

# 端到端通讯安全 检测和增强

张源 博士

系统软件与安全实验室

复旦大学软件学院

IDF 2015 上海

# 第34次中国互联网状况统计报告

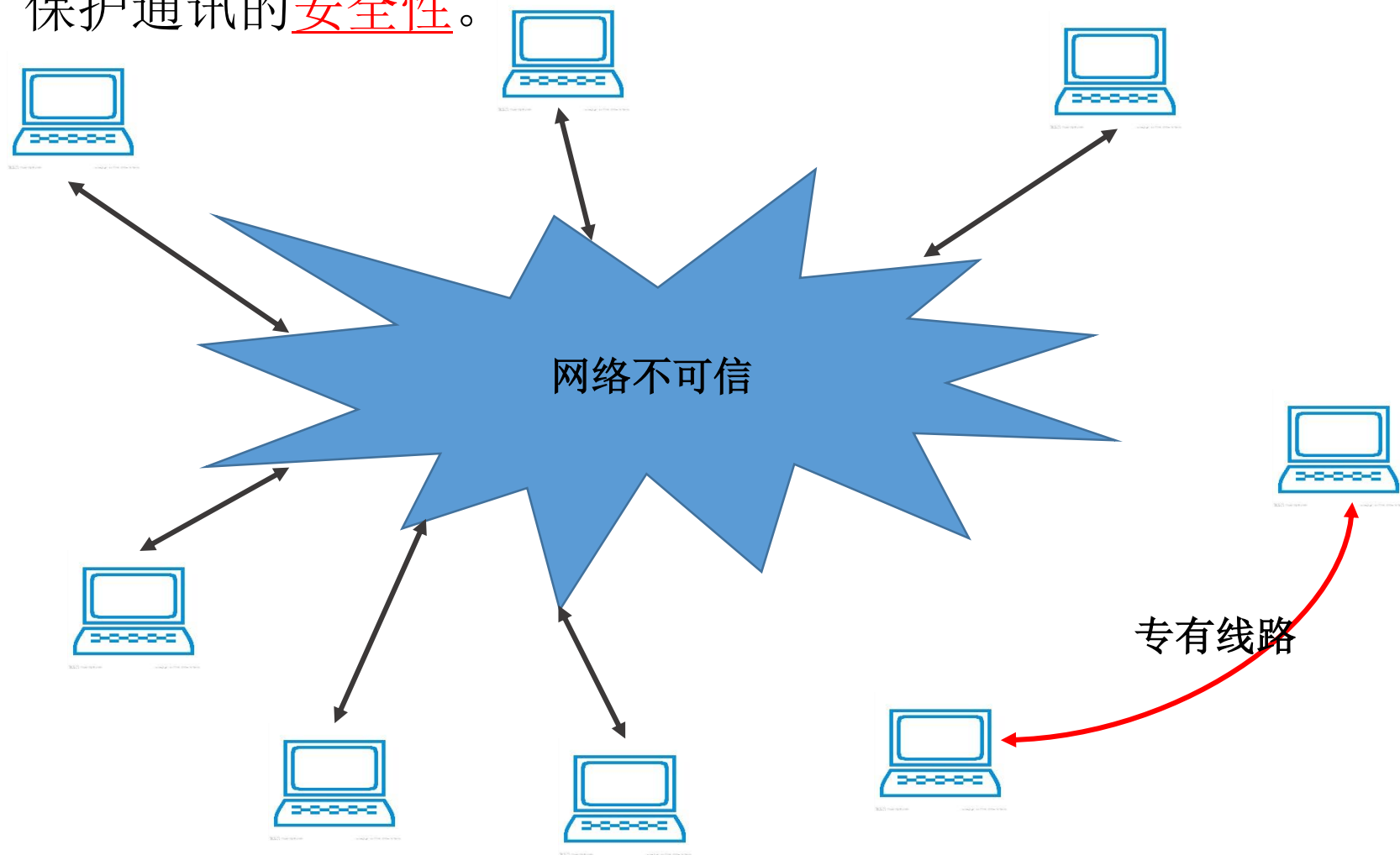
- 截至2014年6月，中国网民规模达6.32亿，互联网普及率为46.9%
- 中国手机网民规模达到5.27亿，手机上网的网民比例为83.4%，传统PC上网比例为80.9%

表：2013.12-2014.6中国网民各类网络应用使用率

	2014年6月		2013年12月		
应用	用户规模（万）	网民使用率	用户规模（万）	网民使用率	半年增长率
即时通讯	56423	89.3%	53215	86.2%	6.0%
网络购物	33151	52.5%	30189	48.9%	9.8%
网上支付	29227	46.2%	26020	42.1%	12.3%
网上银行	27188	43.0%	25006	40.5%	8.7%
互联网理财	6383	10.1%	--	--	--

# 什么是端到端通讯安全？

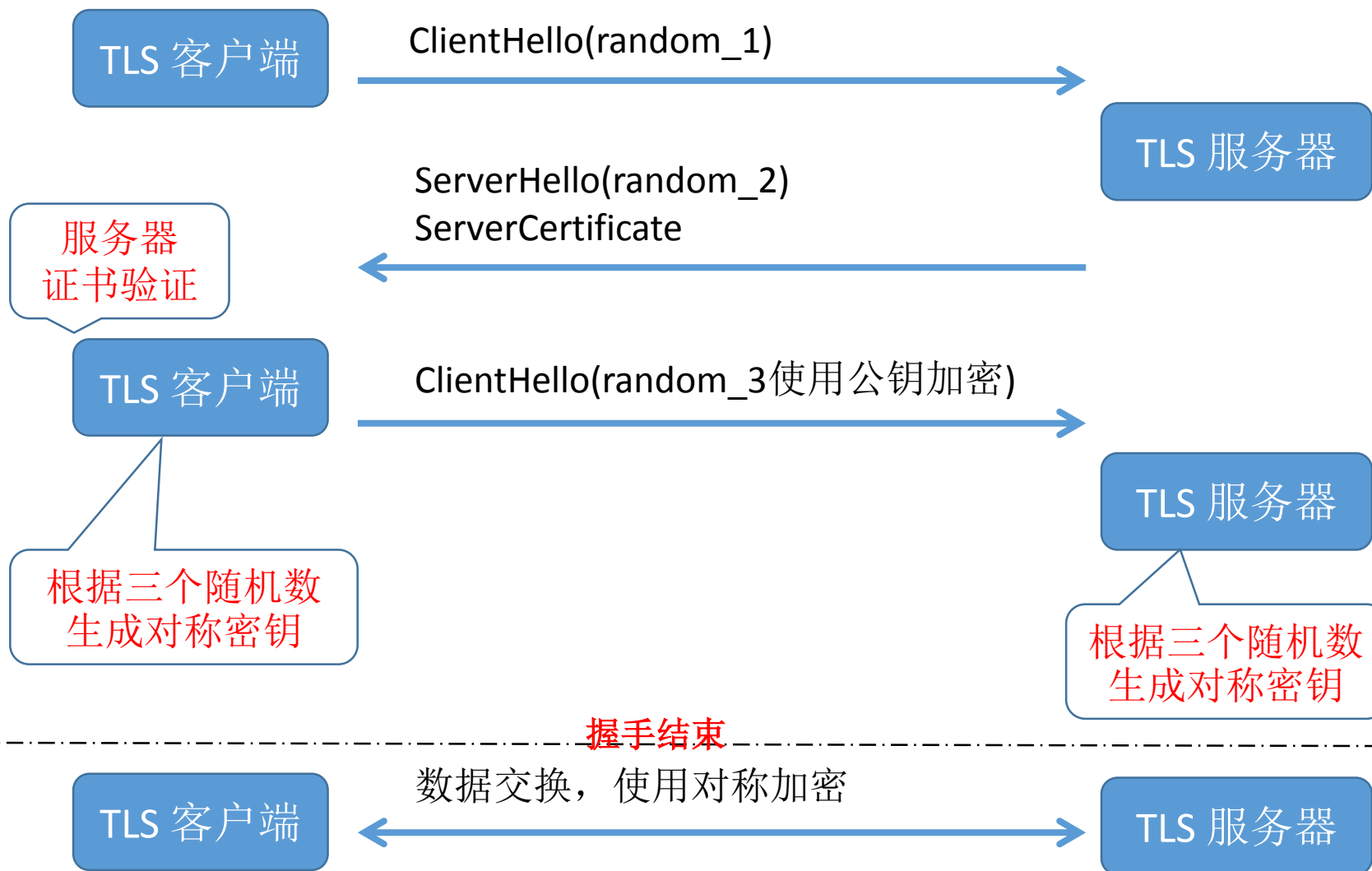
端到端通讯安全：在网络通讯连接不可信的情况下保护通讯的安全性。



# SSL/TLS

- 目标
  - 加密传输→防范窃听风险 (Eavesdropping)
  - 信息校验→防范篡改风险 (Tampering)
  - 身份证书→防范冒充风险 (Pretending)
- HTTPS: HTTP over SSL/TLS
- 版本
  - 1994年 SSL 1.0 NetScape
  - 1995年 SSL 2.0 NetScape
  - 1996年 SSL 3.0
  - 1999年 TLS 1.0 互联网标准化组织ISOC, SSL升级版
  - 2006年 TLS 1.1
  - 2008年 TLS 1.2
  - 2011年 TLS 1.2修订版

# 客户端服务器通讯过程



# X. 509证书

- 基本部分
  - 版本号
  - **证书持有人的公钥**：表明证书持有人的身份
  - 证书的序列号：由CA分配的唯一数字型编号
  - **主题信息**：证书持有人的唯一标示符，例如 CN=Yuan, OU=\*.fudan.edu.cn, O=Fudan University, C=CN
  - **有效期**：开始时间和日期、结束时间和日期
  - 认证机构：证书颁布者的名字
  - **颁布者的数字签名**：证书颁布者的签名，保证此证书颁布后没有被修改过
  - 签名算法标识符：颁布者认证证书时使用的签名算法
- 扩展部分
  - 证书用途，CRL分布点等

# 服务器证书验证

- 服务器发送过来的证书是否有效?
  - 在有效期内
  - 证书颁布者是否有效
- 服务器发送来的证书是否同连接的服务器匹配?
  - 主题信息是否与服务器匹配
  - 当前连接的地址是否与证书中的匹配

# 报告内容

- SSL/TLS简介
- HTTPS常见缺陷
- HTTPS缺陷检测
- HTTPS安全性增强

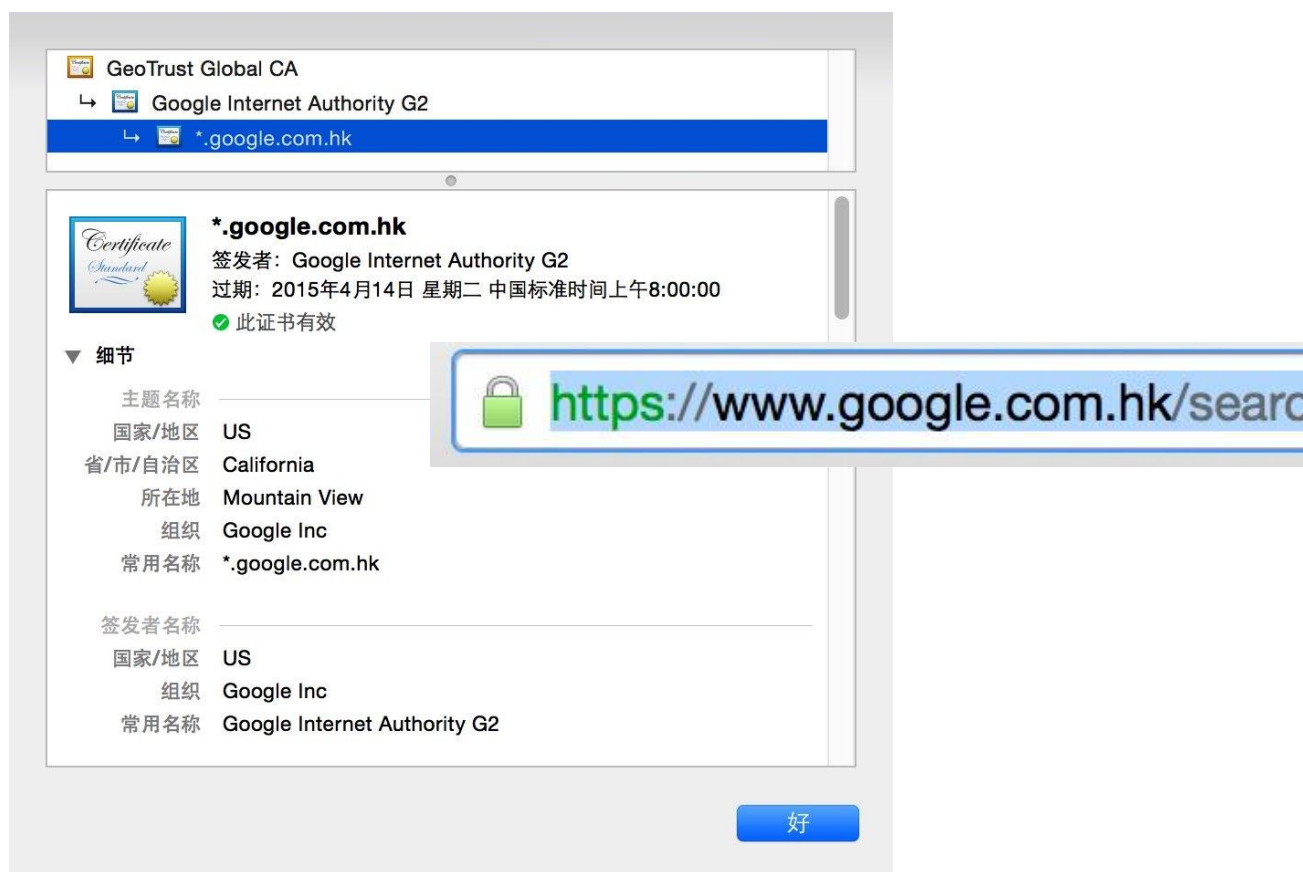


# HTTPS常见缺陷

---

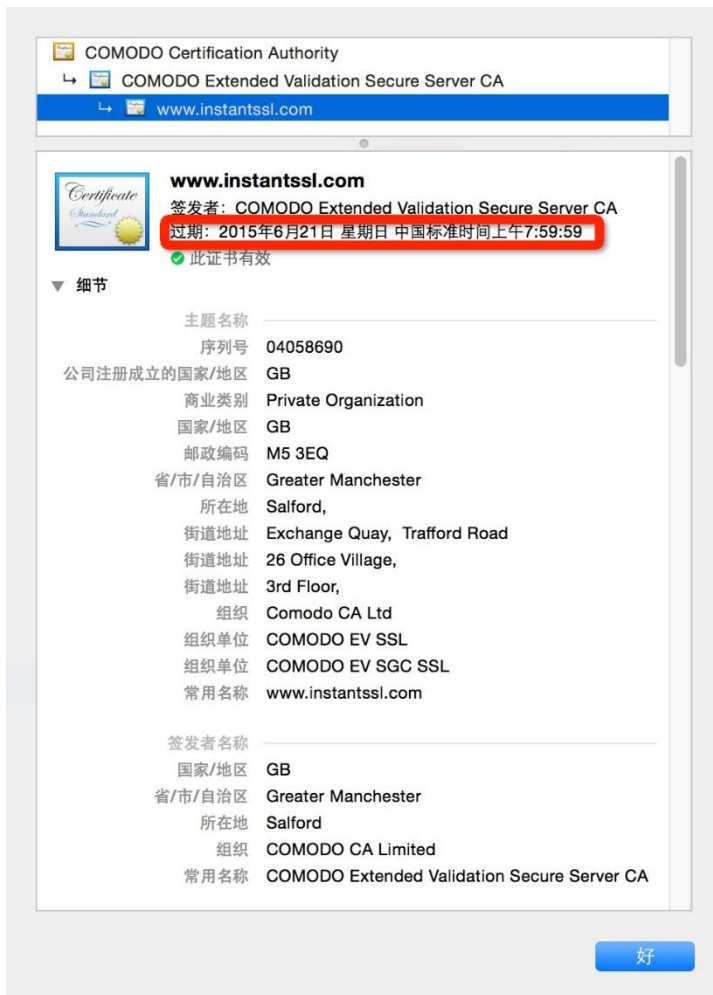
# 服务器域名验证缺陷

- 如果存在缺陷，会导致客户端信任颁发给其他域名的合法证书



# 证书有效期验证缺陷

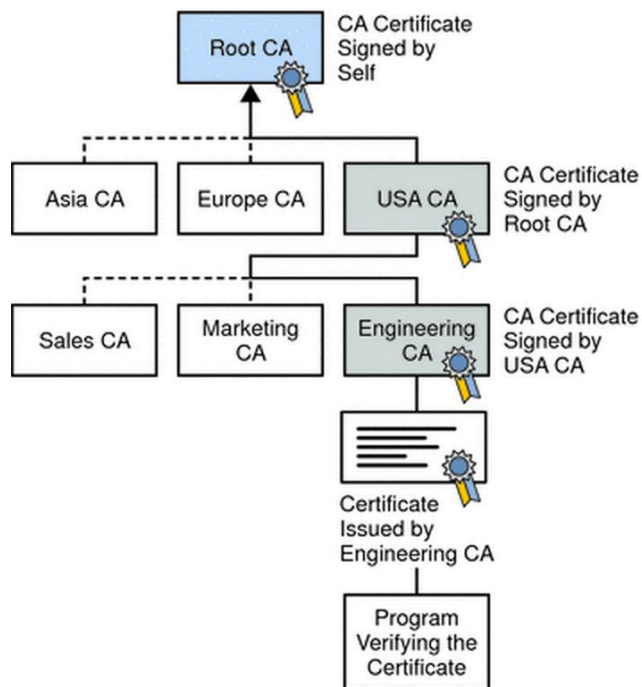
- 如果存在缺陷，会导致客户端信任一个已经过期的证书



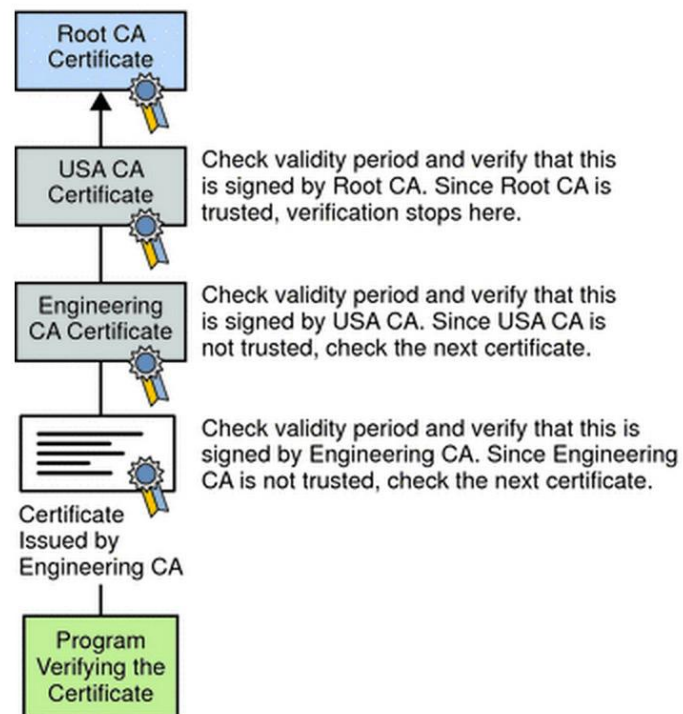
# 证书链验证缺陷

- 是否验证证书链上每一级证书是否**真实地**使用证书颁布者的私钥签名

Figure 5-4 Certificate Chain



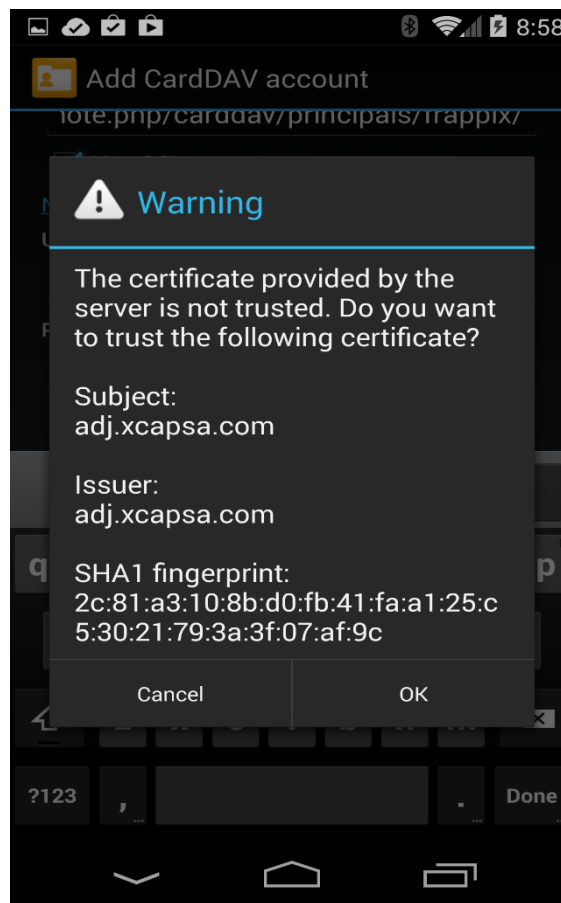
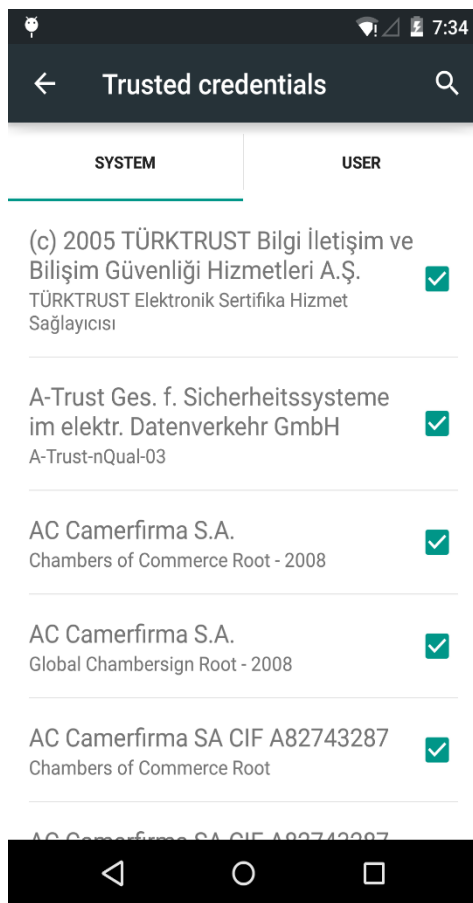
■ Trusted Authority  
■ Untrusted Authority



■ Trusted Authority  
■ Untrusted Authority

# 根证书验证缺陷

- 是否验证根证书是否可信，缺陷包括：接受自签名证书、信任系统预制的的所有CA的证书



# 根证书验证缺陷-例子1

荷兰的CA-DigiNotar, 被攻破以致破产

- 2011年6月20日, COO: “We believe that DigiNotar's certificates are among the most reliable in the field.”
- 2011年7月10日, 发布了一个针对\*.google.com的签名, 被伊朗多个ISP使用
- DigiNotar 承认多个虚假签名: 包括Yahoo!, Mozilla, Wordpress, and Tor
- DigiNotar发现被入侵, 但是没有通知浏览器厂商
- Microsoft, Mozilla, Google, Apple and Opera将DigiNotar从浏览器根证书中移除

# 根证书验证缺陷-例子2

REEBUF

关注黑客与极客

首页

分类阅读 ▾

活动 ▾

作者 <sup>NEW</sup>

公开课

商城

## 国内 iCloud 服务器遭遇中间人攻击，中国苹果用户隐私不保

佐拉  2014-10-19  +17 共24179人围观，发现23个不明物体 资讯

联通对 iCloud 服务器进行 SSL 劫持？



# 根证书验证缺陷-例子2

## iCloud服务器遭遇SSL中间人劫持

- 被劫持iCloud服务器IP: <https://23.59.94.46/>
- 使用苏州的IP访问，得到自行签名的iCloud服务器证书
- 使用台湾的IP访问，得到真正的经过VeriSign CA签名的iCloud证书



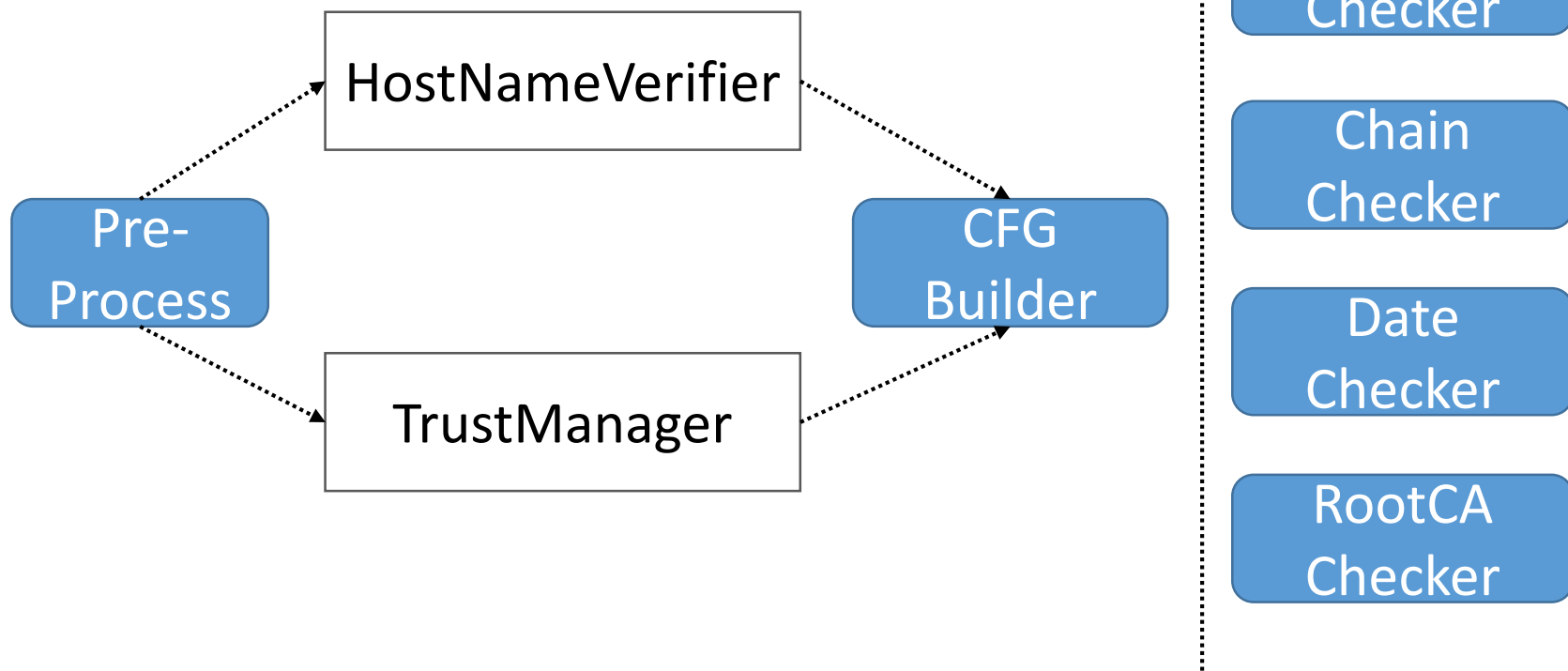


# HTTPS缺陷检测

---

# 检测方法

- HTTPS在安卓中的验证接口
  - HostNameVerifier: 检测是否连接正确的服务器
  - TrustManager: 检测服务器是否可信



# 服务器名称检查 (Name Checker)

- 目标

- 检查证书的CN属性和服务器域名是否匹配
- 服务器名称的验证有HostNameVerifier完成

```
public class Java7HostnameVerifier implements HostnameVerifier {  
  
    @Override  
    public boolean verify(String hostname, SSLSession session) {
```

- 难点

- 域名的匹配过程为字符串或者正则表达式的匹配，因此不存在固定形式的程序模式

- 方法

- 验证完成后返回验证结果 (T/F) 是否固定为T或者固定为F
- 遍历verify函数中的路径，计算每一条路径的返回值

# 证书链验证检查 (Chain Checker)

- 目标：检查TrustManager中是否使用父证书验证子证书的签名

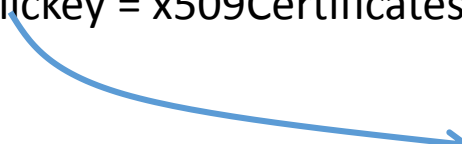
```
if (configuration.isVerifyChainEnabled()) {
    // Working down the chain, for every certificate in the chain,
    // verify that the subject of the certificate is the issuer of the
    // next certificate in the chain.
    Principal principalLast = null;
    for (int i = nSize - 1; i >= 0 ; i--) {
        X509Certificate x509certificate = x509Certificates[i];
        Principal principalIssuer = x509certificate.getIssuerDN();
        Principal principalSubject = x509certificate.getSubjectDN();
        if (principalLast != null) {
            if (principalIssuer.equals(principalLast)) {
                try {
                    PublicKey publicKey =
                        x509Certificates[i + 1].getPublicKey();
                    x509Certificates[i].verify(publicKey);
                }
                catch (GeneralSecurityException generalsecurityexception) {
                    throw new CertificateException(
                        "signature verification failed of " + peerIdentities);
                }
            }
            else {
                throw new CertificateException(
                    "subject/issuer verification failed of " + peerIdentities);
            }
        }
        principalLast = principalSubject;
    }
}
```

# 证书链验证检查 (Chain Checker)

- 方法

- 抽取代码中、对于证书对象操作的循环代码片段
- 识别父证书和子证书
- 在片段中检查是否会调用到verify函数进行证书链验证

```
PublicKey publickey = x509Certificates[i+1].getPublicKey();
```



```
x509Certificates[i].verify(publickey)
```

- 检查代码行为中验证失败是否有异常抛出

```
throw new CertificateException(  
    "signature verification failed of " + peerIdentities);
```

# 证书日期检查 (Date Checker)

- 目标

- 检测TrustManager是否对证书的有效期进行了验证
- 正确的例子

```
if (configuration.isExpiredCertificatesCheckEnabled()) {  
    // For every certificate in the chain, verify that the certificate  
    // is valid at the current time.  
    Date date = new Date();  
    for (int i = 0; i < nSize; i++) {  
        try {  
            x509Certificates[i].checkValidity(date);  
        }  
        catch (GeneralSecurityException generalsecurityexception) {  
            throw new CertificateException("invalid date of " + server);  
        }  
    }  
}
```

- 方法

- 抽取代码中存在的循环，检查是否在每一次循环迭代中进行了有效期检测
- 检查是否存在抛出异常的情况

# 根证书检查 (RootCA Checker)

- 目标
  - 检测TrustManager信任的根证书的范围
- 默认情况下，信任系统中所有预置的CA证书
- 程序实践
  - 验证过程的可信证书存储在KeyStore的类实例中

```
try {  
    trustStore = KeyStore.getInstance(configuration.getTruststoreType());  
    in = new FileInputStream(configuration.getTruststorePath());  
    trustStore.load(in, configuration.getTruststorePassword().toCharArray());  
}
```

- 判断自签名证书

```
try {  
    trusted = trustStore.getCertificateAlias(x509Certificates[nSize - 1]) != null;  
    if (!trusted && nSize == 1 && configuration.isSelfSignedCertificateEnabled())  
    {  
        —  
    }  
}
```

# 根证书检查 (RootCA Checker)

- 方法

- 获取代码中KeyStore的加载行为，识别程序是否使用KeyStore的关键API筛选可信的CA证书
- 检测是否使用KeyStore对根证书进行判断

```
trusted = trustStore.getCertificateAlias(x509Certificates[nSize - 1]) != null;
```

- 对于判定失败后是否有异常的抛出

```
if (!trusted) {  
    throw new CertificateException("root certificate not trusted of " + peerIdentities);  
}
```



# HTTPS安全性增强

---

# 概览

- 服务器证书的身份验证
  - 基于CA架构
  - Convergence
  - DANE (DNS-based Authentication of Name Entities)
  - Key Pinning
- CertShim
  - 在不修改客户端的情况下，增强SSL安全性
  - 灵活、通用安全性增强框架

# 方法一：Convergence

- 目标：防止网络提供者通过替换服务器证书进行MITM攻击
- 方法
  - 当使用SSL连接时，客户端询问其他多个节点连接该服务器返回的证书
  - 假设：攻击者很难对多个网络连接进行中间人攻击
- 问题：
  - Service-side MITM Attack

## 方法二：DANE

- 在DNS服务商处登记此域名有效的服务器端证书的属性
- 例子
  - Alice拥有example.com域名，并且希望所有的客户只接受使用 Charlie这个根证书签名的证书
  - Alice需要在DNS处设置一个TLSA记录
  - \_443.\_tcp.example.com
    - Usage: CA constraint
    - Selector/Matching: SHA-1 digest
    - Certificate for Association: SHA-1 digest of Charlie's certificate
- 问题
  - 部署缓慢

# 方法三：Key Pinning

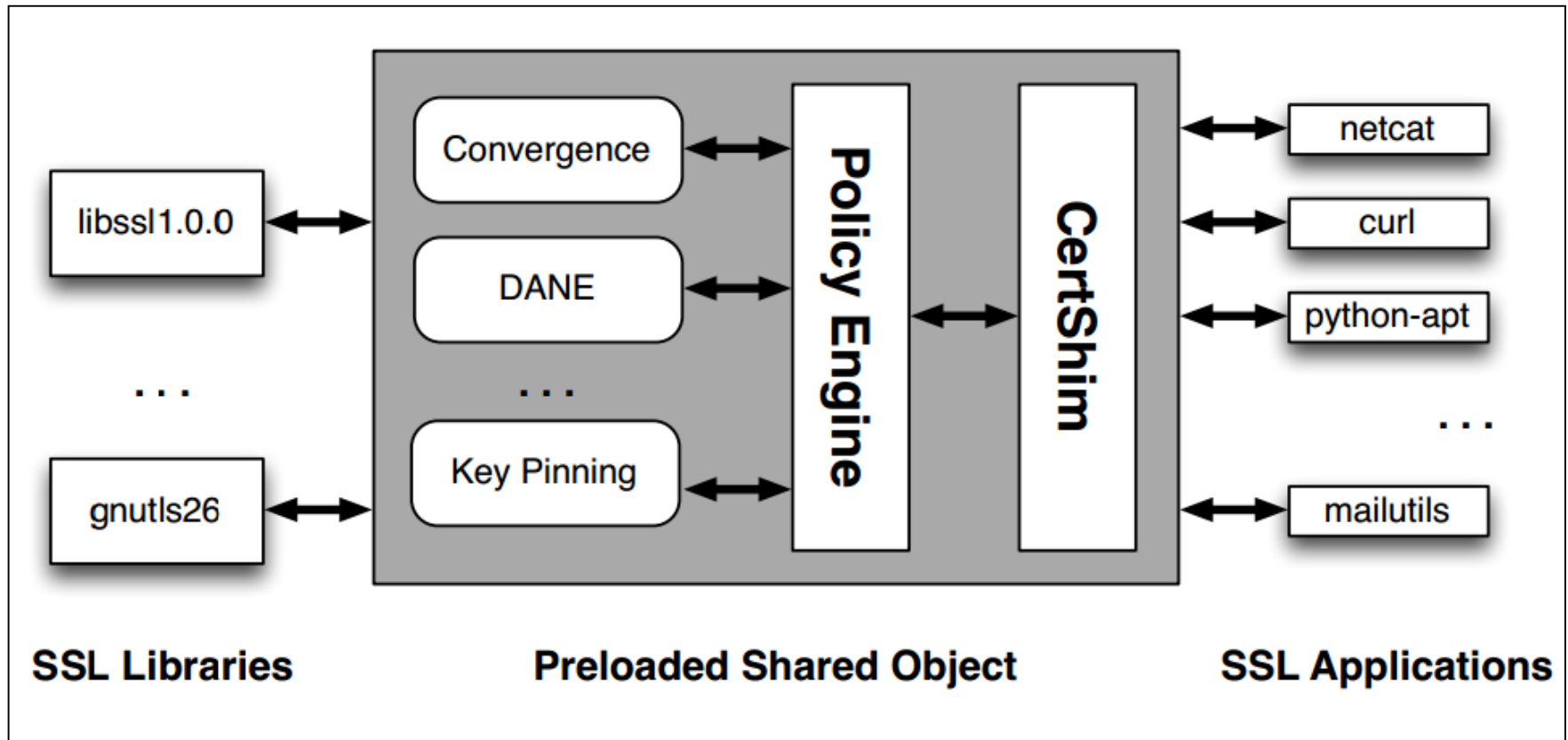
- 方法

- 客户端程序在发布时就预先保存好服务器端的证书
- 在客户端运行时将服务器实际返回的证书同预先保存的证书进行比较

- 问题

- 客户端Key Pinning无法适应服务器端的Key更改
- 无法识别服务器端的Key更改是否恶意

# CertShim



# CertShim-钩子函数

- 验证函数
  - `ssl_get_verify_results()` [OpenSSL]
  - `gnutls_certificate_verify_peer()` [GnuTLS]
- 握手函数
  - `ssl_connect()` [OpenSSL]
  - `ssl_do_handshake()` [OpenSSL]
  - `gnutls_do_handshake()` [GnuTLS]
- 网络上下文函数
  - `getaddrinfo()`, `gethostbyname`, `connect()`

# CertShim-规则

- 规则样例
  - global\_policy
  - command\_policies
  - host\_policies
- 验证方法
  - cert\_authority
  - cert\_pinning
  - convergence
  - dane
  - *vote*

```
1 global_policy: {
2     cert_pinning    = false;
3     cert_authority  = true;
4     convergence     = false;
5     dane            = false;
6     vote            = 1.00;
7 };
8
9 command_policies: ({
10     cmd = "/usr/bin/git";
11     vote = 1.00;
12     methods: {
13         cert_authority = false;
14         convergence    = true;
15     };
16 }, {
17     cmd = "/usr/bin/lynx";
18     vote = 0.50;
19     methods: {
20         cert_pinning    = true;
21         convergence     = true;
22         dane            = true;
23     };
24 });
25
26 host_policies: ({
27     host = "www.torproject.org";
28     vote = 1.00;
29     methods: {
30         cert_authority = false;
31         dane = true;
32     };
33 });
```



# CertShim-规则

- CA不可信情况下的规则

```
cert_pinning = true;  
convergence = true;  
vote = 0.50;
```

- 防范服务端的中间人攻击

```
convergence = true;  
dane = true;  
vote = 1.00;
```

# 报告总结

- HTTPS缺陷类型
- HTTPS缺陷检测
  - 采用静态分析进行代码检测
- HTTPS安全性增强
  - 基于规则，Hook网络连接库进行证书验证管理

谢谢！

