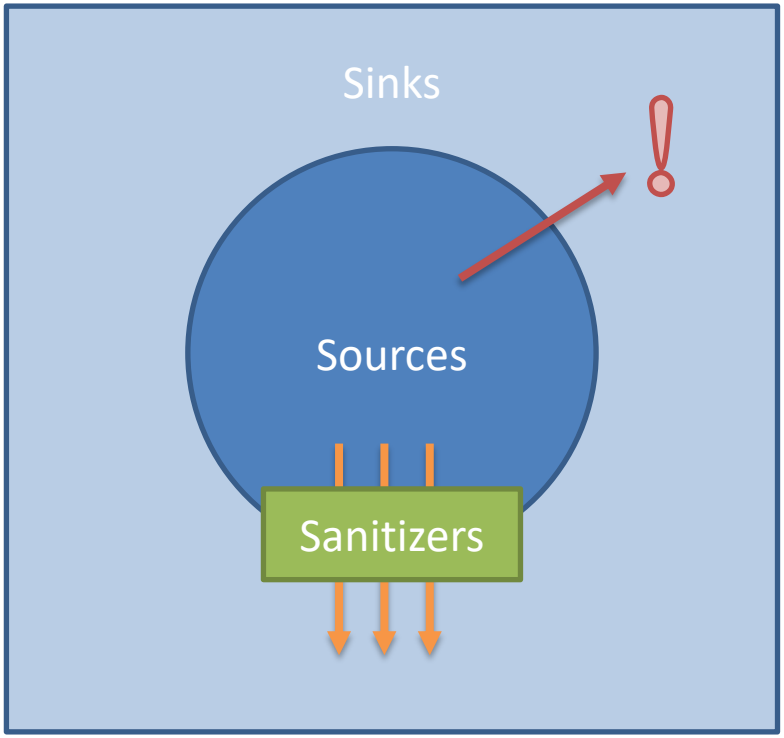
污点分析-定义

- 污点分析关键点可以抽象成一个三元组：

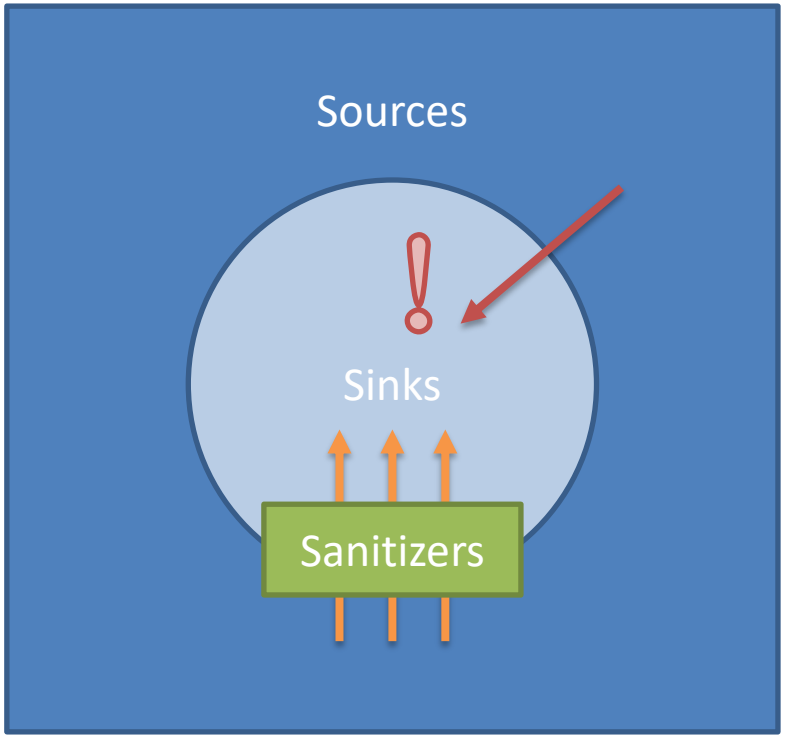
  <sources, sinks, sanitizers>

- sources：即污点源，代表直接引入的不受信任的数据
- sinks：即汇聚点，代表污点检测点，判断是否有被污染的数据到达汇聚点。
- sanitizers：即无害化处理，代表通过数据加密或者移除危害操作等手段使数据传播不再对软件系统的信息安全产生危害。

污点分析-定义



检测隐私数据泄露



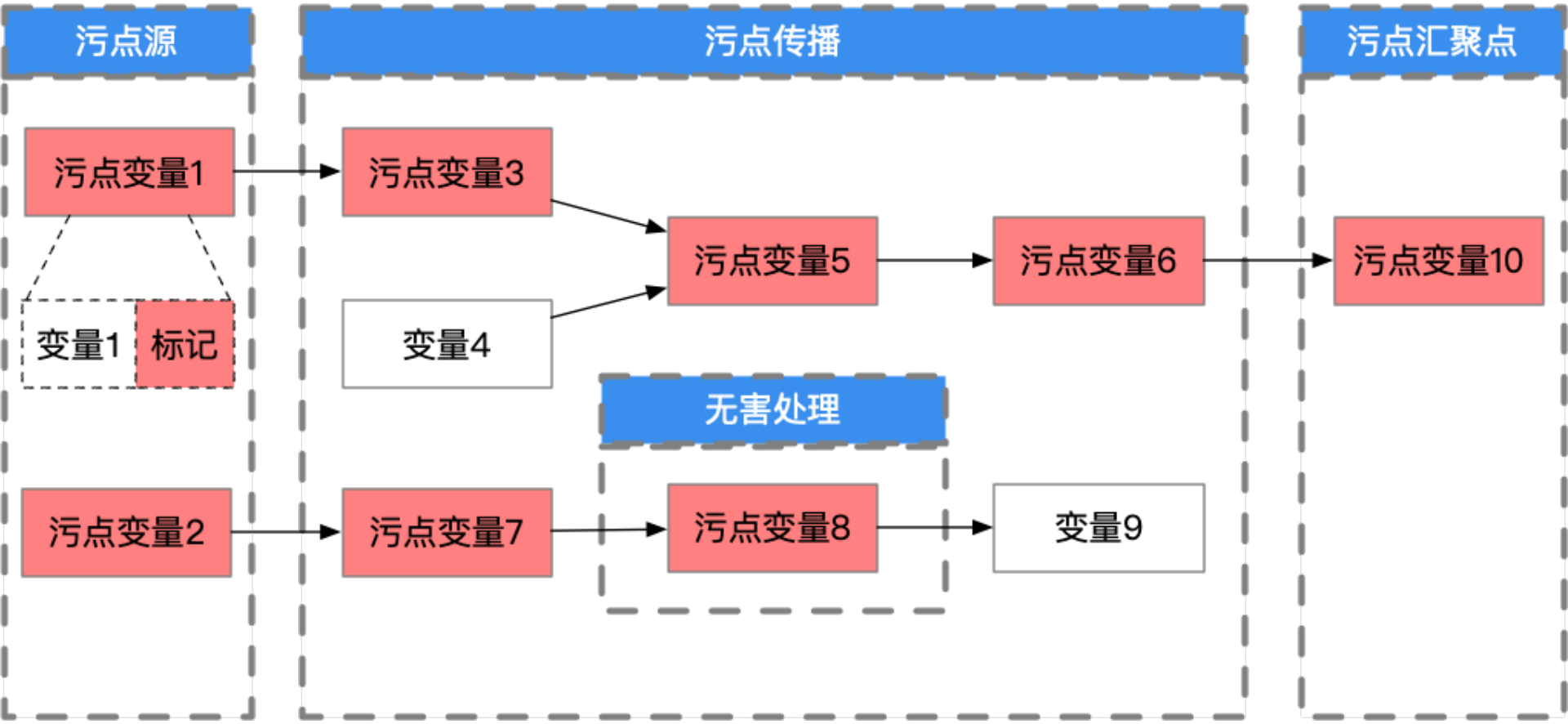
检测危险数据操作

污点分析-原理

- 污点分析的处理过程
- （1）识别污点源：判定哪些输入或数据是污点
- （2）划定汇聚点：划定在哪些区域检测污点
- （3）污点传播分析：将受影响的其他变量\数据\寄存器\内存区块等添加污点标记
- （4）无害化处理：某些变量\数据\寄存器\内存区块等经过处理后不再是污点，去掉其污点标记
- （5）汇聚点检测：检测某些函数输入输出\变量\寄存器\内存区块等处是否存在污点标记
- （6）异常处理：汇聚点检测出污点后，执行指定代码
- 步骤（1）（2）是前提，步骤（3）~（5）无顺序关系

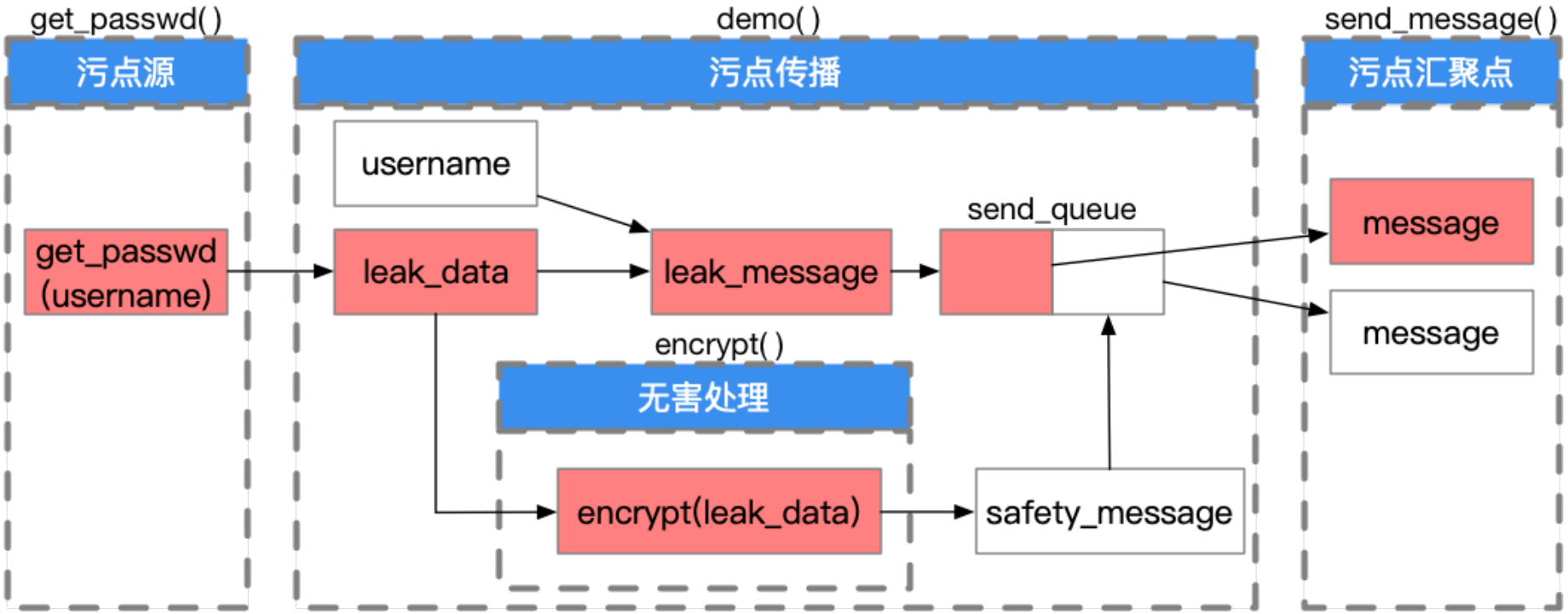
污点分析-原理

- 污点分析过程直观示例



污点分析-原理

```
1  def demo(username):
2      send_queue = []
3      leak_data = get_passwd(username)     # source
4      leak_message =  'Username=' + username + '|passwd=' + str(leak_data)
5      send_queue.append(leak_message)
6      safety_message = encrypt(leak_message)  # sanitizers
7      send_queue.append(safety_message)
8      for message in send_queue:
9          send_message(message)    # sinks
```



污点分析-原理

- **标定污点源（sources）和汇聚点（sinks）：**
- 标定污点源和汇聚点是污点分析的前提
- 污点源是污点分析的“起点”
- 汇聚点是污点分析的“终点”

- 方法
  - 1. 将所有输入标记为“污点”（PHP Aspis）
  - 2.针对具体程序调用的API或者重要的数据类型，设定规则标注部分输入（DroidSafe）
  - 3.使用统计和机器学习技术自动识别和标记污点源

Rasthofer S, Arzt S, Bodden E. A machine-learning approach for classifying and categorizing android sources and sinks. In: Proc. of the Network and Distributed System Security Symp. (NDSS). 2014. [doi: 10.14722/ndss.2014.23039]

污点分析-原理

- **标定污点源（sources）和汇聚点（sinks）：**
- 在不同的应用程序中识别污点源和汇聚点的方法各不相同
- 如何自动标定污点源和汇聚点是目前研究的重点

- 原因：
  - 系统模型、编程语言之间的差异
  - 污点分析关注的安全漏洞类型不同

- 举例：
- 不同安全漏洞：
  - SQL注入检测：Sources：用户输入
    Sinks：执行的SQL语句
  - 栈溢出检测：Sources：入栈数据
    Sinks：栈外空间
- 相同安全漏洞（SQL注入）：
  - 危险数据操作检测：Sources：用户输入
    Sinks：执行的SQL语句
  - 敏感数据泄露检测：Sources：数据库检索结果
    Sinks：发送给客户端的数据

污点分析-原理

- **污点传播分析**

  - **显示流污点传播**：污点数据**直接参与**数据流传播，常见的显示污点传播方式包括：直接赋值传播、通过函数（过程）调用传播以及通过别名（指针）传播

  - **隐式流污点传播**：污点数据**间接控制**数据流传播，常见的隐式污点传播主要通过控制依赖进行传播：污点数据作为程序分支的判断条件

污点分析-原理

- 显示流污点传播

```
1  def demo(username):
2      send_queue = []
3      leak_data = get_passwd(username)     # source
4      leak_message =  'Username=' + username + '|passwd=' + str(leak_data)
5      send_queue.append(leak_message)
6      safety_message = encrypt(leak_message)  # sanitizers
7      send_queue.append(safety_message)
8      for message in send_queue:
9          send_message(message)    # sinks
```

leak_data直接参与运算直接污染leak_message

- 隐式流污点传播

```
1  def demo2(passwd):  # source
2      data = ''
3      if passwd == '123456':
4          data = get_secret()
5      else:
6          data = get_fake()
7      return data      # sink
```

passwd间接污染data

污点分析-原理

- 隐式流污点传播存在的问题

- 欠污染（under-taint）：欠污染问题是由于没有对隐式流污点传播进行适当的处理，导致本应被标记的变量没有被标记。

- 过污染（over-taint）：过污染问题是因为所标记的污点数量过多而导致污点变量大量扩散，一般是由于无害处理不正确和污染粒度选取不佳导致。

污点分析-原理

- 欠污染举例

```
1   def demo3(input):    # source
2       x = False
3       y = False
4       if input == 'ABC':
5           x = True
6       else:
7           y = True
8       if x is False:
9           sink(x)      # sink
10      if y is False:
11          sink(y)      # sink
```

- 部分泄露（partially leaked）：污点信息通过动态未执行部分进行传播并泄露。

污点分析-原理

- 污点传播一般分析方法：
  - 静态分析技术：
    - 特点：1.不运行代码 2.不修改代码
    - 方法：通过分析程序变量间的依赖关系来检测数据能否从污点源传播到污点汇聚点。首先根据程序中的函数调用关系构建调用图（Call Graph），然后在函数内或者函数间根据不同的程序特性进行具体的数据流传播分析。
    - 优点：速度快，能够快速定位污点在程序中可能出现的传播路径
    - 缺点：精度低，因为无法获取程序运行时的一些必要信息，分析结果往往存在偏差，需要人工对分析结果进行复查。

污点分析-原理

- 污点传播一般分析方法：
  - 动态分析技术
    - 特点：1.在程序运行时进行分析 2.只分析运行时可达的支路
    - 方法：通过实时监控程序的污点数据在系统程序中的传播来检测数据能否从污点源传播到污点汇聚点.。首先需要为污点数据扩展一个污点标记(tainted tag)的标签并将其存储在存储单元(内存、寄存器、缓存等)中,然后根据指令类型和指令操作数设计相应的传播逻辑传播污点标记。
    - 优点：分析准确率较静态分析高
    - 缺点：由于引入了代码重写和插桩等机制，会显著增加程序的开销。

Libdft-简介

- Libdft由论文：Kemerlis V P, Portokalidis G, Jee K, et al. libdft: Practical Dynamic Data Flow Tracking for Commodity Systems[J]. Acm Sigplan Notices, 2012, 47(7):121-132.提出，该论文来自于2012 ACM VEE 会议

- Libdft是一款开源通用的动态污点分析框架，，兼容主流的X86架构。可以对复杂的大型商业软件进行动态污点分析。

- Libdft基于Intel动态插桩框架Pin，并作为Pintool的第三方库，可以被用户构建的Pintool所调用。主要的污点传播分析部分功能依靠Pin的API来确定二进制程序的执行情况，同时收集必要的信息。

Libdft-原理

- 框架运行时的程序内存映像

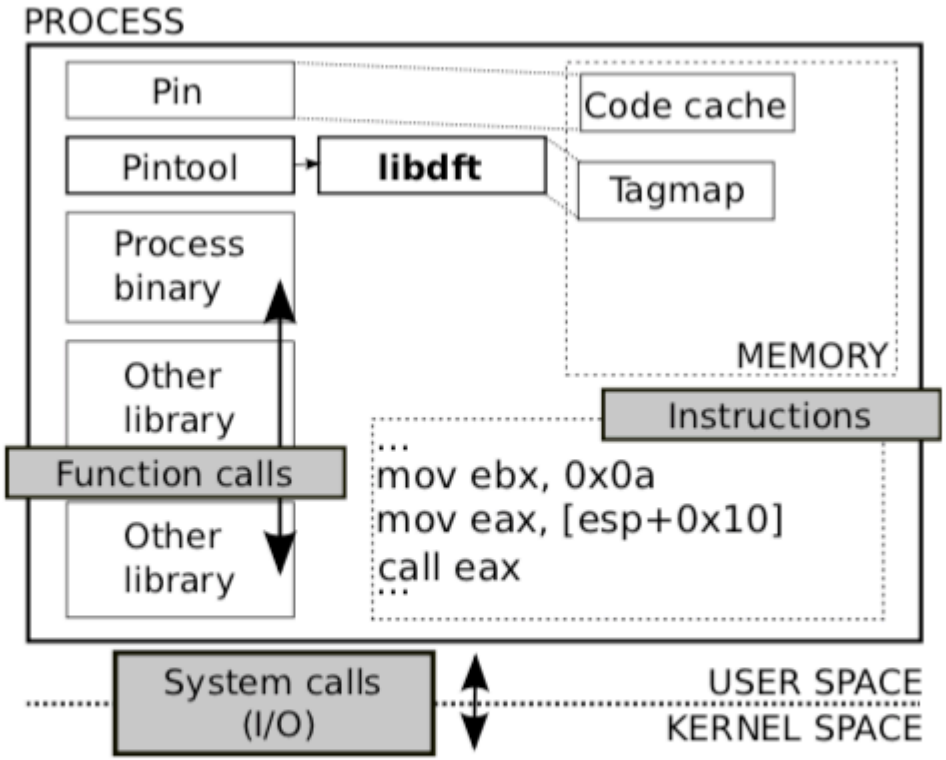
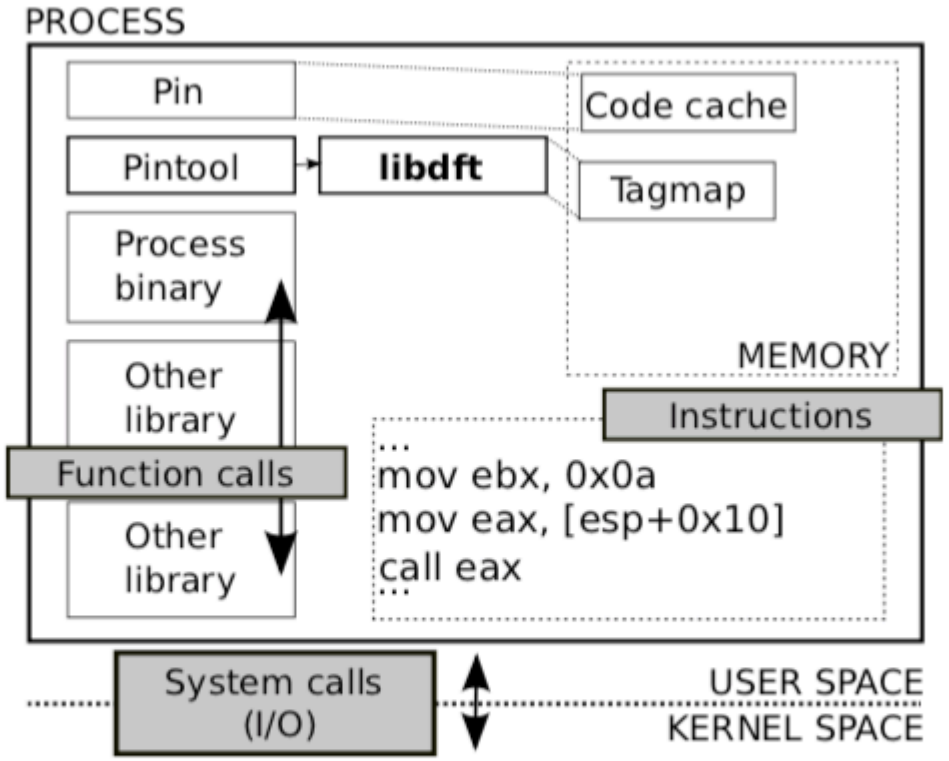


Figure 2. Process image of a binary running under libdft. The highlighted boxes describe possible data sources and sinks that can be used with libdft.

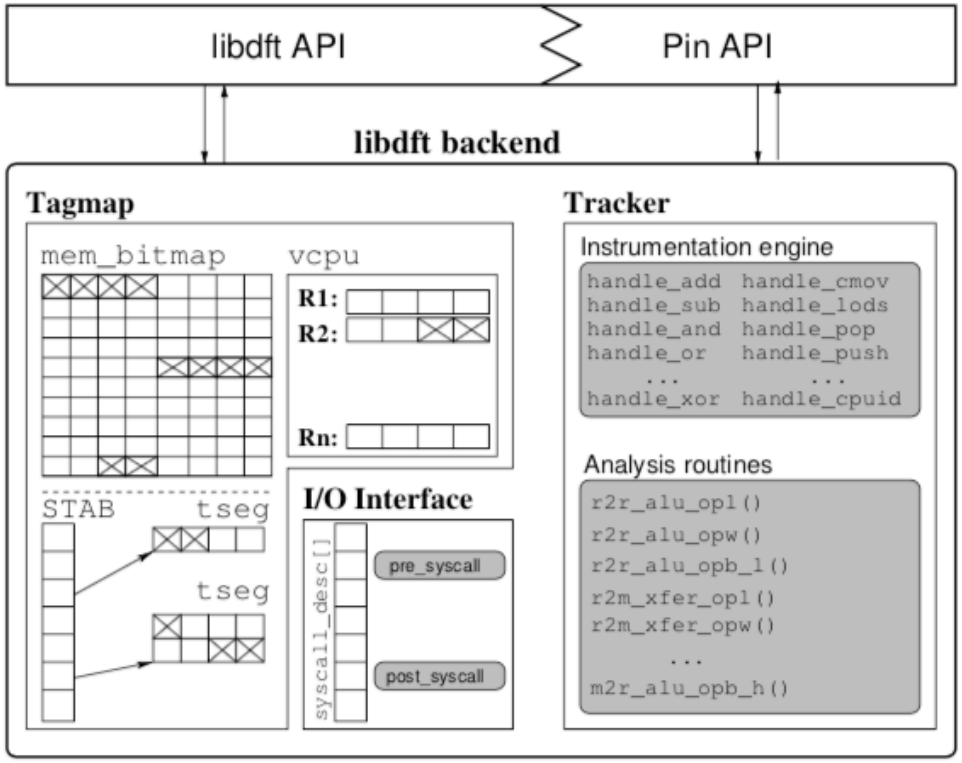
Libdft-原理

- libdft定义了三种可以被看作是污点源（sources）或者汇聚点（sinks）的类型：程序指令、函数调用和系统调用，DFT过程通过在这些位置触发回调函数来实现，这些回调函数也称为污点传播策略。



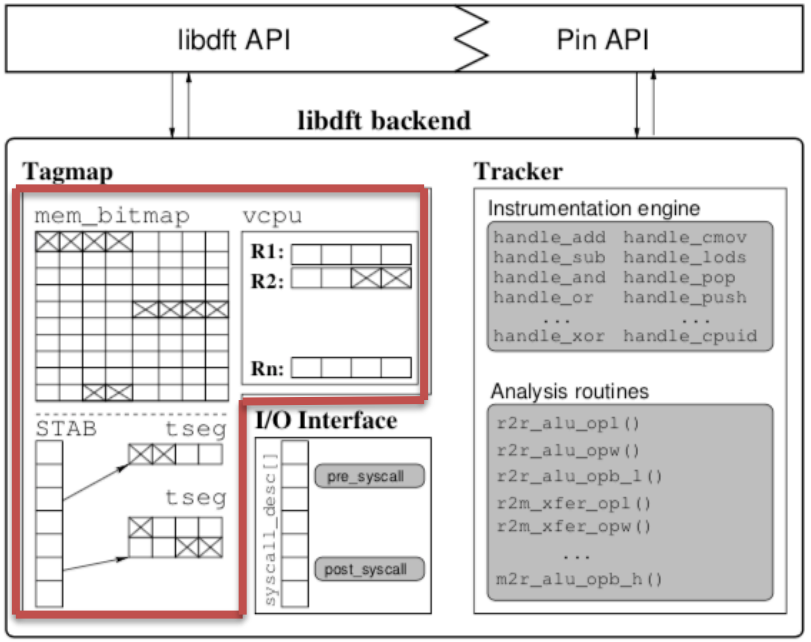
Libdft-原理

- Libdft的三大组件：
- Tagmap：存储数据污点标记信息
- Tracker：逻辑部分，供用户自定义污点传播逻辑
- I/O Interface：处理I/O，用户自定义哪些输入为污点



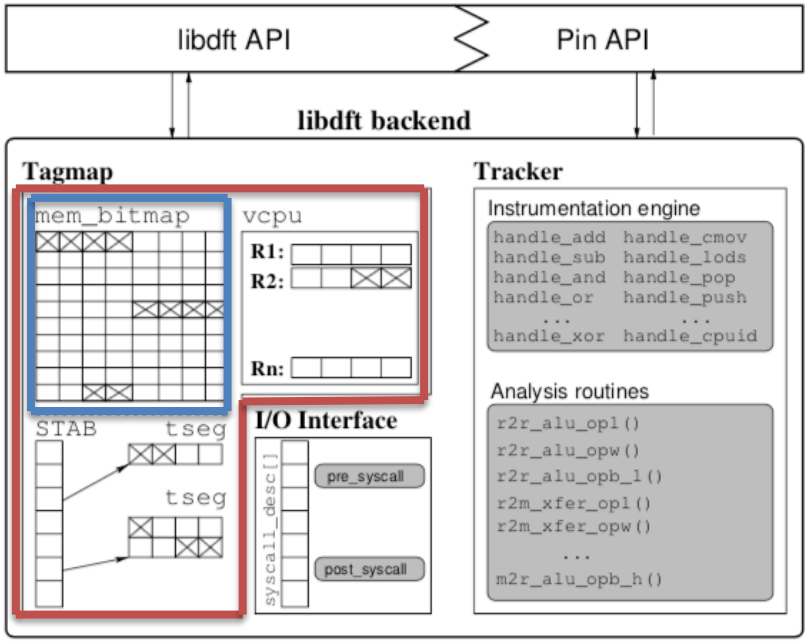
Libdft-原理

- **Tagmap**
- Tagmap是libdft中非常重要的组成部分。它包含一个进程数据的结构的映射，只映射了结构，没有映射内存的具体内容。
- Tagmap保存着该进程数据的污点标记信息



Libdft-原理

- mem_bitmap作为“影子内存”，映射了目标程序的内存结构，并保存其污点标记状态（Tag）。
- 最小粒度为：字节级
- 过小粒度造成过大开销
- 过大粒度造成过污染



Libdft-原理

- libdft支持两种Tag标签格式
- （1）字节标签，即可以将数据标记为不同的类别，最多有8个类别，占8bits。
- （2）位标签：只有被标记和未被标记两种状态。

type1	0	0	0	0	0	0	0	1
type2	0	0	0	0	0	0	1	0
type3	0	0	0	0	0	1	0	0
type4	0	0	0	0	1	0	0	0
type5	0	0	0	1	0	0	0	0
type6	0	0	1	0	0	0	0	0
type7	0	1	0	0	0	0	0	0
type8	1	0	0	0	0	0	0	0

type1	0
type2	1

Libdft-原理

- 当使用位标签时：libdft将内存每1byte固定映射成为Tagmap->mem_bitmap中的1bit位，无论该内存是否被进程占用。即对于32位操作系统，mem_bitmap会固定占用12.5%的内存。查询标记状态时使用内存地址的高29位作为偏移量到mem_bitmap中去寻找包含源真实地址污点标记信息的字节，然后用低3位在该字节中继续索引，最终找到目标信息。

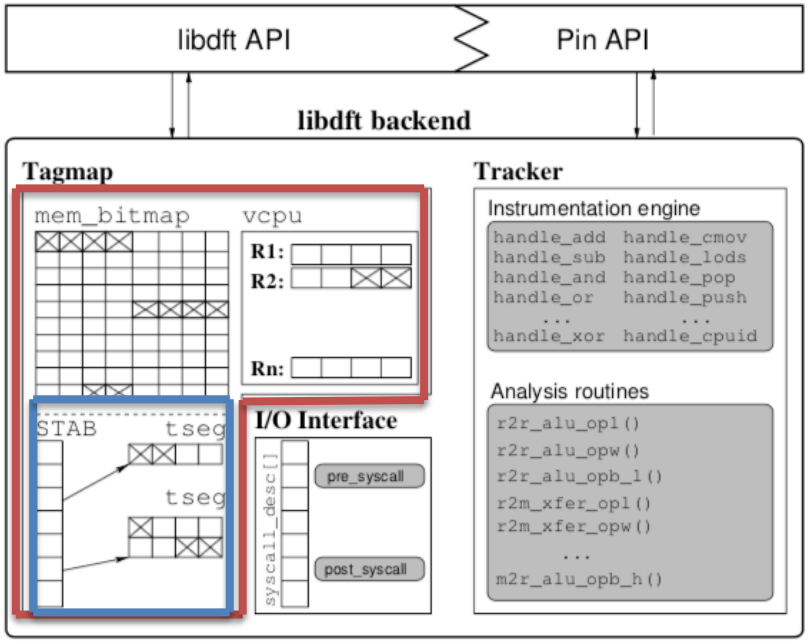
- 目的：降低真实地址和Tagmap映射地址查询时间，减少开销。

Libdft-原理

- 对于采用64位操作系统或使用字节标记时，libdft采用动态的管理方式储存标注信息。因此使用字节标记时会造成更多开销。

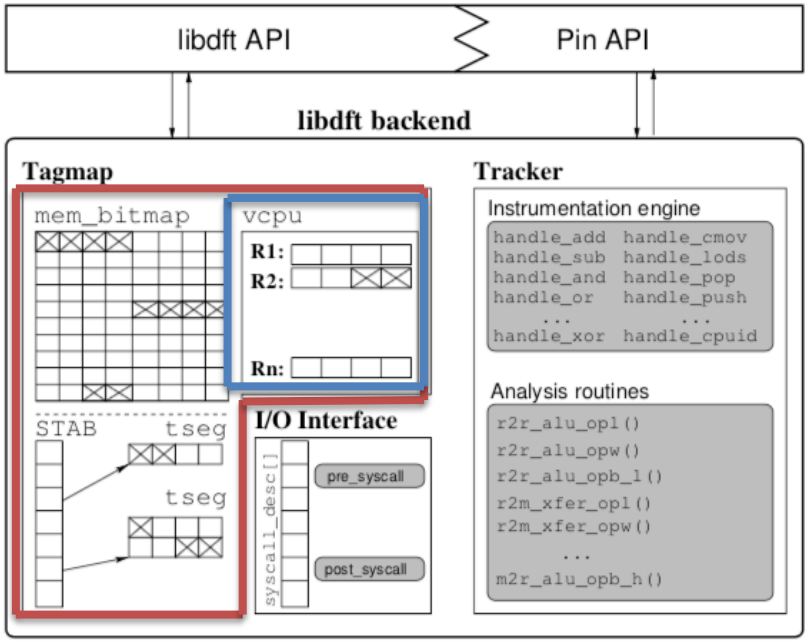
Libdft-原理

- STAB是一个虚拟段转换表，用以将虚拟地址映射到tagmap段中的相应字节
- 存在意义：程序在内存中是不连续的（逻辑和物理上都不连续）



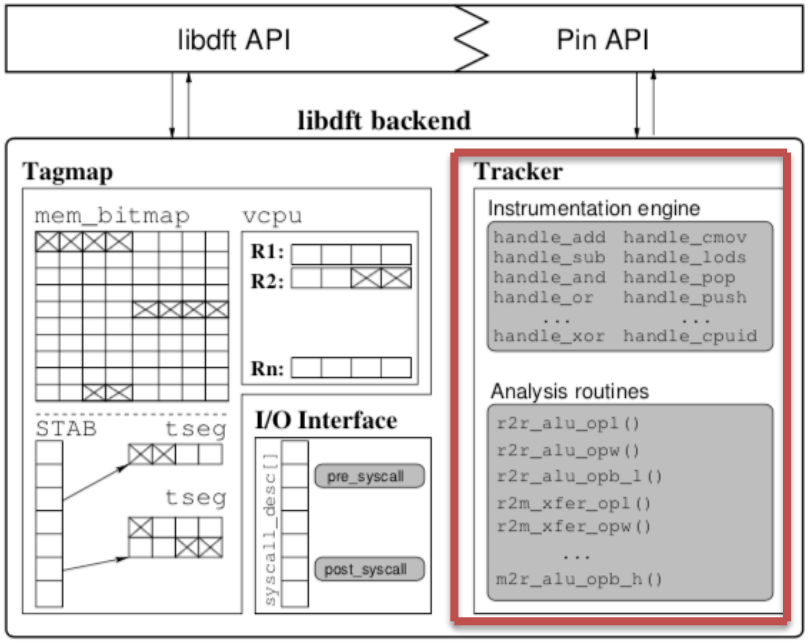
Libdft-原理

- vCPU用来保存CPU寄存器的标记信息。
- libdft为每一个线程都维护了一个vCPU结构，用来标记各个线程运行时的寄存器的状态。同时libdft会捕捉线程建立和销毁，从而可以动态的管理vCPU。



Libdft-原理

- Tracker是libdft的核心，是逻辑代码所在，包含：
- 1.指令引擎（instrumentation engine）
- 2.分析例程（analysis routines）



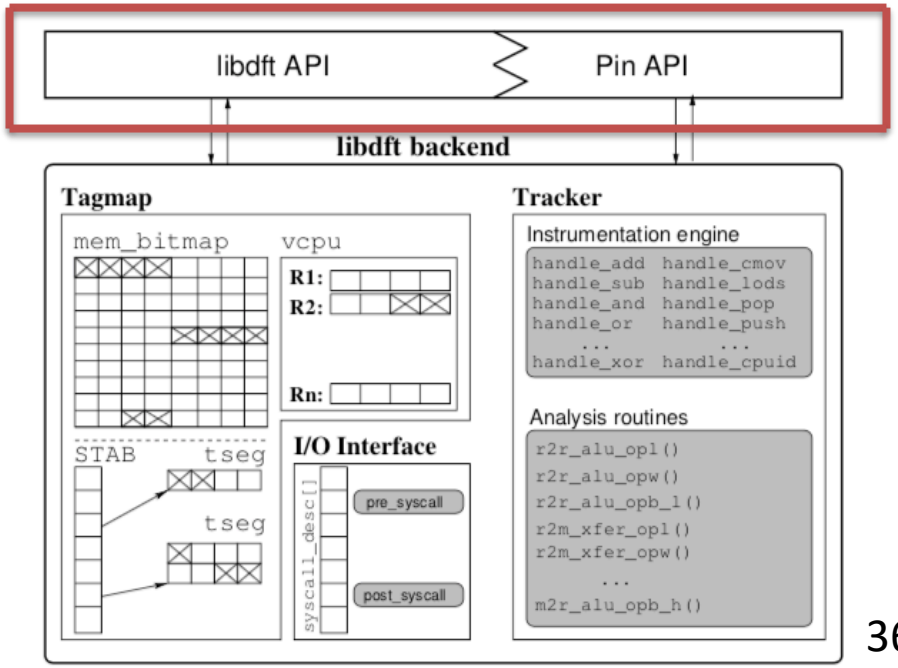
Libdft-原理

- 指令引擎：负责检查程序指令，以确定应对不同的情况时应该注入哪种分析程序，在这里libdft封装了Pin的API来检查每条指令的类型，并收集相应的信息（操作数、寄存器状态等）并调用分析例程对信息进行判断。

- 分析例程：包含为不同情况而实现污点追踪的逻辑代码，包括将sources数据添加污点标记、污点传播策略、无害化处理等操作。

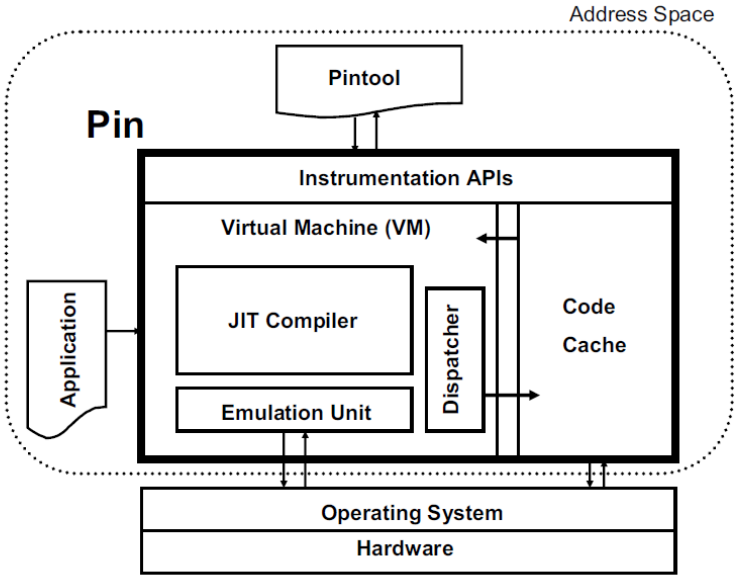
Libdft-原理

- Pin？
- Pin是Intel公司开发的动态插桩框架，可以实现指令级插桩操作。最主流的动态插桩工具
- 插桩？
- 在程序的特定位置插入指定的代码并执行。



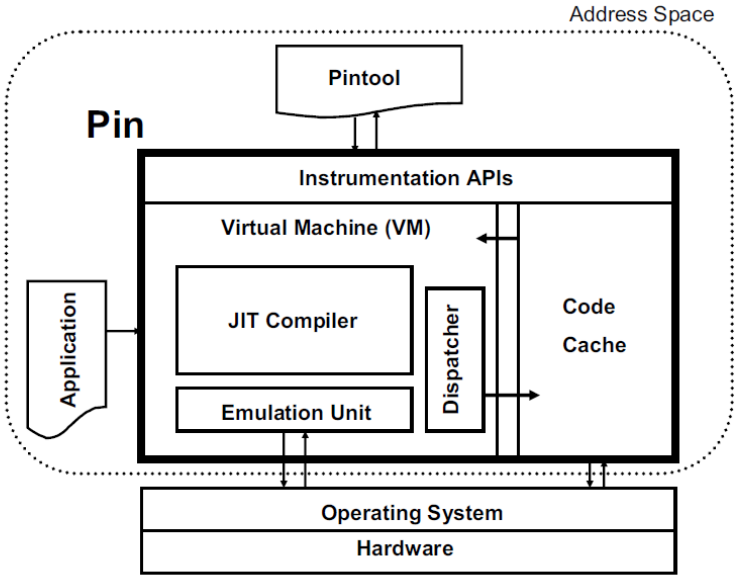
Libdft-原理

- 简要了解Pin原理
- 输入：二进制程序、Pintool



Libdft-原理

- Step1： Pin.exe 以Debug模式执行
- Step2：以子进程的形式激活检测对象Application.exe
- Step3：当子进程加载kernel32.dll后，阻塞子进程
  (使用ptrace API或Debugging API)
- Step4：注入pinvm.dll到目标应用程序中
- Step5：PinVM抢先运行并加载pintools.dll

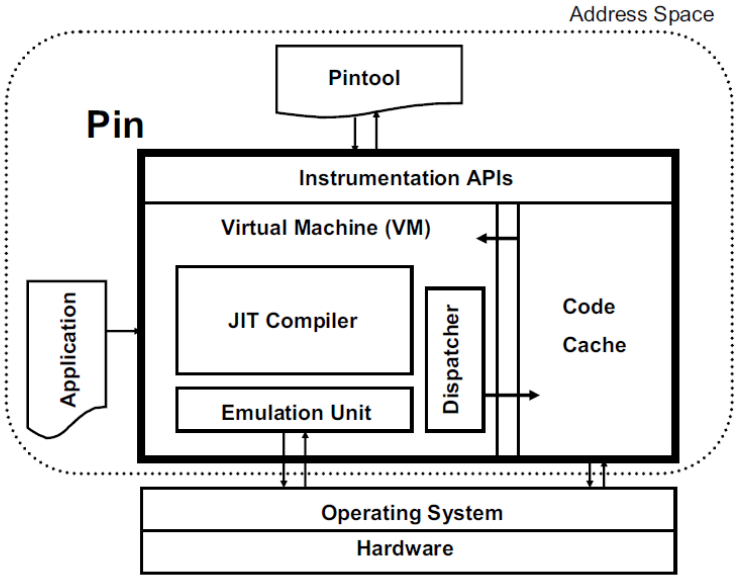


Libdft-原理

- Step6：对于目标程序的每一条指令根据Pintool中的API调用通过JIT Compiler即时“编译”成含有Pintool代码的新的指令并执行。

```
movzx   ecx, [rax+0x2]
call    0x77ef7870
cmp     rax, rdx
jz      0x77f1eac9
```

```
 ..some code
 ..some code
movzx   ecx, [rax+0x2]
 ..some code
 ..some code
call    0x77ef7870
 ..some code
 ..some code
cmp     rax, rdx
 ..some code
 ..some code
jz      0x77f1eac9
```



Libdft-原理

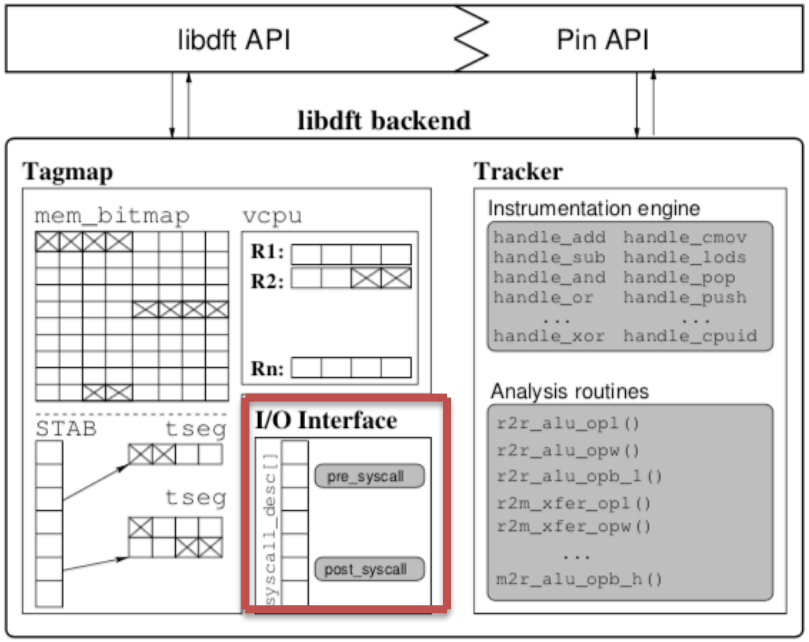
Pin提供了丰富的API，包括：

- 定义代码插桩粒度（指令级、流级、函数级、镜像级）
- 定义代码插入位置（指令执行前、后）
- 判断代码的类型（Mov ，JNZ ，CMP……）
- 获知CPU寄存器状态
- 写入内存的数据、位置、长度
- 当前、下一条指令的地址
- ……

Libdft-原理

- I/O接口（I/O Interface）负责监控用户进程和系统内核的I/O，用户可以在这里注册自己的回调函数，该回调函数包含了标记输入数据的规则。

- 程序大部分的外部输入输入都通过系统内核的传递（命令行、文件、socket等）



Libdft-原理

- libdft的常用API

Function	Description
`libdft_init()`	Initialize the tagging engine
`libdft_start()`	Commence execution
`libdft_die()`	Detach from the application

`ins_set_pre()`
`ins_set_post()`
`ins_set_clr()`
	Register instruction callbacks to be invoked before, after, or instead libdft's instrumentation

`syscall_set_pre()` `syscall_set_post()`	Hook a system call entry or return
`tagmap_set{b,w,l}()` `tagmap_setn()`	Tag {1, 2, 4} and `n` bytes of virtual memory

Table 1. Overview of libdft's API.

参考文献

- [1] Kemerlis V ， Portokalidis G ， Jee K , et al. libdft: Practical Dynamic Data Flow Tracking for Commodity Systems[J]. Acm Sigplan Notices, 2012, 47(7):121-132.
- [2] Luk C K . Pin : building customized program analysis tools with dynamic instrumentation[C]// Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation. ACM, 2005.

参考文献

- [3] 宋铮, 王永剑, 金波, et al. 二进制程序动态污点分析技术研究综述[J]. 信息网络安全, 2016(3):77-83.
- [4] 王夏菁, 胡昌振, 马锐, et al. 二进制程序漏洞挖掘关键技术研究综述[J]. 信息网络安全, 2017(8):1-13.
- [5] 张婧, 周安民, 刘亮, et al. 基于动态污点分析的栈溢出Crash判定技术[J]. 计算机工程, 2018, 44(4).
- [6] 董国良, 臧洌, 李航, et al. 基于污点分析的二进制程序漏洞检测[J]. 计算机技术与发展, 2018.
- [7] 王蕾, 李丰, 李炼, et al. 污点分析技术的原理和实践应用[J]. 软件学报, 2017(04):120-142.

参考文献

- [8] https://blog.csdn.net/stonesharp/article/details/51693336
- [9] https://blog.csdn.net/brucexu1978/article/details/8858689

谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。