网络相关

网络劫持

知识文档：《浅析劫持》

《iOS开发安全须知》 这篇比较进阶，目前对于iOS开发还不需要阅读，应用的SDK一般也会做这些事。

初步了解

网购、直播、人工智能等的发展->网络流量的爆发->运营商为节约成本、给用户投放广告->劫持的产生

终端用户发起访问请求；
流量通过网络出口对外发起访问；
访问流量被镜像一份给劫持系统的DPI设备；
4.DPI对流量进行分析判断，比如http get、80端口等数据
缓存系统判断是否热点资源，比如连续请求5次的相同内容；
给用户发送响应请求，告诉客户本地即是客户需要访问的内容；
由于本地的缓存系统离客户更近，所以客户更早收到缓存系统的响应；
用户和本地的缓存系统建立网络交互，源站的响应回来的晚，会自动断开；
如果本地有缓存内容，则给用户响应内容，如果本地没有，会计算访问次数。当达到响应的内容时。

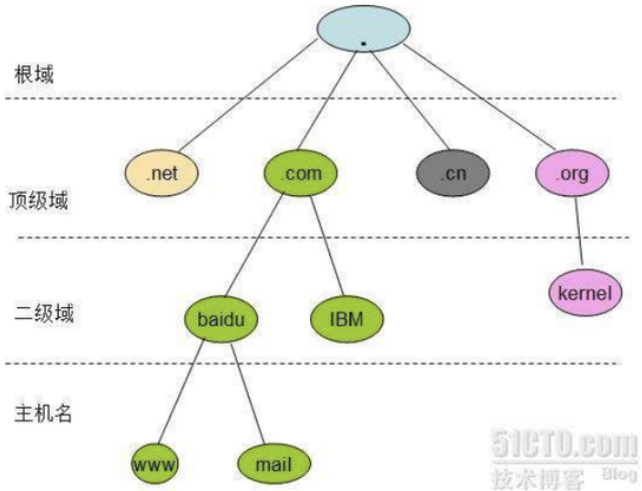
篡改用户请求的目的地地址，从异常服务器响应请求，或者直接篡改真实服务器的响应内容。
用户的敏感信息可能泄露，或接受错误响应。

DNS劫持

DNS劫持

DNS服务器劫持

DNS劫持指篡改了某个域名的解析结果，使得指向该域名的IP变成了另一个IP，导致对相应网址的访问被劫持到另一个不可达的或者假冒的网址。



www.baidu.com.root

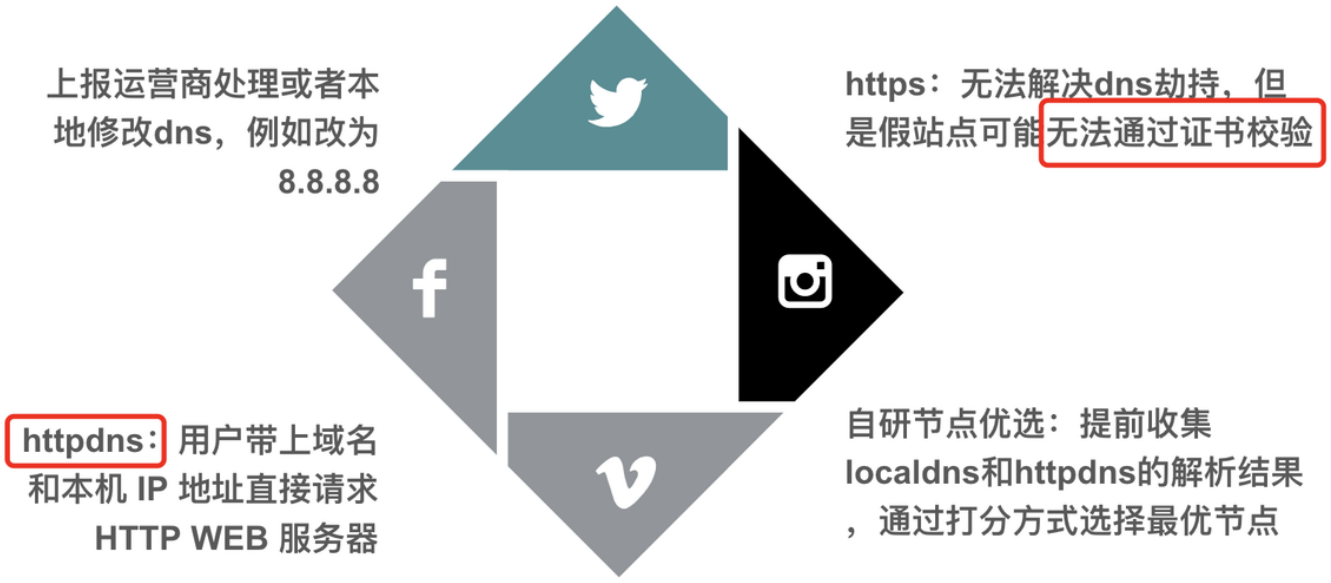
DNS劫持

本地DNS劫持

系统感染病毒后，将本地的dns缓存偷偷篡改掉。

如右图，直接篡改host文件指定服务器IP。

解决方案



关于HTTPDNS：在TTNET视频里看到过介绍。找了篇原理：HTTPDNS原理是什么

DNS解析流程：

1. 浏览器中输入 www.linkedkeeper.com，发出解析请求。
2. 本机的域名解析器 resolver 程序查询本地缓存和 host 文件中是否为域名的映射关系，如果有则调用这个 IP 地址映射，完成解析。
3. 如果 hosts 与本地解析器缓存都没有相应的网址映射关系，则本地解析器会向 TCP/IP 参数中设置的首选 DNS 服务器（我们叫它 Local DNS 服务器）发起一个递归的查询请求。
4. 服务器收到查询时，如果要查询的域名由本机负责解析，则返回解析结果给客户机，完成域名解析，此解析具有权威性。如果要查询的域名，不由 Local DNS 服务器解析，但该服务器已缓存了此网址映射关系，则调用这个 IP 地址映射，完成域名解析，此解析不具有权威性。
5. 如果 Local DNS 服务器本地区域文件与缓存解析都失效，则根据 Local DNS 服务器的设置（是否递归）进行查询，如果未用开启模式，Local DNS 就把请求发至13台 Root DNS。如果用的是递归模式，此 DNS 服务器就会把请求转发至上一级 DNS 服务器，由上一级服务器进行解析，上一级服务器如果不能解析，或找根 DNS 或把转请求转至上上级，以此循环。
6. Root DNS 服务器收到请求后会判断这个域名是谁来授权管理，并会返回一个负责该顶级域名服务器的一个 IP。
7. Local DNS 服务器收到 IP 信息后，将会联系负责 .com 域的这台服务器。
8. 负责 .com 域的服务器收到请求后，如果自己无法解析，它就会找一个管理 .com 域的下一级 DNS 服务器地址给本地 DNS 服务器。

9. 当 Local DNS 服务器收到这个地址后，就会找 linkedkeeper.com 域服务器，10、11重复上面的动作，进行查询。
10. 最后 www.linkedkeeper.com 返回需要解析的域名的 IP 地址给 Local DNS 服务器。
11. Local DNS 服务器缓存这个解析结果（同时也会缓存，6、8、10返回的结果）。
12. Local DNS 服务器同时将结果返回给本机域名解析器。
13. 本机缓存解析结果。
14. 本机解析器将结果返回给浏览器。
15. 浏览器通过返回的 IP 地址发起请求。

HTTPDNS原理：
HTTPDNS 利用 HTTP 协议与 DNS 服务器交互，代替了传统的基于 UDP 协议的 DNS 交互，绕开了运营商的 Local DNS，有效防止了域名劫持，提高域名解析效率。另外，由于 DNS 服务器端获取的是真实客户端 IP 而非 Local DNS 的 IP，能够精确定位客户端地理位置、运营商信息，从而有效改进调度精确性。

HTTPDNS解决的问题：
- Local DNS 劫持：由于 HttpDns 是通过 IP 直接请求 HTTP 获取服务器 A 记录地址，不存在向本地运营商询问 domain 解析过程，所以从根本避免了劫持问题。
- 平均访问延迟下降：由于是 IP 直接访问省掉了一次 domain 解析过程，通过智能算法排序后找到最快节点进行访问。
- 用户连接失败率下降：通过算法降低以往失败率过高的服务器排序，通过时间近期访问过的数据提高服务器排序，通过历史访问成功记录提高服务器排序。

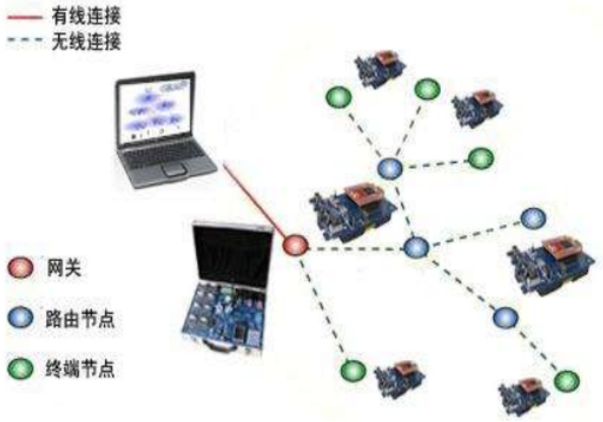
HTTP劫持

HTTP劫持

用户和目标节点经过N跳：路由器、交换机、服务器等等，且http不加密，每一跳都存在劫持风险。

中间某跳劫持后响应302给用户，并且响应快于真实服务器，导致后者的响应被拒绝。用户到302给的地址请求。

对比文件大小
抓包



运营商如果出于节省成本的目的，会对热点文件做镜像到某台服务器，通过302方式直接响应文件。

解决方案

使用HTTPS：内容加密，运营商无法做标记

拆包：一般只会检测TCP连接建立后的第一个数据包，如果其是一个完整的HTTP协议才会被标记。所以可以将HTTP请求分拆到多个数据包内，躲过运营商的标记。

自研协议：cdn推出自研协议，如网宿的wip协议

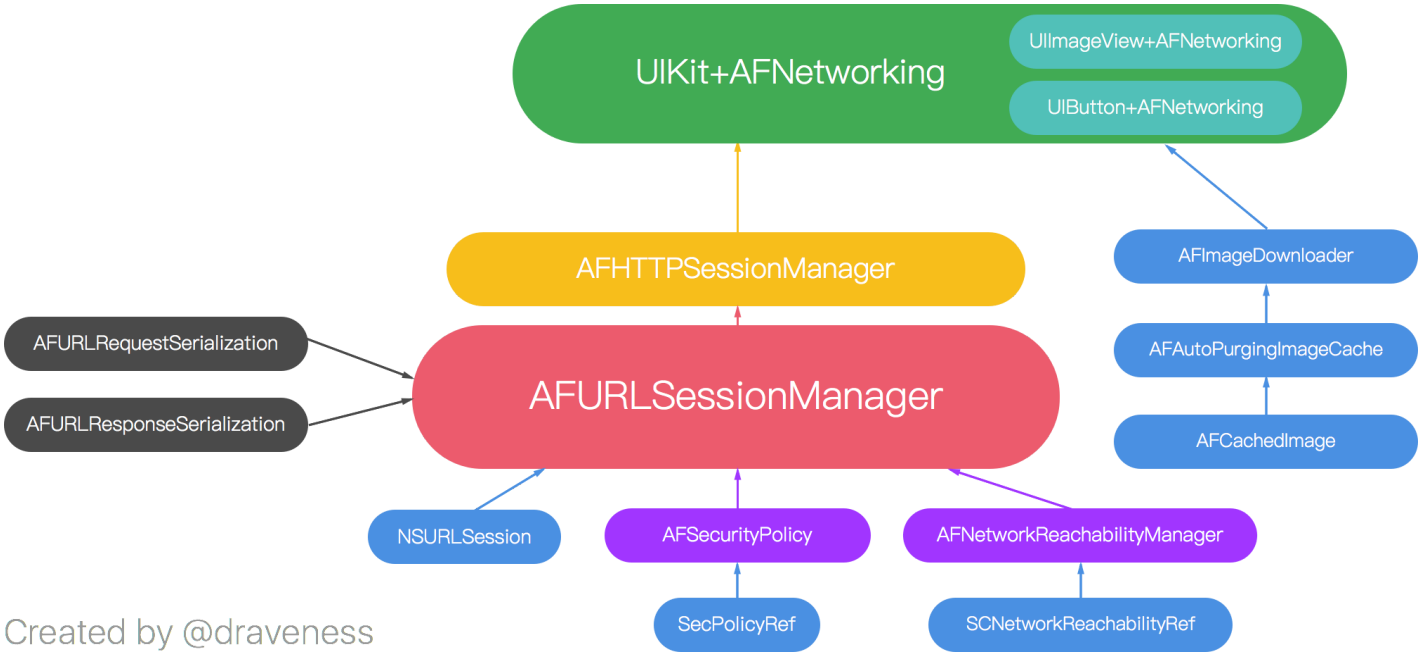
使用HTTPS带来的问题是性能有所降低，因为多了TLS（HTTP2有提升性能）；
DNS劫持可能带来HTTPS不可用，导致白屏。

还可以使用加密代理，在客户端与web服务器之间添加一个代理服务器，在用户和代理之间经过运营商节点，通过各种加密方式保证安全，只要确保代理服务器不被劫持，就可以避开HTTP劫持。

AFNetworking阅读笔记

https://draveness.me/afnetworking1 参考资料

- AFNetworking是iOS开发中必不可少的组件之一。
- 可以说**使用 AFNetworking 的工程师构成的社区**才使得它变得非常重要。



NSURLSession

提供下载内容的api，支持一系列身份验证，支持后台下载。

- 使用步骤
  - 实例化一个 `NSURLRequest/NSMutableURLRequest`，设置 URL
  - 通过 `- sharedSession` 方法获取 `NSURLSession`
  - session 上调用 `- dataTaskWithRequest:completionHandler:` 方法返回一个 `NSURLSessionDataTask`
  - 向 data task 发送消息 `- resume`，开始执行这个任务
  - 在 completionHandler 中将数据编码，返回字符串

```
1 NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:[[NSURL
  alloc] initWithString:@"https://github.com"]];
2 NSURLSession *session = [NSURLSession sharedSession];
3 NSURLSessionDataTask *task = [session dataTaskWithRequest:request
```

```
                                        completionHandler:^(NSData * _Nullable data, NSURLResponse * _Nullable response, NSError * _Nullable error) {
                                            NSString *dataStr = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
                                            NSLog(@"%@", dataStr);
                                        }];
[task resume];

// 运行以上代码会在控制台打印出github首页的html
```

AFNetworking

- 使用步骤
  - 以服务器的**主机地址或者域名**生成一个 AFHTTPSessionManager 的实例
  - 调用 `- GET:parameters:progress:success:failure:` 方法

```
AFHTTPSessionManager *manager = [[AFHTTPSessionManager alloc] initWithBaseURL:[[NSURL alloc] initWithString:@"hostname"]];[manager GET:@"relative_url" parameters:nil progress:nil
    success:^(NSURLSessionDataTask * _Nonnull task, id  _Nullable responseObject) {
        NSLog(@"%@" ,responseObject);
    } failure:^(NSURLSessionDataTask * _Nullable task, NSError * _Nonnull error) {
        NSLog(@"%@", error);
    }];

```

如果要发送不安全的http请求，在info.plist中添加关键字。
默认接受json。

AFNetworking 的调用栈

在这一节中我们要分析一下在上面两个方法的调用栈，首先来看的是 `AFHTTPSessionManager` 的初始化方法 `-initWithBaseURL:`

```
- [AFHTTPSessionManager initWithBaseURL:]
    - [AFHTTPSessionManager initWithBaseURL:sessionConfiguration:]
        - [AFURLSessionManager initWithSessionConfiguration:]
            - [NSURLSession sessionWithConfiguration:delegate:delegateQueue:]
            - [AFJSONResponseSerializer serializer] // 负责序列化响应
            - [AFSecurityPolicy defaultPolicy] // 负责身份认证
            - [AFNetworkReachabilityManager sharedManager] // 查看网络连接情况
        - [AFHTTPRequestSerializer serializer] // 负责序列化请求
        - [AFJSONResponseSerializer serializer] // 负责序列化响应
```

从这个初始化方法的调用栈，我们可以非常清晰地了解这个框架的结构：

- 其中 `AFURLSessionManager` 是 `AFHTTPSessionManager` 的父类
- `AFURLSessionManager` 负责生成 `NSURLSession` 的实例，管理 `AFSecurityPolicy` 和 `AFNetworkReachabilityManager`，来保证请求的安全和查看网络连接情况，它有一个 `AFJSONResponseSerializer` 的实例来序列化 HTTP 响应
- `AFHTTPSessionManager` 有着**自己的** `AFHTTPRequestSerializer` 和 `AFJSONResponseSerializer` 来管理请求和响应的序列化，同时**依赖父类提供的接口**保证安全、监控网络状态，实现发出 HTTP 请求这一核心功能

- AFNetworking其实是对NSURLSession的高度封装，提供一些简单易用的接口。
- 该博客接下来对AFNetworking展开进阶分析，先搁置。

RPC详解

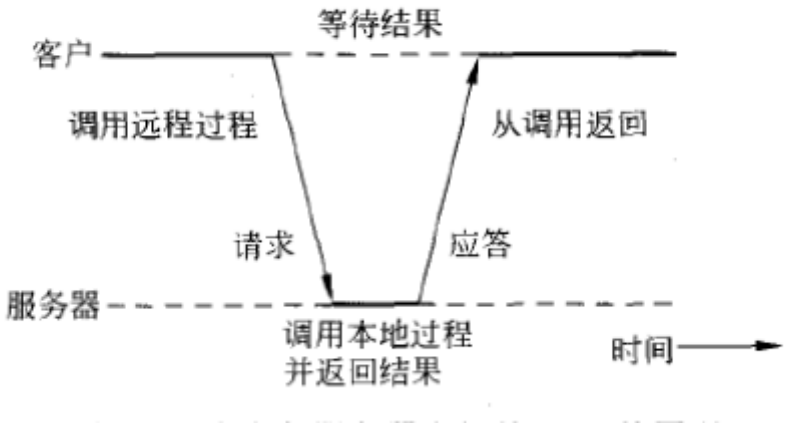
https://waylau.com/remote-procedure-calls/

概念

远程过程调用Remote procedure call。
RPC 是指计算机 A 上的进程，调用另外一台计算机 B 上的进程，其中 A 上的调用进程被挂起，而 B 上的被调用进程开始执行，当值返回给 A 时，A 进程继续执行。调用方可以通过使用参数将信息传送给被调用方，而后可以通过传回的结果得到信息。而这一过程，对于开发人员来说是透明的。

远程过程调用采用客户机/服务器(C/S)模式。请求程序就是一个客户机，而服务提供程序就是一台服务器。和常规或本地过程调用一样，远程过程调用是同步操作，在远程过程结果返回之前，需要暂时中止请求程序。使用相同地址空间的低权进程或低权线程允许同时运行多个远程过程调用。

过程



1. 客户过程以正常的方式调用客户存根；
2. 客户存根生成一个消息，然后调用本地操作系统；
3. 客户端操作系统将消息发送给远程操作系统；
4. 远程操作系统将消息交给服务器存根；
5. 服务器存根调将参数提取出来，而后调用服务器；
6. 服务器执行要求的操作，操作完成后将结果返回给服务器存根；
7. 服务器存根将结果打包成一个消息，而后调用本地操作系统；
8. 服务器操作系统将含有结果的消息发送给客户端操作系统；
9. 客户端操作系统将消息交给客户存根；
10. 客户存根将结果从消息中提取出来，返回给调用它的客户存根。

服务器存根是客户存根在服务器端的等价物，也是一段代码，用来将通过网络输入的请求转换为本地过程调用。

以上过程就是客户端对客户存根发出本地调用转化成对服务端发出对本地调用，而客户端和服务端都不会意识到中间过程的存在。

实现

- 如何传递参数?
  - 传递值参数

- 传递引用参数：实现困难，需要对引用创建副本
- 如何表示数据？
  - 遵循标准，避免不同机器的字节顺序、整数范围、浮点表示等差异
  - 隐式类型：只传递值，不传递变量类型或名称
  - 显式类型：传递每个字段的类型和值
- 如何选用传输协议？
- 出错？
  - 本地过程调用没有调用失败的概念，远程过程调用需要添加测试以及异常捕获
- 性能怎么样？
  - 比起本地过程调用自然是大大降低，但是这并不能阻止我们使用RPC
- 安全？
  - 客户端发送消息到远程过程，那个过程是可信的吗？
  - 客户端发送消息到远程计算机，那个远程机器是可信的吗？
  - 服务器如何验证接收的消息是来自合法的客户端吗?服务器如何识别客户端？
  - 消息在网络中传播如何防止时被其他进程嗅探？
  - 可以由其他进程消息被拦截和修改时遍历网络从客户端到服务器或服务器端？
  - 协议能防止重播攻击吗？
  - 如何防止消息在网络传播中被意外损坏或截断？