27 secure-data-flow

数据流跟踪的作用：保护重要信息，防御可疑的输入信息

密码泄露原因

1、**KeyLogger**：在手机上安装键盘记录器/触摸记录器

聊天时谈到某个产品，淘宝就马上推荐了？输入法在作怪

2、Phishing：安装恶意软件并诱使用户输入密码

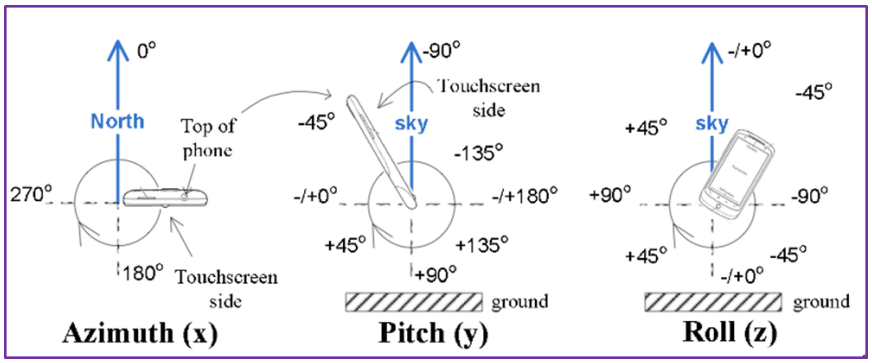
3、MemScan: 使用rootkit并扫描内存中的任何纯文本

/dev/pmem 777 物理地址泄露

4、Screen Capture 截屏

5、Cold-boot：通过物理攻击扫描内存获取文本内容

6、Side Channel：陀螺仪



Tainting: Data Flow Tracking（污迹追踪监控机制）

敏感数据的存活时间应该尽量小，换页、休眠、虚拟机挂起和核心转储(core dump)等都有可能会导致关键数据泄露。

Taint：遇到有颜色数据加密，海关不让含有颜色数据的包过

除色? 要把数据和表示颜色的数据紧密地绑定起来，taint被刷掉

其他数据怎样和taint产生关联的？

Dynamic Taint Analysis

```
i = get_input();
two = 2;
if(i%2 == 0){
  j = i+two;
  l = j;
} else {
  k = two*two;
  l = k;
}
jmp l;
```

Variable	Value	Taint Status
i	6	true
two	2	false
j	8	true
l	8	true

Variable	Value	Taint Status
i	7	true
two	2	false
k	4	false
l	4	false

关联信息存到表里，谁负责表格的维护？

编译器！

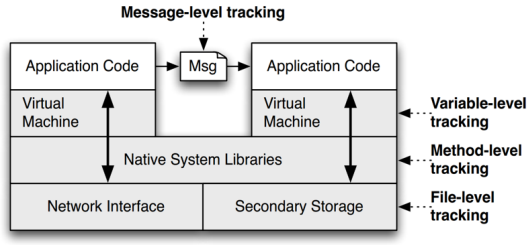
速度慢了10倍qwq

第一个i=6是true，是因为是get_input()输入的数据，而two=2是false是因为它和用户输入无关。j=8是true，是因为它由j=i+two产生，可以理解为taint这里有一个or操作，i是true，two是false，那j就是true。同理，l是true。而下面那个表，i同理是true，two=2也是false但此时由于控制流中，走得是else，因此k和l的产生都与i无关，所以它们的taint是false。

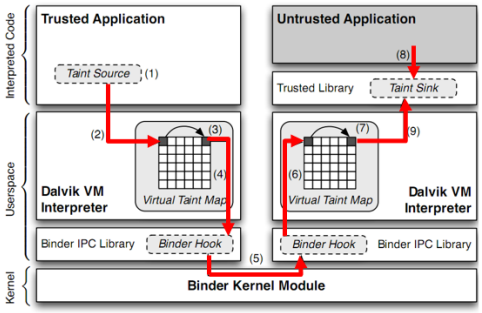
IMEI是每个设备专有的，属于隐私数据

安卓APP以前是以bytecode形式呈现的，要用JVM转成binary，在JVM转binary的过程中可以加很多taint。需要将taint粒度粗化，如果一个文件有一个地方是taint，会把整个文件标成taint（减少taint数量），使得overhead控制在30%以内。

TaintDroid



TaintDroid Data Flow



《BUG猎手的自我修养》

How does a Hacker Search a Bug?

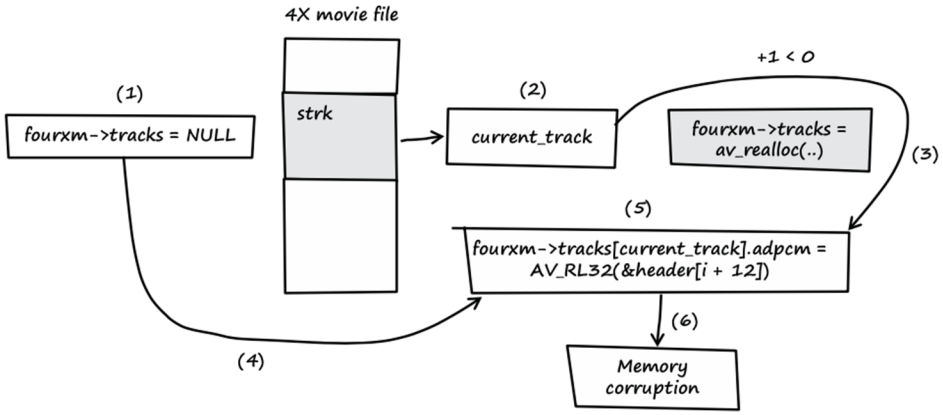
Step-0: Chose a library (open-source, of course) → 少走的路bug多

- Let's try ffmpeg, which is used in Chrome, VLC, Mplayer...
- There are also rumors that YouTube uses it for conversion

Step-1: List the demuxers of ffmpeg

Step-2: Identify the input data

Step-3: Trace the input data



Taint检测模型&性能影响

TaintCheck Detection Modules

TaintSeed: Mark untrusted data as tainted

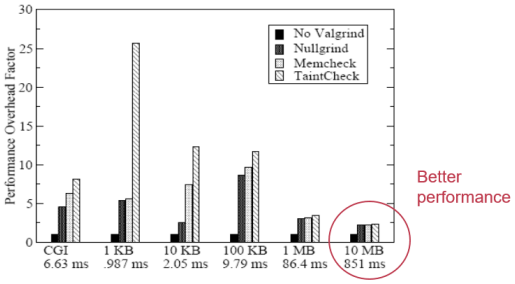
TaintTracker: Track each instruction, determine if result is tainted

TaintAssert: Check is tainted data is used dangerously

Figure 1. TaintCheck detection of an attack. (Exploit Analyzer not shown).

Performance Overhead of Apache

A more representative case, network and I/O

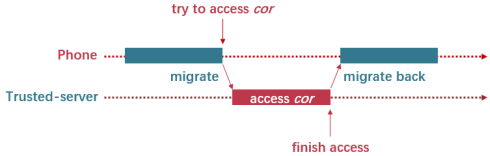


目标：No data to protect

识别出关键的数据并加密（逐步加密成密文），输入密码后又变成明文

Security-oriented offloading

- An app is migrated to the trusted node when accesses *cor*
- The offloading engine is based on COMET [OSDI'12]



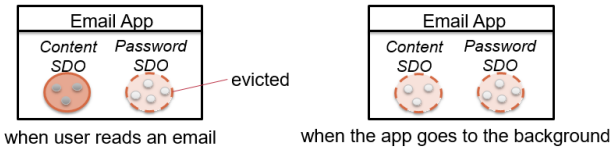
CleanOS [OSDI'12]: New Abstraction: *SDO*

Sensitive Data Object

Applications create SDOs and add sensitive data to them

CleanOS implements three functions for SDOs:

1. Tracks data in SDOs using taint tracking
2. Evicts SDOs to a trusted cloud whenever idle
3. Decrypts SDO data when it is accessed again



安全通道

加密的hash，不可逆；hash可以用来确定是否数据内容被人修改了

Encryption Properties

```
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message
```

```
encrypt(34fbcbd1, "hello, world") = 0x47348f63a67926cd393d4b93c58f78c
decrypt(34fbcbd1, "0x47348f63a67926cd393d4b93c58f78c") = hello, world
```

property: given the `ciphertext`, it is (virtually) impossible to obtain the `message` without knowing the `key`

```
MAC(key, message) → token
```

```
MAC(34fbcbd1, "hello, world") = 0x59cccc95723737f777e62bc756c8da5c
```

property: given the `message`, it is (virtually) impossible to obtain the `token` without knowing the `key`

(it is also impossible to go in the reverse direction)

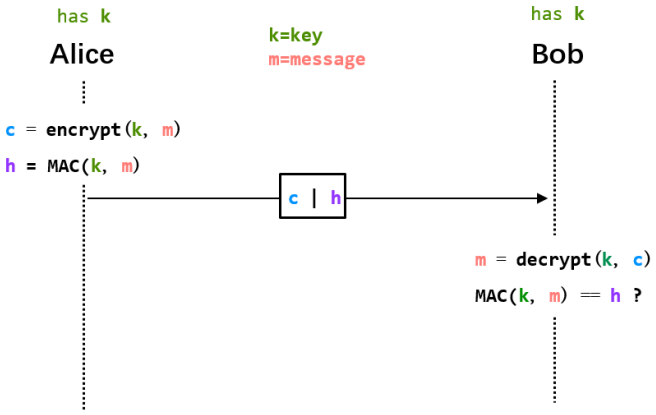
加密：key+message

解密：加密文件+key

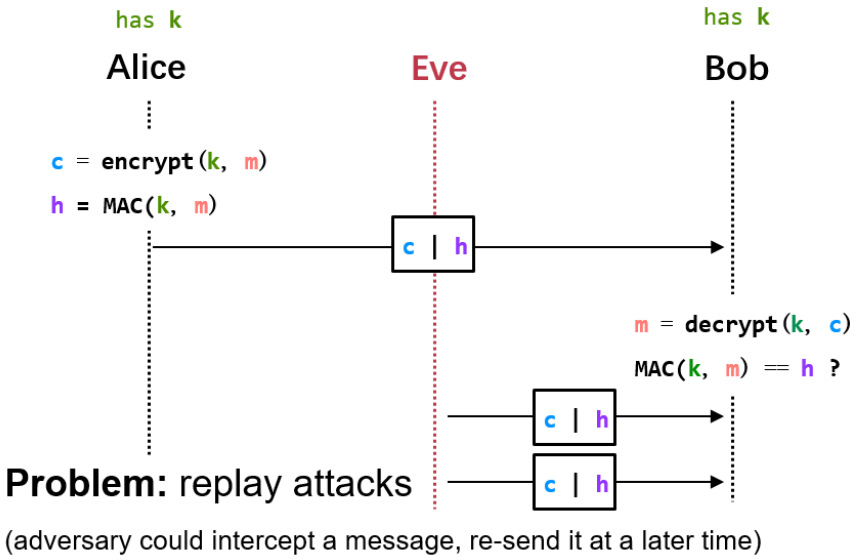
效果：没有key很难看到message

MAC ：只有message没有key几乎不可能得到token

正常过程：

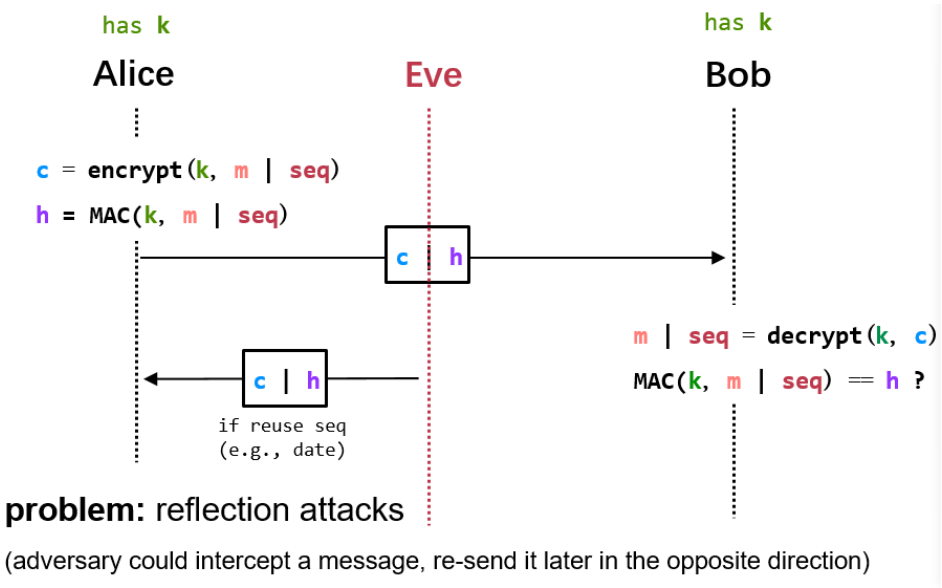


Eve截获信息后冒充Alice给Bob发信息？

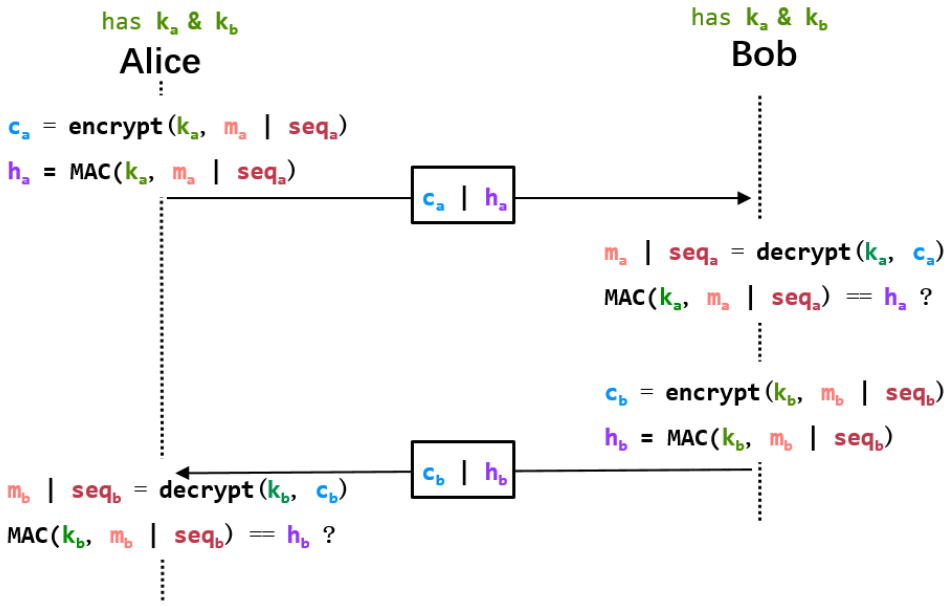


解决办法：在加密信息里多加一个信息序列的编号

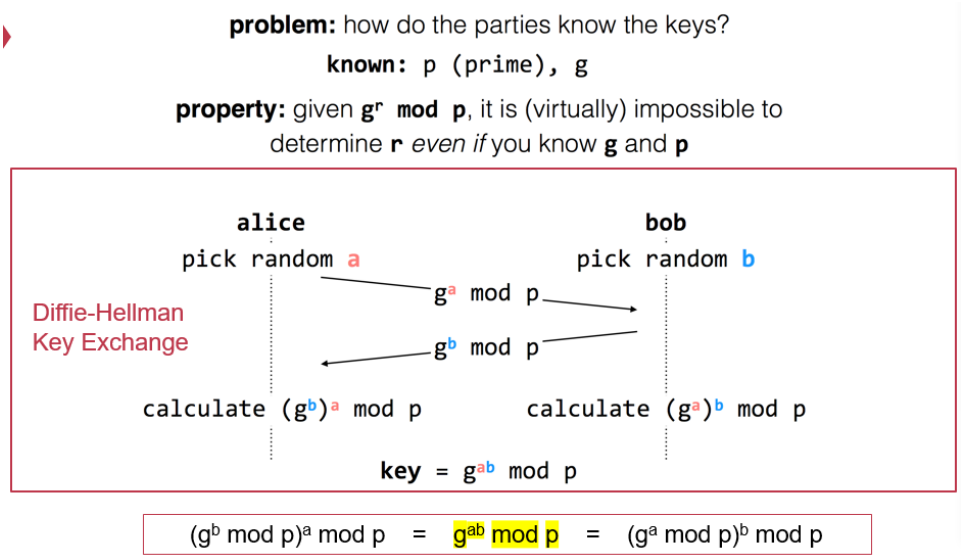
新的问题：如果Alice和Bob会互发消息，Bob可能也会使用相同的序列号给Alice发信息。这样Eve可以截获一条Alice的消息，稍后再发送给Alice，让Alice以为是Bob发的



解决办法：双方的key不同，而且都知道自己和对方的key



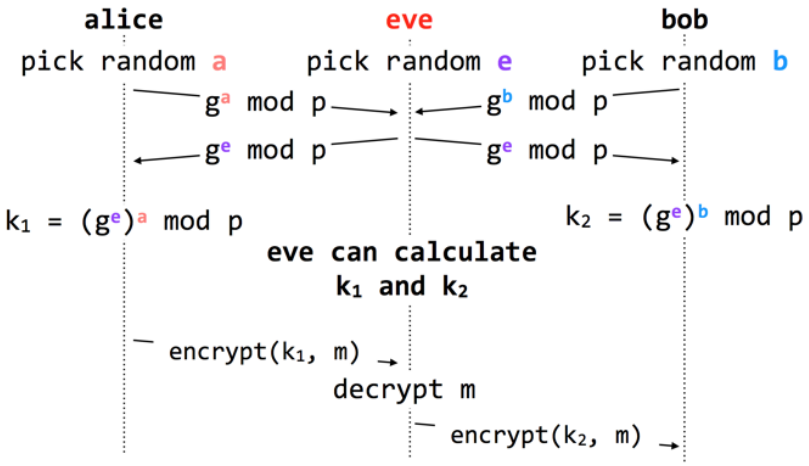
双方如何统一key?（要考虑信息在网络种传输可能被窃取）



交换密码

潜在问题：不能保证不被冒充

Diffie-Hellman Key Exchange & Man-in-the-Middle



problem: alice and bob don't know they're not communicating directly

RSA公开密钥密码体制

- **向某一个人发信息，不希望其它人知道**：用对方的公钥加密，只有对方可以用自己的私钥解密
- **向所有人发一个公告**：用自己的私钥签名，大家可以用你的公钥来验证你的身份
- **找一个第三方(CA)**：存Bob和Bob的公钥加上自己的签名，作为一个证书。全世界只有十几个，所有人都**无条件相信**，被浏览器预存。

  谁负责CA？民间组织管理，有风险

  如果签了一个不该签的：召回证书，要定期检查有哪些证书

信任与安全

编译器的代码有bug：

Trusting Trust: Some Observations

Stage I:

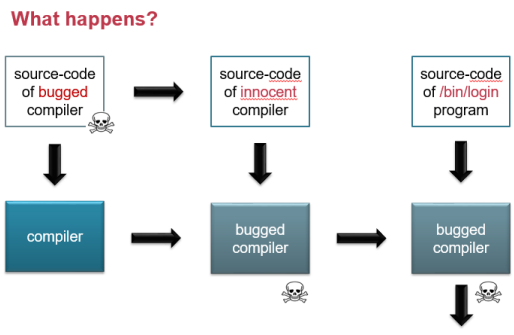
– A program can, when executed, output its own source-code

Stage II:

– A compiler can *learn* the meaning of a symbol

Stage III:

– A compiler may (deliberately) output incorrect machine code



不要相信任何不是完全由你自己写的代码！XcodeGhost风波，Unix万能密码

信任的根源

相信OS中的核心部分→微内核，相信CPU(硬件) ——但有可能bit翻转

因为可能被植入恶意bug，所以要尽量给最小的权限

TCB: Trusted Computing Base

TCB is the parts that are trusted

- Process never trust another process, but trusts all its threads
- OS never trust a process, but trusts hardware
- VMM never trust a VM, but trust hardware
- Which parts do you trust in your laptop? Your phone?

TCB is the only metric of security

- Bug is inevitable

Root of Trust

There must be some thing to trust

Other trusts are based on the root of trust

- Aka., *bootstrap*

Root of trust can be software or hardware

- Software: BIOS, VMM, kernel loader, kernel, etc.
- Hardware: TPM, secure processor, etc.

BIOS→kernel 逐步验证

可信平台模块TPM：提供物理保护，判断OS是否合法（微软win11有装TPM2.0，只允许电脑装一个OS）。TPM有一个内嵌的私钥，每一个TPM都是唯一的，像一个钥匙。

TPM的作用：

1、私钥的密封存储：物理上安全（大部分时间）

2、用私钥签名：用于远程认证

3、用散列链度量：用于本地信任链