

信安复习笔记

Basic of information security

- 策略定义安全，机制实现策略
- 信息三要素
 - **Confidentiality** 保密性-对信息和资源的隐藏
 - 访问控制机制
 - **Integrity** 完整性-数据或资源的可信度
 - 预防机制和检测机制
 - 数据完整性和来源完整性
 - **Availability** 可用性-能够访问数据和数据源
- Trust and knowing assumptions(安全以此为基础)（用假设判断策略是否正确描述了所需要的安全等级）
 - 需考虑安全各个方面
 - 策略的公理（制定策略需要的假设）
 - 正确地获取并满足安全需求
 - 正确地区分“安全”“非安全”的状态
 - 安全策略可由安全机制实施
 - 机制
 - 假设机制是执行策略的；
 - 多种机制的并集实现了安全策略的所有规定
 - 机制都被正确实现并被安装和管理
 - 机制依赖基础设施的支持
 - secure precise partial 三种机制
- **assurance** 安全保障
 - Assurance: 测量多大程度上可以信任系统会做其应做的事情。
 - 规范（specification）
 - 系统的目的被规范定义
 - 对预期功能的声明，并非对规范本身
 - 设计（design）
 - 系统如何满足规范
 - 实现（impomentation）
 - 创建符合该设计的系统
- 运作问题
 - 开销分析
 - 预防和恢复
 - 风险评估
 - 我们需要保护某一件东西吗？
 - 花多大力气去保护？
 - 法律与人的习惯
 - 这种行为合法吗？法律会支持这种行为吗？
- The human factor
 - 技术30%，管理70%
 - 组织问题
 - 能力和责任

- 经济受益
- 人的问题（安全系统的核心是人）
 - 外部人员和内部人员（）后者威胁更大
 - 未经训练的人员
 - 利用人性弱点的社会工程学技术
- 信息安全的目标
 - 预防
 - 发现
 - 修复
- 一些常见攻击
 - **Passive attacks:** 监听
 - **Active attacks:** 修改、延迟、重放、拒绝

DOS攻击：发送pin数据包，发送地址填A，接收地址是局域网上的广播地址，然后局域网中所有主机都会向A回复引起宕机

2.1 history

- 早期的密码都是怎么被攻破的？
 - 无法通过Frequency Analysis.频率分析
 - 一些英文单词的出现频率要大于其他单词
- 凯撒密码(3位)
- 维吉尼亚方阵
 - 传统的频率分析无用了
 - 密钥不断重复，密钥是第几个就找到R行的代替
 - 混合使用offset和lineorder，一横一竖（KRYPTOS PALIMPSEST）
- 维吉尼亚方阵破译
 - 重复关键字会导致密文中的重复。
 - 然后通过找出密文中重复间隔的因数来确定关键字的长度。
 - Example: The sequence T-H-J-D repeats after 18 spaces. The numbers 1, 2, 3, 6, and 9 are all factors of 18. Therefore, possibly: – The key is 1 letter long and repeats 18 times. (Can be discounted) – The key is 2 letters long and repeats 9 times. – The key is 3 letters long and repeats 6 times. – The key is 6 letters long and repeats 3 times. – The key is 9 letters long and repeats 2 times.
 - 通过检测过个重复元素寻找可能长度，然后通过设长度，频率分析解答（每一个都是凯撒偏移量密码）
- 一次性密码本
 - Works when one unique key is used once to encipher/decipher one message.
- 密钥分配问题

- 为何一次性密码本没有被推广？
- One time keys must be at least as long as the messages they encipher. – Keys must be communicated between parties outside of the enciphered channel. – Keys can only be used once.
- 首先手上要有一本一次性密码本用以加密文件，接着将一次性密码本里的字母，与被加密文件的字母给依序按某个事先约定的规定一一相混，其中一个相混的作法是将字母指定数字(如在英语中，将A至Z依序指定为0至25)然后将一次性密码文本上的字母所代表的数字和被加密文件上相对应的数字给相加，再除以该语言的字母数，假设是n(如英语为26)，若就此得出来的某个数字小于零，则将该小于零的数给加上n，如此便完成加密。
- 举个例子，若要加密讯息“This is an example”，而用以加密的一次性密码本如下所示： MASKL NSFLD FKJPQ 则利用指定数字的方法，可分别将两者给做以下的转换： This is an example → 19 7 8 18 8 18 0 13 4 23 0 12 15 11 4 MASKL NSFLD FKJPQ → 12 0 18 10 11 13 18 5 11 3 5 10 9 15 16 两者依序相加后得到的讯息如下： 31 7 26 28 19 31 18 18 15 26 5 22 24 26 20 将以上得到的讯息模26后可得： 5 7 0 2 7 5 8 8 11 0 5 22 24 0 20 它也就变成了 FHACHFIILAFWYAU 而若要解密以上信息，反向操作即可。
- Beale Papers
- 第二纪元
- 恩尼格马
 - $26 * 26 * 26 = 17,576$ possible cipher alphabets
 - $17576 * 6 = 105456$ Pressing the ‘a’ key would cause a circuit to be created through the rotors and illuminate the letter ‘B’ on the lamp board. ‘B’ is the ciphertext. The first rotor would then click forward one position. The rotors advanced as keys were pressed. Once rotor one completed a complete revolution of 26 characters, rotor two would click forward once, and so on.
 - 100,391,791,500 possible plug board settings (怎么算的) ?
 - 3 rotors 17,576 6 rotor orientations 6 6 plug board settings 100,391,791,500
 - <http://www.codesandciphers.org.uk/enigma/steckercount.htm>
- 恩尼格码破解

2.2 brief intro to cryptography

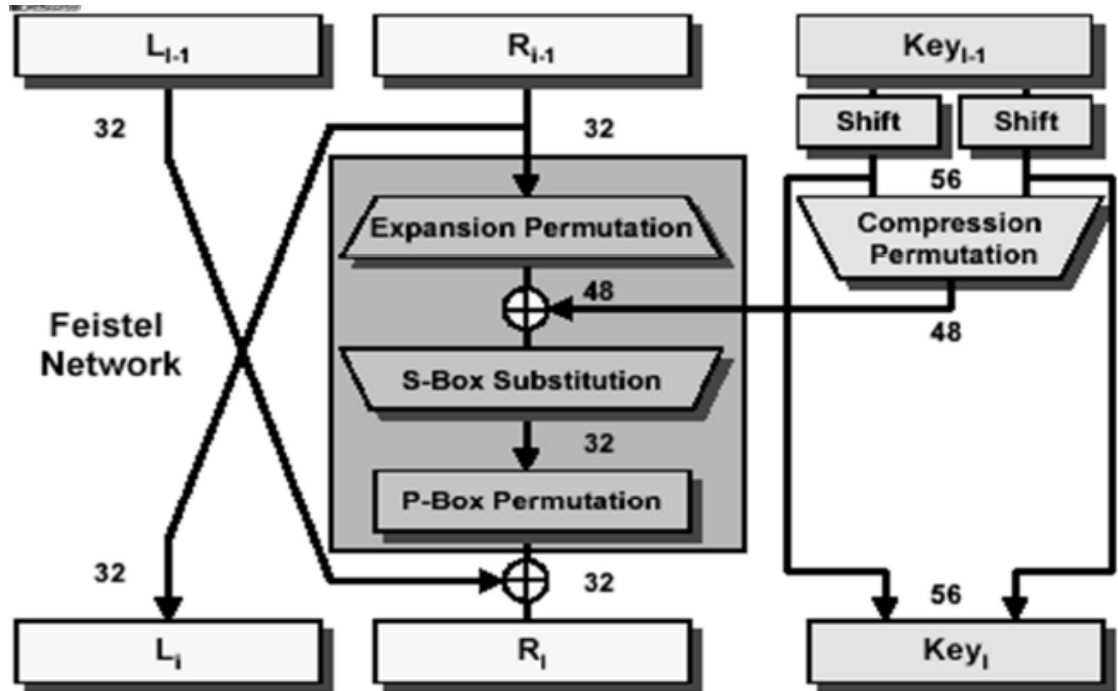
- 术语
 - 加密过程
 - 解密过程
 - 符号
 - P Plaintext
 - C Ciphertext
 - E Encryption function
 - D Decryption function
 - $E(P) = C$ encrypting plaintext yields ciphertext

- $D(C) = P$ decrypting ciphertext yields plaintext
- Restrict Algorithm
 - 算法需要保密
- Key-Based Alogrithm
 - 秘钥需要保密
- 攻击攻击攻击攻击攻击
 - 只获得密文
 - 知道一些密文和明文的对应
 - 攻击者可以将任何明文加密（破解）
 - 真正的攻击一般不会破坏加密性（算法）本身；
 - 如猜秘钥攻击
- 加密算法种类
 - Secert-key 秘密秘钥（对称）
 - public-key 公钥（非对称）
 - 数字签名和哈希算法
- 对称加密
 - 同一把秘钥——两端加密解密
 - 两个算法——加密算法和解密算法
 - 秘钥必须事先分发
- 堆成加密要点
 - 防窃听
 - 安全频道进行交易
 - 安全存储
 - 记住密码
 - 认证
 - challenge /response
 - 完整性监察
 - 消息校验和 - 加密校验和
- substitution Cipher
 - Modern substitution ciphers take in N bits and substitute N bits using lookup table: called S-Boxes
- Transposition cipher(转置)
 - 文本横着插进去，密文竖着放出来（P-BOX）
- Block Cipher
 - n bit n bit加密，不考虑块与块之间的影响

- In a good block cipher, each output bit is a function of all n input bits and all k key bits

DES

- 加密：用KA，将文件编码为64bit 的chunk
- key 56bit + 8bit parity
- 解密：再用一次KA，以密文为输入，除了key是倒序的；
- 64 plain \rightarrow 16 iteration (with 56bit key) \rightarrow 32 bit swap \rightarrow inverse trans \rightarrow output
- iteration: $L(i-1)$ 异或 $f(R(i-1), K_i)$ 为新的 R_i ，原 R_i 为 L_i



- 攻破原因：Key is only 56bits, 2 exp 56 72,057,584,037,927,936– Computing capability is increasing exponentially– Parallel attack – exhaustively search key space

Beyond DES

- Triple DES put the output of DES back as input into DES again with a different key, loop again: $3 \times 56 = 168$ bit key
- AES shall be designed so that the key length may be increased as needed. block size $n = 128$ bits, key size $k = 128, 192, 256$ bits
- MARS, twofish, RC6, Serpent, Rijndael – Winner! (Rijndael)
- **ECBECBECBECB**
- 针对块的两种攻击方式
 - ciphertext only attacks, 利用现有密文
 - build a codebook of $\langle C_k, \text{guessed } P_k \rangle$ pairs (chosen plaintext attacks). Replay Attacks?
 - 阻止方式：Inhibits replay attacks and codebook building: identical input plaintext $P_i = P_k$ won't result in same output code due to memory-based chaining. IV = Initialization Vector use only once
- 流加密
 - Rather than divide bit stream into discrete blocks, as block ciphers do, XOR each bit of your plaintext continuous stream with a bit from a **pseudo-random sequence** key \rightarrow PRS PLAIN XOR PRS = Cipher • At receiver, use same symmetric key, XOR again to extract plaintext

- 上一个加密的s1成为新的key，然后变成s2，以此类推
 - 同样有sender IV 和 receiver IV
- 非对称加密
 - 对称加密的问题：若未曾谋面，如何递交密钥？且记忆大量密钥成本太高了；
 - SINGLE MOST IMPORTANT
- 非对称加密机制
 - plain->bob公开的公钥和算法->bob自己的秘密私钥解密->plain
 - 私钥不可通过公钥获得
 - 公钥直接公开就行
- 具体实现（RSA）
 - 公钥：(e, n)
 - 加密： $c = m^e \bmod n$
 - 私钥：d
 - 解密： $c^d \bmod n$
 - e, n, d选择：
 - chooses 2 large primes (each at least 100 digits): p, q multiplies p and q: $n = p * q$ – finds out two numbers e & d such that $e * d = 1 \pmod{(p-1)(q-1)}$
 - 一些说明
 - 信息m的大小介于[1, n];
 - 加密长信息：hybrid cryptosystem
- 为什么RSA安全？
 - 破解策略：偷到私钥
 - 知道原理后暴搜（时间长）
 - 通过原理计算d problem: Given two numbers (r,s), the algorithm outputs a number x such that $r * x = 1 \pmod s$.
 - 时间开销主要在 $n = p * q$ 分解上
 - 但是乘起来容易分解难！
 - longer than 155 decimal digits 更安全
 - 更为好的方法：伪造一个自己的钥匙对，发消息欺骗alice换公钥了！
 - 使用数字签名防止
- 私钥算法和公钥算法的优劣
 - 公钥算法慢，代价昂贵， VLSI chip难以获得或昂贵
 - 私钥中密钥分配很成问题
- 结合
 - 用公钥算法分发钥匙，用私钥算法加密信息
- 数字签名的必要性
 - 用私钥加密，用公钥解密(Bob->Chthy)
 - $s = md \bmod n$
 - $t = se \bmod n$
 - (e,n)
 - [0,n]
- 很长文本签名用哈希
 - 先hash数据，然后为哈希后的数据签名
- 单向哈希-MD5（128），SHA-1（160）
 - 进去长度不定，出来一定是这个长度
- 好的单向哈希算法

- 易于分析（计算）（amy document很快出结果）
- 难以逆运算（单向）
- 难以发现冲突（两个不同的数据几乎不可能有相同哈希值）
- MD5, SHS,HAVAL
- 数字签名？
 - 不可伪造：（unforgeble）
 - 签字人不可否认
 - 可被广泛验证
 - doc之间不同
 - 易于实现
- 重要签名技术
 - RSV, Schnorr。 DSS，椭圆曲线
- MD5被破解

3.1 Authentication 认证

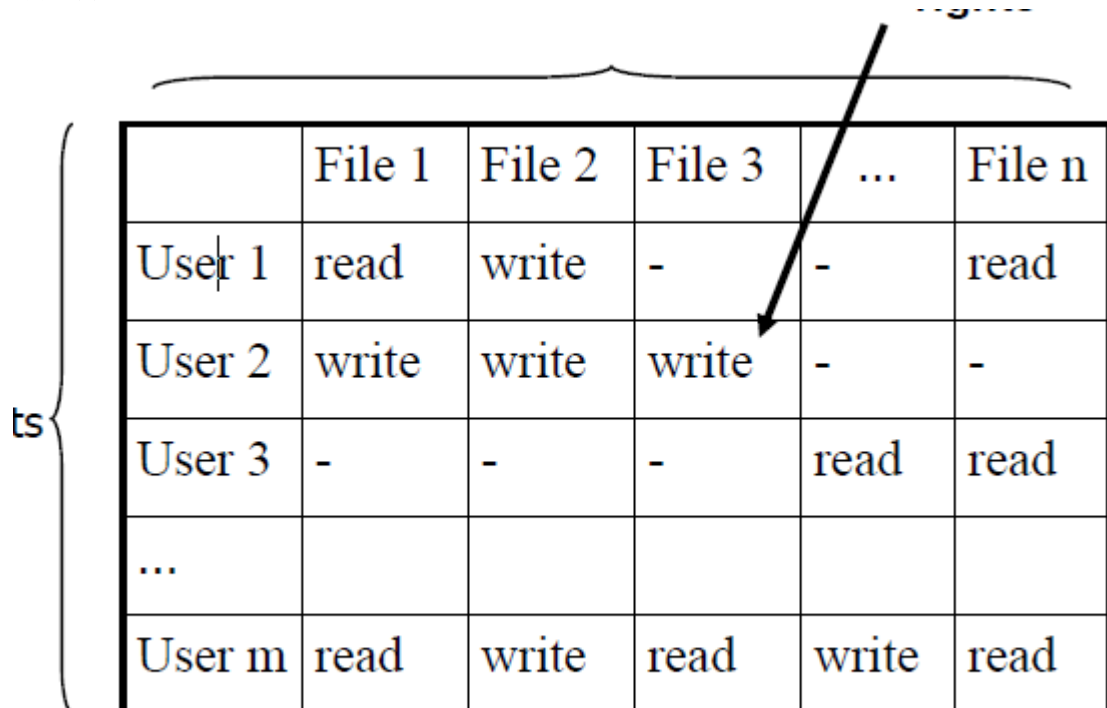
- 基本问题
 - 如何证明你是你？
- 多种方式
 - 密码密钥
 - 哪里：ip
 - 物理和行为
 - 安全标记
- 基于密码的
 - 容易被窃听
 - 密码文件不容易保守秘密
 - 猜出一个简单密码很容易
 - 系统如何检测密码？
 - Unix 策略——哈希
 - 存储密码的哈希（文件危险问题解决）
 - 用户输入密码后先转换成哈希在比对
 - 单向哈希
 - 传统unix策略：使用des以密码为key加密null串
 - password -> 8 char
 - run 25 times
 - 问题：密码并非完全随机
 - 可穷举
 - 人们倾向于用有意义文本做密码
 - 传统字典攻击方式
 - 找到密码文件，找到各种userID
 - 计算字典中所有word的哈希一个一个试，因为字典往往不大
 - 解决策略1：salting
 - 密码第一次设定时随机选择，接在密码前面一起哈希
 - 用户拥有相同密码可能有不同的密码文件
 - 但是字典攻击仍然存在
 - 优点：防止预先建立字典哈希表一一对照。攻击者必须对每个密码都验证一遍dict
 - 影子密码（shadow pass）

- 密码哈希只能被系统管理员读取;
 - 添加expiration date (到期日期)
 - 早先的实现 (Linux) 叫做login program, 有缓冲区溢出问题
 - 其他策略
 - 生物识别 (语音), 图形密码, 人脸识别等
- 基于生物识别的
 - 指纹, 声音, 人脸, 键盘计时
 - 优点
 - 不会被披露、丢失、遗忘;
 - 缺点
 - 花销, 安装, 保持
 - 对算法的要求
 - 判断出错? 是的不让进, 不是却进去(fraud rate insult rate)
 - 隐私?
 - 被伪造后无法撤销
 - 手写签字
 - 生物识别错误率
 - fraud rate (不是却进去) 和 insult rate (是的不让进)
 - 提升容忍域会导致前者偏高
 - F 和 I 一般成反比 (相乘为常数)
 - 依据用途确定阈值
 - 其他
 - 人脸识别 (20% error)
 - 指纹 (16点匹配, $F < 1e-3, I < 1e-1$)
 - 虹膜扫描
 - Irises are very random, but stable through life • Different between the two eyes of the same individual – 256-byte iris code based on concentric rings between the pupil and the outside of the iris – Equal error rate better than $< 0.0001\%$ – Best biometric mechanism currently known
 - hand geometry
 - 声音, 耳形, 静脉pattern, 脸部温度
 - 风险
 - 犯罪分子给出指纹次序错误
 - 可以通过记录攻破
- 基于认证协议的
 - 流程
 - bob让alice证明自己——Protocol ap1.0 : alice说我是alice
 - bob看不到alice, 所以攻击者可以声称自己是alice
 - 这时候试着加上ip? prot.2. 我是alice, 你看我的ip

- 没啥用，ip可以伪造
- prot3: 我是alice，这是接头暗号（secret password）
- palyback attack: 先记住你的暗号然后再伪造！
- prot4: 我是alice，这是加密的接头暗号
- 没用！
- 目标
 - 避免palyback攻击
 - number(R) used only once in a lifetime
 - prot5: to prove Alice "live", Bob sends Alice nonce R, Alice must return R, encrypted with shared secret key
 - 弊端：没有避免服务器数据库读取
 - 使用nonce，但用非对称加密

3.2 授权

- 目标
 - 你可以做什么？
 - 如何控制权限？
- 访问控制
 - 假设
 - 系统知道user是谁（已经经过认证）
 - 访问请求需经过gatekeeper，不可被绕过（bypass）
- 访问控制矩阵

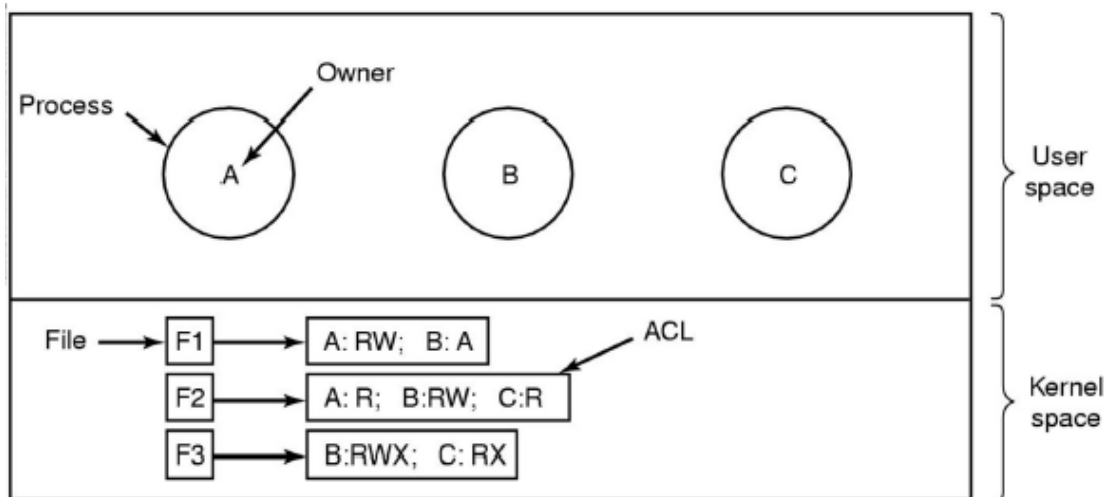


The diagram shows an Access Control Matrix (ACM) with a large curly brace on the left labeled 'ts' (subjects). The matrix has columns for 'File 1', 'File 2', 'File 3', '...', and 'File n'. The rows represent different users: 'User 1', 'User 2', 'User 3', '...', and 'User m'. Each cell in the matrix contains a permission value: 'read', 'write', or '-'. An arrow points from the 'File 3' column header to the cell containing 'write' for 'User 2'.

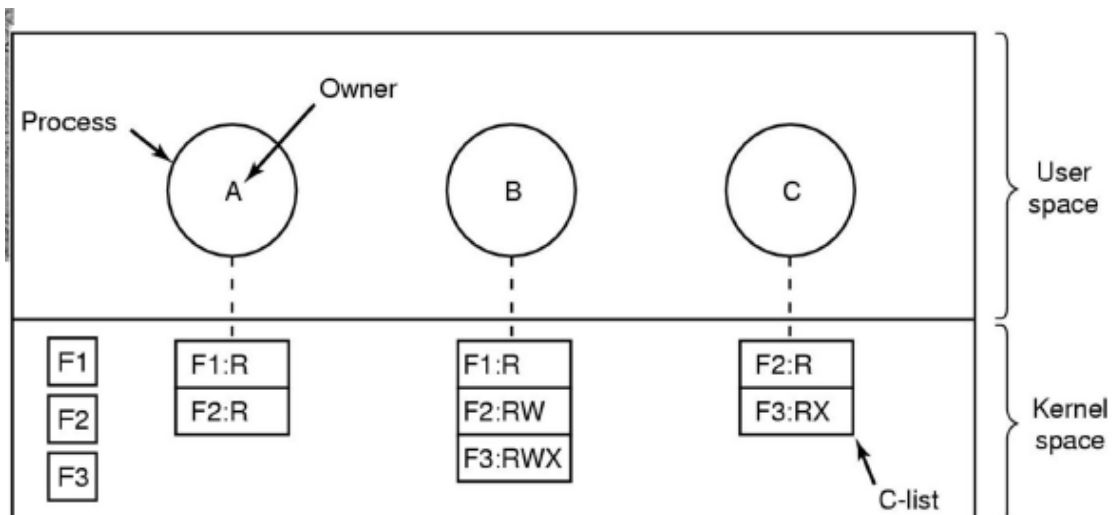
	File 1	File 2	File 3	...	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
...					
User m	read	write	read	write	read

-
- 抽象概念
 - 主体（subject）
 - 对象（文件，etc.）
 - 权限（rights）
 - sub（row）->ob(col)
- sub和用户

- 注意 user 可能对应多个principal
- user有很多principal，但一个prin只有一个user
- 确保了责任制（accountability）
- 一个主体是一个代表着某一特定principal的程序/应用
- 一个principal可以是闲置的，也可以被多个主体代表
- principal和subject
 - 一般情况下（不总是！）每个主体只有一个principal
 - 一个prin的所有主体有相同的权限
- 对象（Object）
 - 种类
 - 文件，路径（文件夹），内存片区
 - sub也可以成为ob! kill subject（之类的）时候
- 访问控制列表vs功能
 - 如何实现？
 - ACL（访问控制列表）
 - 存储矩阵的列（user123.....）和资源一起，文件拥有用户
 - 功能
 - 用户为每个资源持有不可伪造的票证
 - ACL

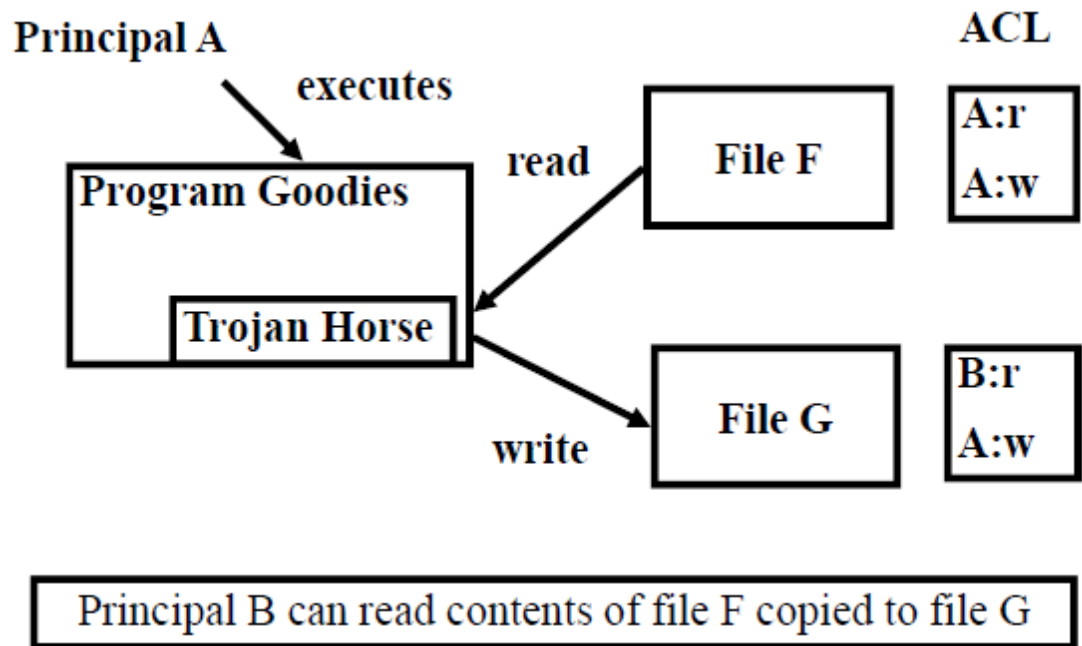


Capa



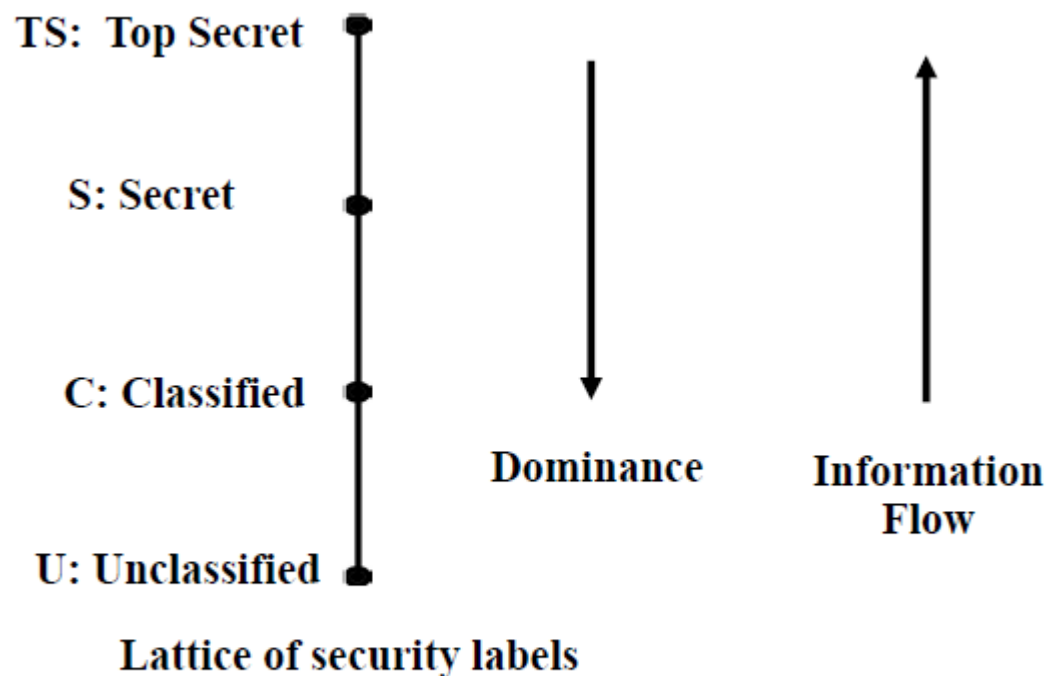
两者比较

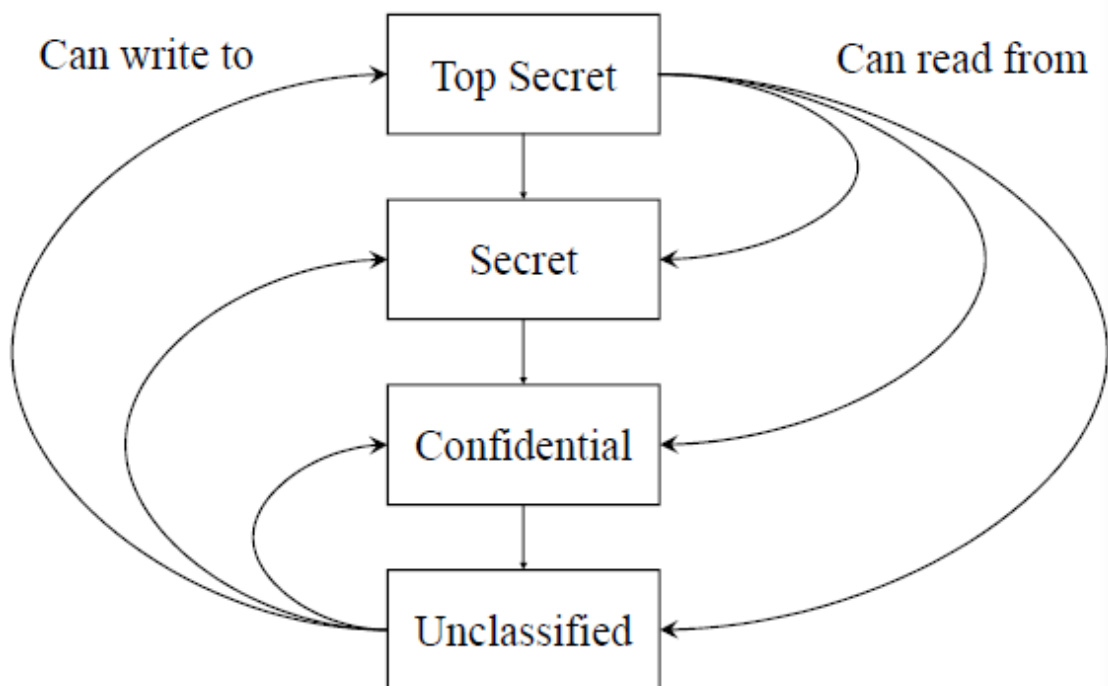
- ACL需要对subject认证
- capa不需要认证sub但是需要不可伪造性和对功能传播的控制
- ACL基于ob分发权限，capa基于sub分发权限
- 选择ACL?
 - 大部分操作系统用ACL保护文件；
 - 大部分操作系统使用acl的简略版，只有三个条目（类unix）：如Owner，group，other
- 以unix为例：
 - User：谁拥有这个文件
 - 和owner在一个group里的其他用户
 - 除此之外的其他用户
 - 文件操作权限
 - Read， write， Execute
 - 9bits owner group other
 - RWXRWXRWX
 - 111101100（754）
- 有一些操作系统还有别的条目，如给别的用户添加权限等
- Capa?
 - capa提供对subject的更细粒度的最小权限控制，尤其是针对特定任务创建动态的的即时subject
- 自主（DAC）与强制访问控制(MAC)
 - 概述
 - 前者允许访问权限在sub之间传递；主体对某一访问权限的拥有足以允许（其授权其他sub）访问该对象了。
 - 换句话说，我有了这个权限，我就能给别人
 - 后者将主体对对象的访问限制在安全标签的规定之内
 - DAC的固有缺陷
 - 无限制的DAC允许来自Ob的信息可以被读取到可以由主体写入的任何其他Ob
 - 这严重依赖对用户的信任，而且无法保证特洛伊木马不这样做
 - 特洛伊木马
 - 特洛伊木马是经过正式授权的用户安装的流氓软件，尽管该用户可能不知情。特洛伊木马程序会执行用户期望的操作，但还会利用用户的合法权限导致安全漏洞。



- **MACMACMACMACMAC**

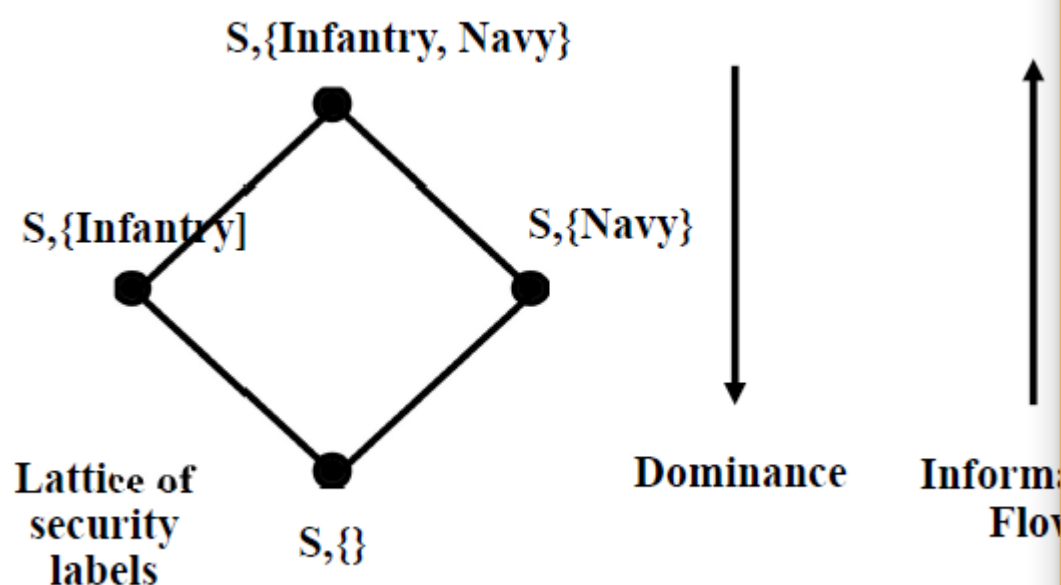
- 目标：防止未经授权的信息泄露
 - 专门解决潜藏的信息流；
- Bell-LaPadula Model 第一步
 - 线性划分安全等级
 - security clearance 和 security classification





-
- s可以读o, 当且仅当o的等级低于s且s有读o的权限 (MAC和DAC合用, NO_READ_up)
- s可以写o, 当且仅当o的等级高于s且s有写o的权限 (NO_WRITE_DOWN)
- Bell-LaPadula Model 第二步
 - 包含类别 (clearance, cate.set)

```
( Top Secret, { General Staff, Infantry, Navy } )
( Confidential, { General Staff, Infantry } )
( Secret, { General Staff, Navy } )
( Unclassified, { Navy } )
```



- 隐蔽通道
 - 有时并不是有意而为之
 - 木马之间的信息传递

- user是被信任的，但sub不总被信任，因为可能存在特洛伊木马；
- 资源耗尽通道
 - Given 5MB pool of dynamically allocated memory
 - High-Level Process bit = 1 □ request 5MB of memory bit = 0 □ request 0MB of memory
 - Low-Level Process request 5MB of memory if allocated then bit = 0 otherwise bit = 1
- 负载敏感通道
 - High-Level Process bit = 1 □ enter computation intensive loop bit = 0 □ go to sleep Low-Level Process perform a task with known computational requirements if completed quickly then bit = 0 otherwise bit = 1
- 对付通道
 - 关闭频道或减慢速度
 - 检测尝试使用该频道
 - 容忍它的存在
- 等级
 - D: 无安全需求
 - C: DAC
 - B, A: MAC
 - C1, C2

◆ C1

- Cooperating users at same level of sensitivity
- Access control; users can protect their own data
- Discretionary access control

◆ C2

- Finer granularity of control
- Better audit functions; each individual access to each object can be tracked

■

- B1, B2, B3, A1

- Beyond MAC/DAC
 - CDAC（取决于内容，比如说你只能看到50k一下的薪水）,RBAC（取决于角色user被授予角色，提高了效率但cannot specify fine-grained rules）,CBAC（基于上下文（事件发生环境，比如薪水只有在年底才能更改））

4.1 恶意代码

- 什么是恶意逻辑？
 - 导致安全策略被破坏的一系列指令集
 - unix的shell脚本
- 恶意代码/逻辑的种类
 - 特洛伊木马
 - 既有用户知情的目的也有用户不知情的

- 比如setuid shell (set user ID upon execution)
- 可复制特洛伊木马
 - 木马插入编译器及后继版本

Thompson[995]在 login 程序当中嵌入一种特洛伊木马。当用户登录时,木马会接受一个固定的口令以及用户的正常口令。但是,任何查看 login 程序源代码的人都会立刻发现这个木马。为了掩饰该木马,Thompson 让编译器检查当前正在编译的程序。如果是 login 程序,那么编译器会加入额外代码,以使用固定的口令。现在,login 程序中就不需要加入额外代码了。因此,检查 login 程序源代码的分析人员将看不出任何错误。如果分析人员用这段源代码进行编译,那么他会认为所得到的可执行程序是未受破坏的。

但所加入的这些额外代码在编译器的源码中可以被发现。为了消除这个问题,Thompson 修改了编译器。这个改进后的第二版本会检查编译器(事实上,就是 C 语言的预处理程序)是否正被重新编译。如果是,那么就插入一些修改编译器的代码,这些代码的目的是加入这个木马以及 login 程序的木马。他编译了编译器的第二版本并安装了可执行文件。然后把破坏了的源代码替换成老版本的编译器。至于 login 程序,检查源代码将看不出任何问题,但在编译和安装编译器时会插入两个特洛伊木马。

Thompson 尽力确保编译器的第二版本不被发布。该编译器在系统中保留了相当长一段时间,直到有人用另外一个系统中的新版本覆盖了它的可执行文件[839]。Thompson 的观点^①是:“任何源代码级的验证和审查都不能使用户免于使用不可信代码”。该要点在后面会不断重申。

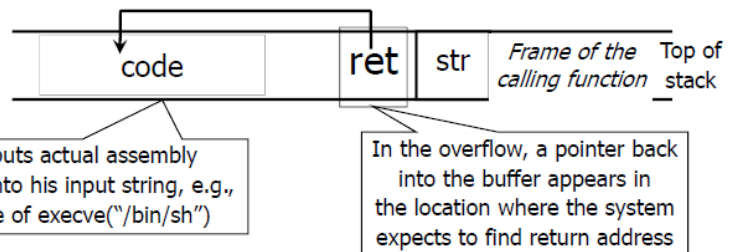
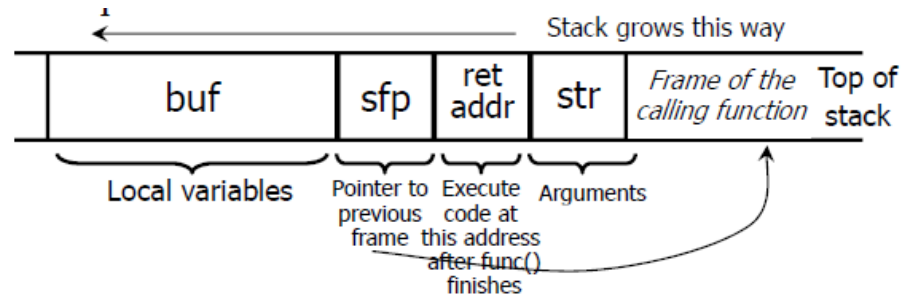
○ 电脑病毒或蠕虫

- 插入过程和执行过程
- 伪代码

```
beginvirus:
  if spread-condition then begin
    for some set of target files do begin
      if target is not infected then begin
        determine where to place virus instructions
        copy instructions from beginvirus to endvirus into
target
        alter target to execute added instructions
      end;
    end;
  end;
  perform some action(s)
  goto beginning of infected program
endvirus:
```

- 病毒种类
 - Boot sector infectors
 - 插入引导扇区或磁盘
 - 系统读盘时执行
 - Executable infectors
 - 感染可执行文件
 - 病毒只感染一部分
 - 方式
 - 破坏(易侦测)
 - 插入头(文件大小问题)
 - 插入头尾,病毒环绕
 - Multipartite viruses

- 以上两种都可以
- TSR viruses
 - app结束了还在内存中活跃 (Terminate and stay resident)
- Stealth viruses
 - 秘密感染文件
- Encrypted viruses
 - A virus that is enciphered except for a small deciphering routine
- Polymorphic viruses
 - 每次复制都会改变结构
- Macro viruses
 - 由一系列指令组成的病毒，它们被解释而不是直接执行
- 蠕虫
 - 和病毒的区别
 - 病毒感染其他程序来复制，一半插入code中
 - 蠕虫自我复制且可以独立运行
 - 组成
 - 邮件攻击
 - 手指攻击
 - 缓冲区溢出攻击



- 被信任远程主机攻击
- 字典密码攻击
- 过程
 - 选择攻击目标
 - 放钩子
 - 拉入蠕虫代码
 - 获取编译器命令
 - 继续下一层传播
 - 开始主动感染

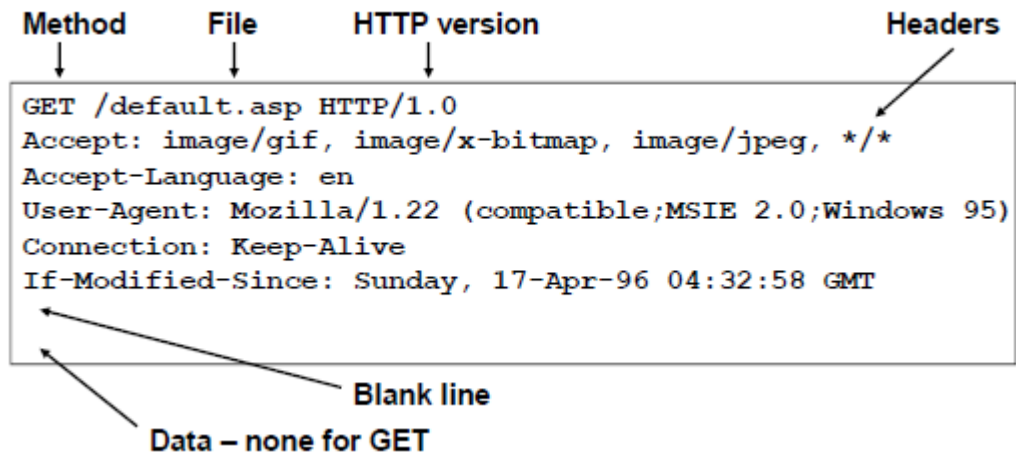
○ 其他

5 可信任操作系统

- 信任和安全
 - 信任是relative。specific的
- 设计
 - 原则
 - Economy of Mechanism
 - KISS principle (Keep it Simple (less than go wrong) , Silly)
 - Fail-Safe Defaults(故障安全默认值)
 - 根据权限进行基本访问决定而不是排除
 - ☐ Remove illegal characters:
 - illegal_chars = “;:/\!” str = [c from input if c not in illegal_chars] **Better:** legal_chars = “abcdefg...” str = [c from input if c in legal_chars]
 - SQL 注入
 - ' OR 1=1 --
 - Complete Mediation
 - Every access to every object must be checked for authority
 - Open Design 开源
 - The design should not be secret
 - The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords.
 - Separation of Privilege
 - 在可行的情况下，需要两个钥匙才能解锁的保护机制比仅允许访问演示者的钥匙更加健壮和灵活。
 - Require multiple conditions to grant privilege
 - Least Privilege
 - 特权够用就行
 - Least Common Mechanism
 - 减少程序间共享状态
 - Psychological Acceptability 界面友好简单
 - 用户友好，有助于用户接受保护机制
 - 保护机制不以影响用户体验为代价（访问资源）

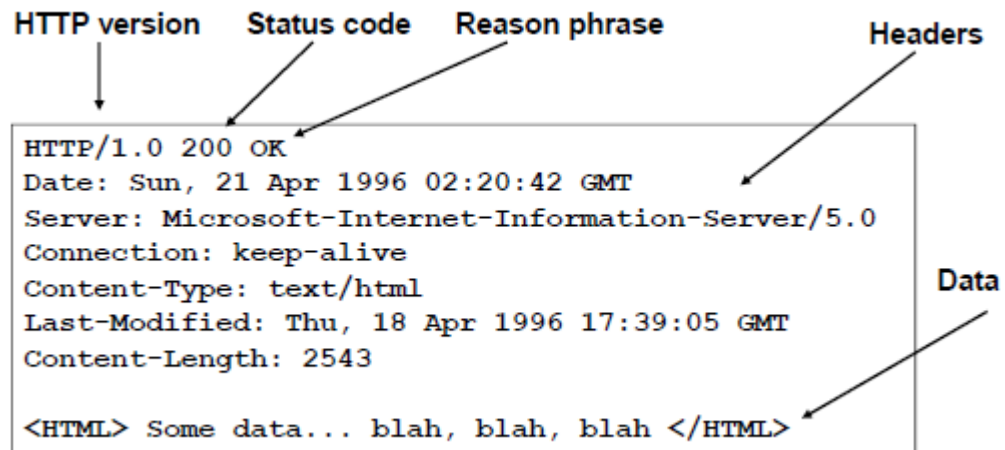
6.1 网络安全

- HTTP
 - 方法: GET, POST, HEAD
 - 无状态请求与响应协议
 - 请求相互独立
 - 无状态
 - 演变
 - HTTP 1.0
 - HTTP 1.1
 - req格式
 - method
 - file
 - version
 - headers
 -



- resp格式

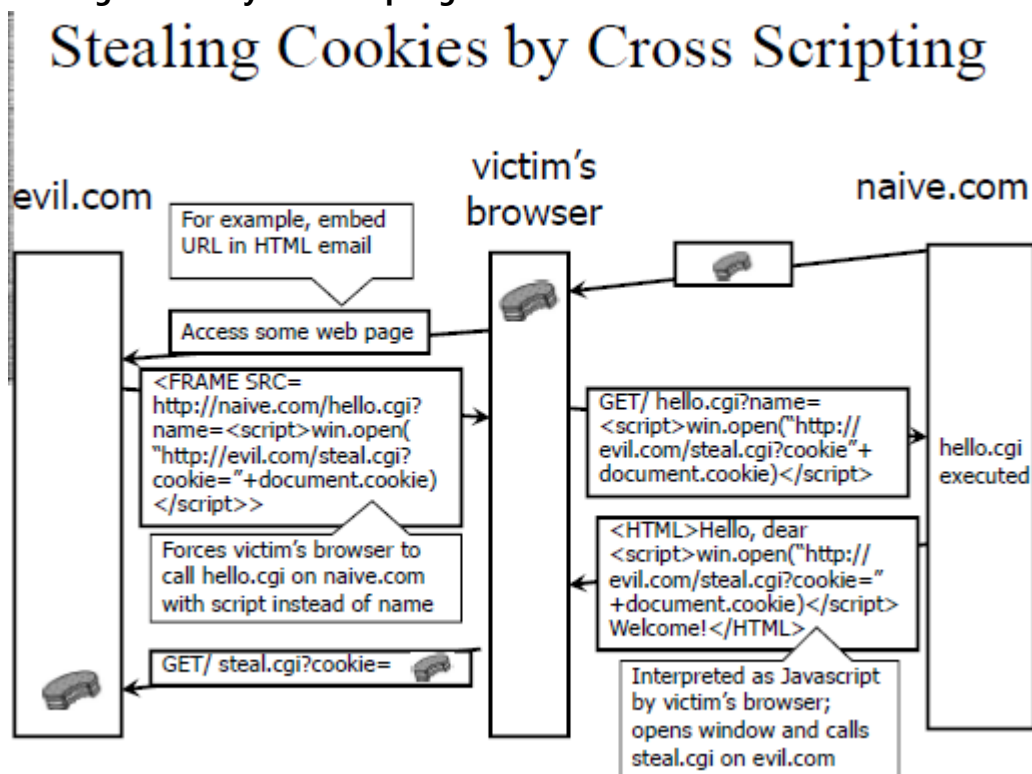
- 版本
- 状态码
- reason语句
- headers
- data



- http认证
- 客户端给服务器url和方法 (get/post)
- server返回401 (没有授权) fresh, realm, random nonce
- 客户端发送hash(usr, realm, pas) hash2(method, url) hash3(h1, server nonce, h2)
- 原始浏览器会话
 - 网站看参数 (php?pid=269)
 - 查找item
 - checkout
 - 特点: session在url中, 容易在网络中被读取
- 用户使用密码登录网站, 获得授权, 使用包含授权的url访问资源
 - 用户不需要再次获取授权
 - 但是这很容易猜出其他用户的序列号, 因为授权码是一个全局的序列
 - Fix——使用随机授权码
- 坏主意: 将状态编码到url中
 - 不稳定
 - 暴露在窃听下
 - url不私密

- cookie
 - 用途
 - Authentication
 - Personalization
 - Tracking
 - 暂时cookie, 永久cookie, 第三方cookie (来源或发向其他网站的cookie)
- 存储状态
 - 不能存在html中, 可以改
 - 存在cookie中
 - 对cookie设置mac
 - 改变价格?
 - 不存在服务器上 (地址欺诈, 存储问题)
- 认证码
 - 使用哈希
 - 使用crypt()
 - 更糟: 可以被利用提取秘钥
 - 更好的认证码
 - capability+expiration+hash(server secret, capa, expir)
- javascript 安全
 - 运行在沙箱中
 - 同源策略 (只能从相同服务器, 协议或接口中读取数据)
 - 为脚本设计权限

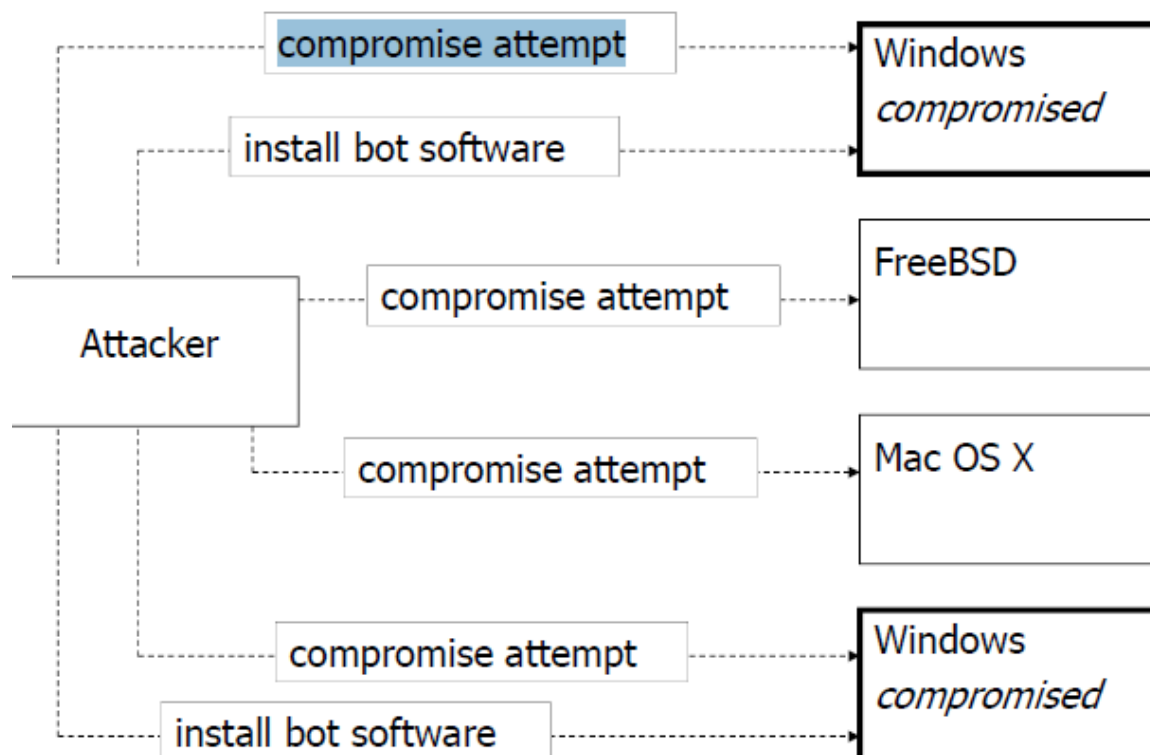
○ **stealing cookies by cross scripting**



-
- 预防: Preprocess any input from user before using it inside HTML
- URL重定向
- SQL注入 '1 = 1 -- " exec(cmdshellxxx)
 - 未初始化的输入

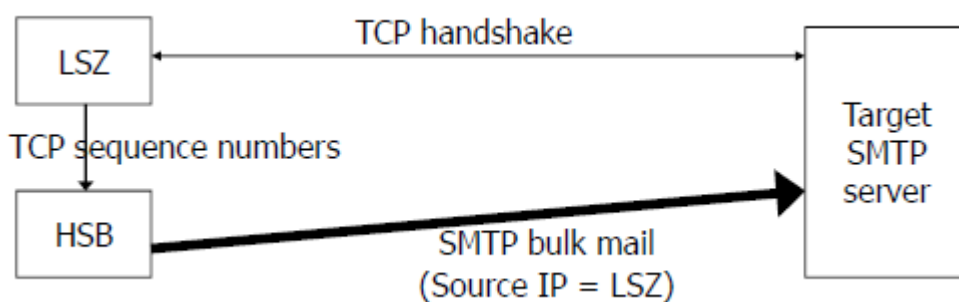
7.2 僵尸网络，垃圾邮件，拒绝服务攻击

- 僵尸网络
 - 概念
 - 能够按照指示行事的自治程序网络
 - 远程控制
 - 控制方式：IRC，P2P
 - 用途
 - DDoS
 - 垃圾邮件、点击欺诈
 - 为蠕虫提供温床



- 感染过程
 - 插入shellcode
 - sehllcode下载安装actuall bot
 - bot关闭防火墙和杀毒软件
 - 定位IRC Server， 连接加入通道
 - 重设认证模式防止被其他机器再度利用；
- 繁殖
 - （每个肉鸡）扫描新受害者的ip
 - 这一过程是自动的
 -
 - 主动botnet管理
 - Detect non-responding bots, identify "superbots"
 - Evidence of botnet-on-botnet warfare
- 拒绝服务攻击
 - 利用现有的网络协议
 - Smurf
 - Ping of Death

- SYN flood
 - UDP flood
- 分布式拒绝服务
 - 建立僵尸网络
 - 命令僵尸对受害者进行同时一致攻击
 - 受害者被数据包冲崩溃
- 工具
 - Trin00
 - 扫描已知的缓冲区溢出
 - 使用远程shell安装攻击守护进程
 - 发送命令，使用明文密码获取权限
 - att->mas TCP mas->zombie UDP
 - 为了防止侦测，守护进程会在其他人连接到zom，且mas已经获取权限时发出警告；
 - Tribal Flood Newwork
 - Stacheldraht
 -
- 垃圾邮件
 - 邮件欺诈
 - 原因
 - 通过SMTP发送，没有认证机制
 - 邮件来源是被发送者设定的
 - 收件人只能看到直接发给他的那个peer的ip（途径很多peer）
 - 开放的中继
 - 存在善良中继和恶意中继，善良中继会在header中告诉你sourceip 恶意中继不是这样
 - 隐藏垃圾邮件的来源
 - 防止被列入黑名单
 - 可以利用僵尸网络实现



-
- 开放http代理
 - squid网络缓冲区
 - 使得squid become a mail relay
- Bobax蠕虫
 - 感染高带宽
 - 缓慢
 - 在被感染的僵尸上安装开放中介

防火墙

- 包筛选器
 - 查看包头决定该包的命运 (IP,TCP,UDP,ICMP) (allow, deny)
 - 无状态&快速
 - ftp包筛选
 - 防地址伪造
 - allow proto=TCP AND (sourceIP=inside OR ACK=true) (单向防火墙)
 - allow port=25 AND destIP=mailserver 只允许 用25端口发给mail服务器
 - 非军事区：大众都可以访问的网络
 - 利用非军事区绕开防火墙？
 - from Internet to the DMZ to protect servers from DMZ to intranet to protect against compromises
 - 限制
 - 没有连接语义 (semantics)，基于单独包
 - 没有应用程序语义——仅仅基于ip和port
 - 数据包碎片——ip允许一个数据包被切片，这可能导致片段攻击 (frgmentation attack)
 - 他不能
 - 禁止特殊的url
 - 检测邮件病毒
 - 屏蔽ActiveX插件
 - 该防火墙被描述为第一代防火墙，其工作在OSI模型的layer3，过滤的参数是静态设定的。其主要根据网络层和传输层的数据包头部，以及数据流的传输方向进行过滤。根据该描述，静态包过滤防火墙和Stateless Packet Filtering防火墙是一致的。由于是静态包过滤，所以该防火墙的效率也是比较高的。该防火墙也被称为无状态分组过滤防火墙，路由器中所使用的扩展ACL即是这种防火墙的典型。
- stateful firewall
 - Reconstruct connection state
 - Make decisions based on **flows**, not on **packets**
 - **Some application protocol parsing may also be done**
 - 例子：telnet
- 应用级防火墙
 - 该防火墙被描述为第三代防火墙。其主要功能是在建立连接之前，基于应用层对数据进行验证。所有数据包的数据都在应用层被检测，并且维护了完整的连接状态以及序列信息。应用层防火墙还能够验证其他的一些安全选项，而且这些选项只能够在应用层完成，比如具体的用户密码以及服务请求。代理服务器防火墙应该属于应用级防火墙的一种具体实现。

	Security	Performance	Modify Client Applications?
Packet Filter	Low	High	No
Session Filter	Medium	Medium	No
App. GW	Hight	Low	Unless transparent, client application must be proxy-aware & configured

- 比较