**Access Control**

**Reference Monitor** = access control concept that mediates all accesses to objects by subjects

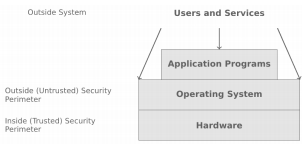
Security Kernel = hardware firmware/software elements of a TCB. Implement RM concept.

* **Mediates all access (Completeness)**
* **Verifiable**
* **protected from modification**

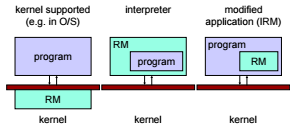
**Trusted Computing Base** = totality of protection mechanisms within computer

* RESPONSIBLE for enforce security policy
* needs correct input by sys admin

Placing RM



* Hardware e.g. AC in microprocessor
* OS Kernel e.g. hypervisor (VM)
* OS e.g. AC in Unix, Win
* Services Layer: AC in DB/JVM
* Application: security checks in app code



**OS integrity**

4 main categories

1. Execution Domains
2. Process separation
3. Memory Protection
4. I/O Controls

OS = enforces ALL access control policies

* To bypass = modify OS = INTEGRITY issue
* OS itself = object of access control
  + But users = able to use and not (MISUSE) OS
  + USES
    - **Status info** (mode of ops)
    - **Controlled invocation** (restricted priv)

OS = distinguish (by OS or by USER)

* Status flag
  + Unix: user or root

Controlled Invocation = sys perform set of OPS in supervisor mode and return to user BEFORE returning back to user mode

* Gate mechanism (transfer control between exec domains)
  + Validates all argument to prevent compromise
* Uses Trap (save everything, exec command, and restore everything)
* Trap and interrupt table = interesting point of attack (redirect pointers)

**HARDWARE**

Core = focus = assurance at higher level

* Can reduce overhead
* Others cannot be bypassed

Registers:

* Program counter
* Stack pointer
* Status registers

ALU

Process

Threads

* Avoid full context switch, but avoid potential security control

Execution Domains

* User/supervisor Mode (not adequate)

**Protection Rings**

* Each subject and object assigned number
  + 0 = OS Kernel
  + 1 = OS
  + 2 = utilities
  + 3 user processes
* 0 = highest degree of protection

Intel:

* 2bit field (4 priv levels)
* Simplified (from multics)
* Use gates (equal or lower)

Intel Gate:

* 1. Enter gate, set ring level to that gate  
  2. Param validation (TOCTOU problem)

(copy stack into inner ring, if in outer ring can be manipulated)

TOCTOU = time of check, and time of use (malicious activities after checking, and before using)

Descriptors

* prive level, access control tables, gates
* accessed by selector

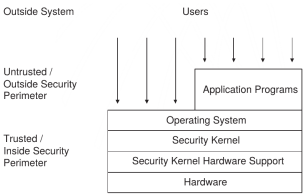
Current Priv Level (CPL)

* looking at descriptors

Confused Deputy Problem (luring attack)

1. invoke subroutine through gate, CPL change to level 0
   1. on return, CPL is restored to 3
2. Outer ring ask inner ring procedure to copy inner ring object to outer ring
3. Luring attack/confused deputy problem

* ARPL instruction (adjust priv)
* Will set to outer ring priv when copying out

**Security Kernel Concept**

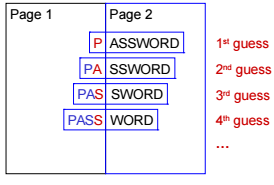
Process Separation (protect those on the same priv level)

Segmentation (logical)

Paging (physical)

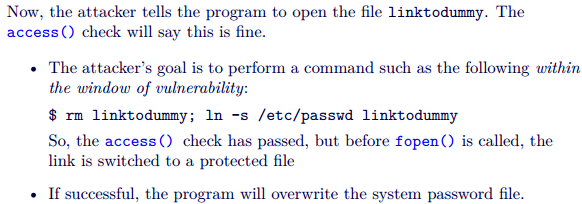
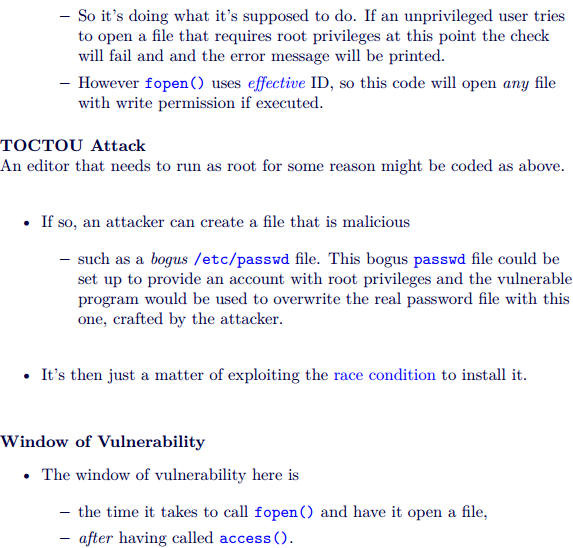
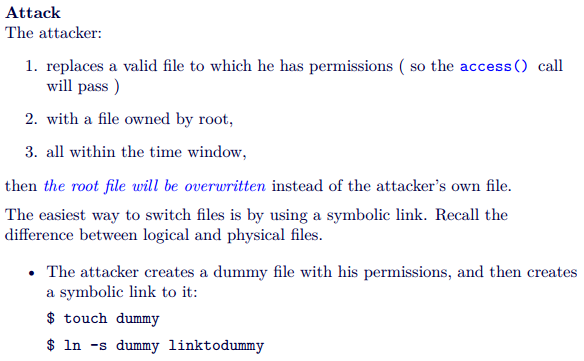
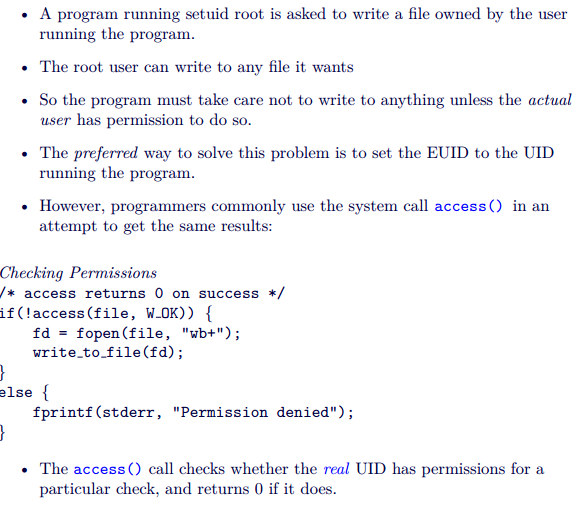
* Page fault (create covert channel)

Covert Channel Exploitation

1. Stored more than 1 page = page fault
2. Password scheme where password checked char by char.
3. Access denied when incorrect
4. Password stored a page boundary, first page = correct, next page = wrong  
   

Secure Addressing

* Memory protection
* Address sandboxing (segment identifier and offset)
* Relative addressing (uses fence registers [top of user space[)
* Bounds checking (bounds register [btm of user space])

**TOCTOU problem**

**Avoiding TOCTOU Problem**

* dealing with file descriptors or file pointers, we ensure that the file on which we are operating does not change behind our back after we first start dealing with it
* do an fstat() on the resulting file descriptor.
* avoid doing your own access checking on files
  + set the EUID and the EGID to the appropriate user
* Use truly random filenames
* use file descriptors instead of file path names
* Perform file descriptor binding first
* Verify/validate the assumptions made about the file system

**Memory Errors**

**2 outcomes**

The system detects the anomalous situation, and it blocks the operation (e.g., Java), or 2.

The system cannot detect the anomaly, and the operation is executed (e.g., C)

**Why?**

Some languages (e.g., C) lack of proper boundary checks

Channelling problem: data and control elements share the same channel.

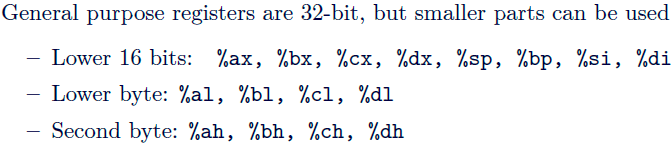
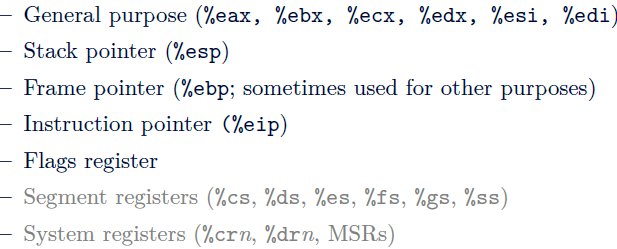
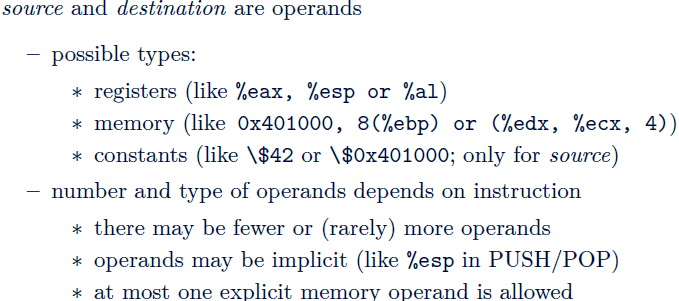
* Channel (stack, heap)
* Data (func args and local var, heap management metadata)
* Control elements (code pointers, data pointers)
* Popular
  + Arch/OS dependent
  + Local & remote
  + Can modify data/control flow of an app
* Tools = make it easier

**Assembly**

AT&T syntax



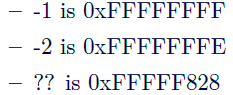
**Mnemonic =** 





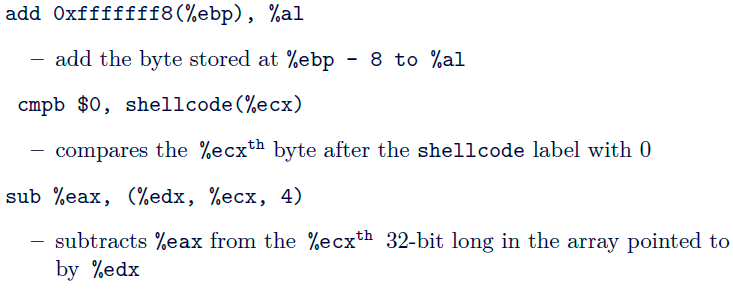
**Intel uses little endian**

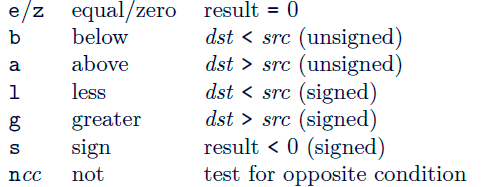
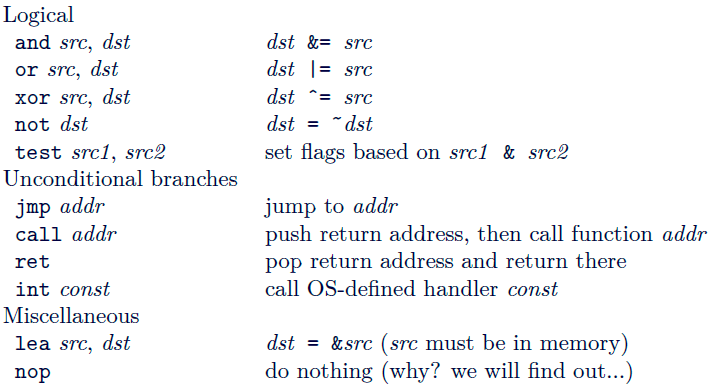
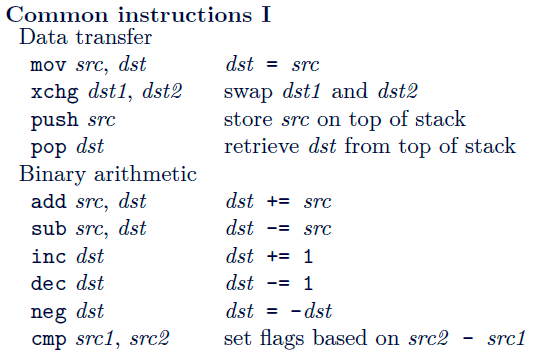
Signed integers = 2’s complement, Sign flip bits

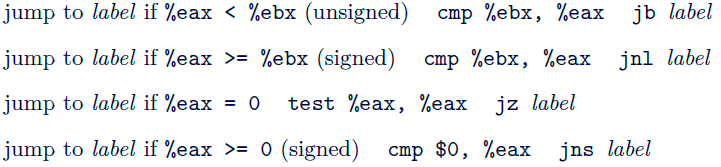


XOR %eax, %eax = zero itself

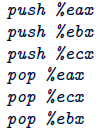
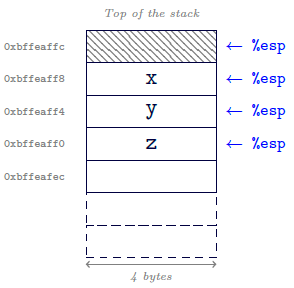
Movl $42, 0x401000   
= store 42 in 32 bit long address 0x401000



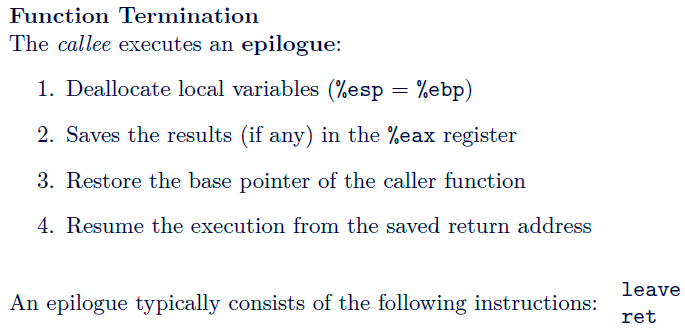
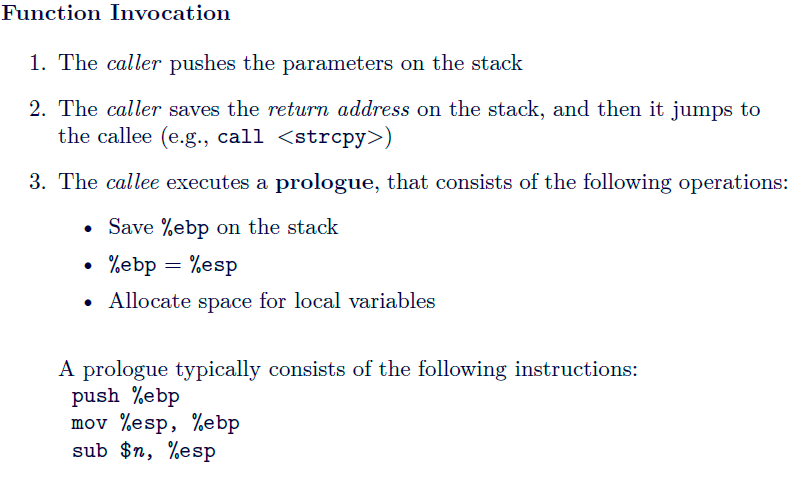
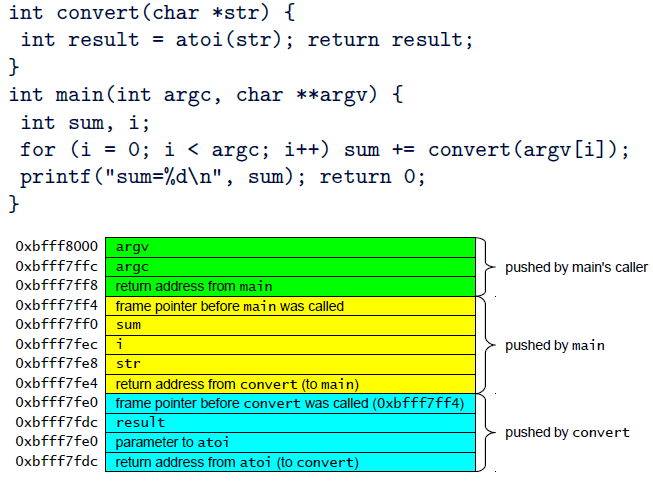
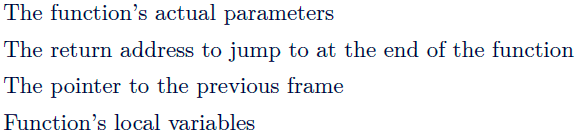


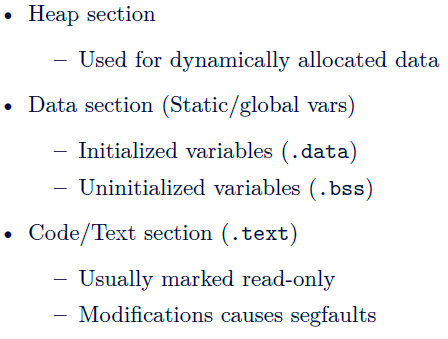
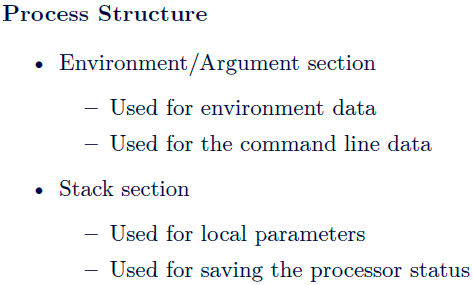
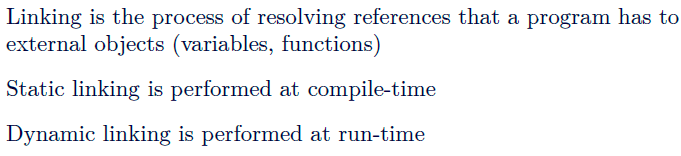
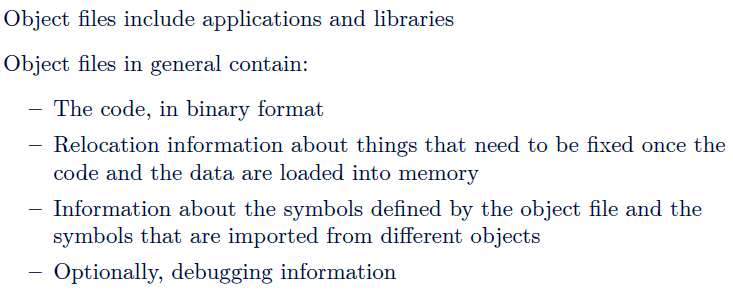


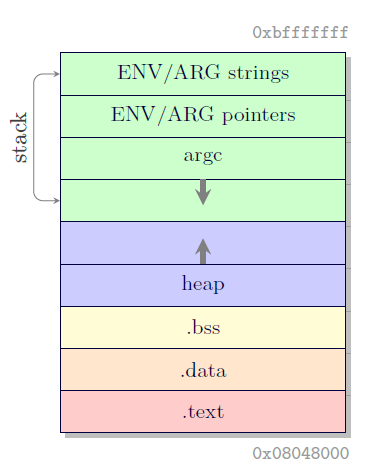
Stacks



**Stack = composed of frames (consequence of function calls)  
addr of current frame = %ebp**



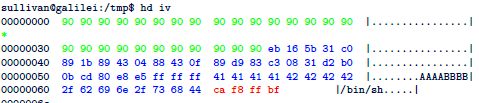
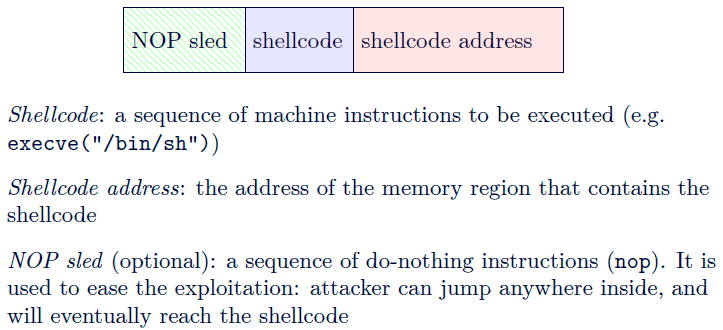
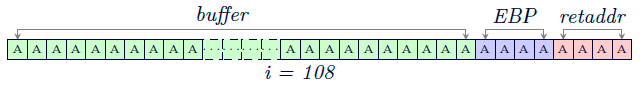
**Processes**



**Stack overflow**

If correctly crafted, it is possible overwrite the return address with a user-defined value

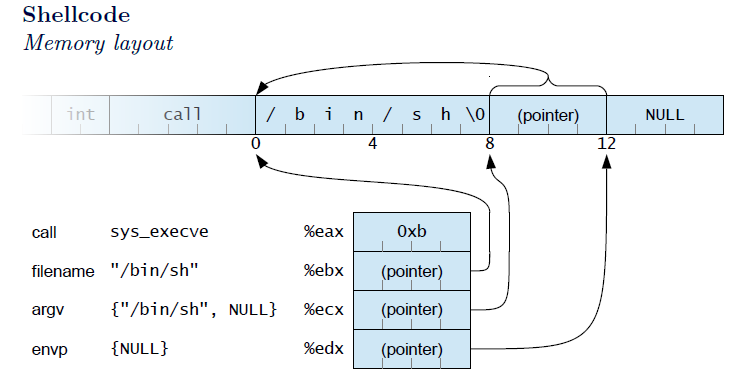
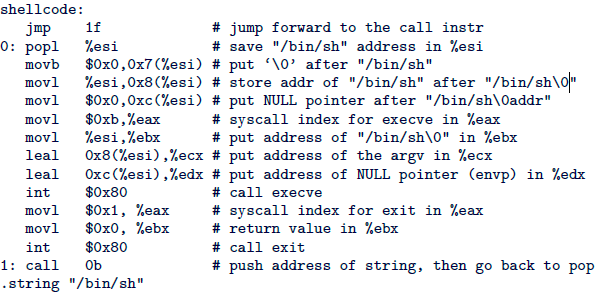
Functions: gets() strcpy(), strcat(), 





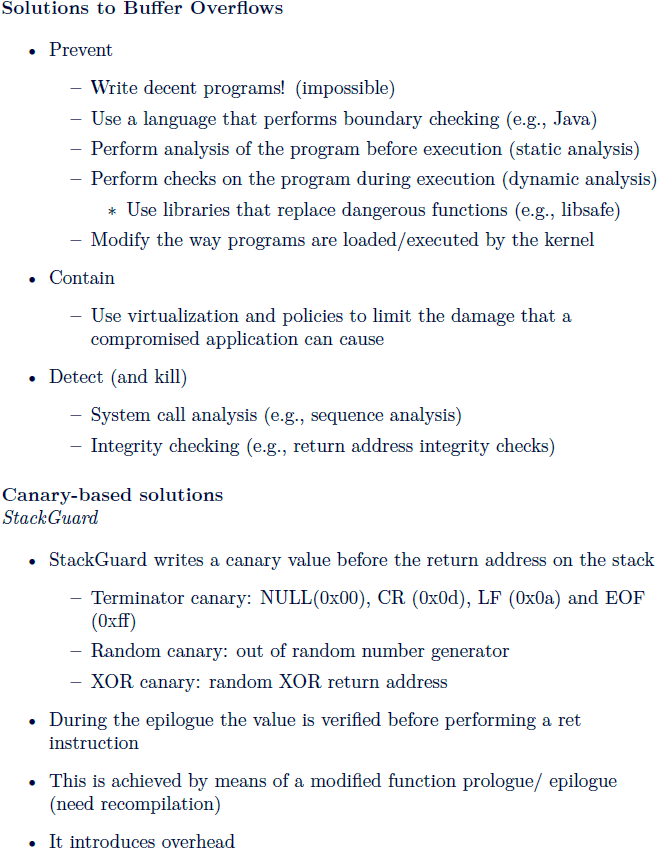
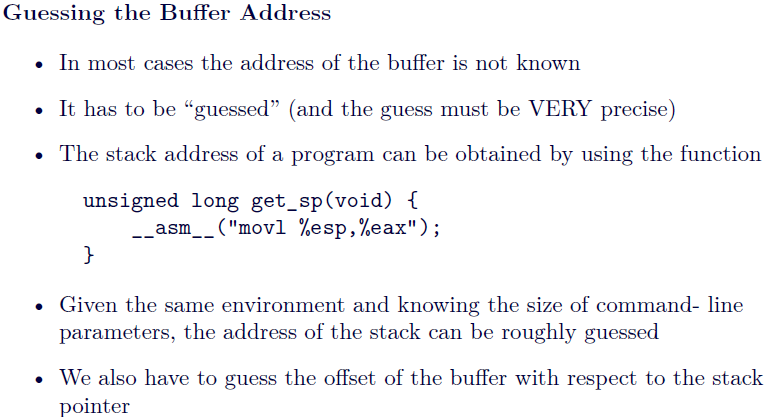
Requires NULL pointer to end the string (reuse null pointer at end of argv)

Position of code is unknown, but requires string address, so position a call instr before /bin/sh and ref it



Null bytes = stop copying

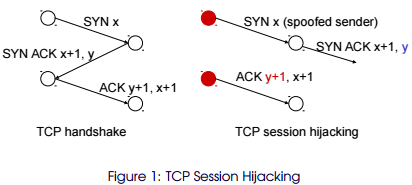
* Use XOR to make it zero
* Mov null to it
* Copy upper space to lower space in the register

Value of 1 = XOR, then INC  
  


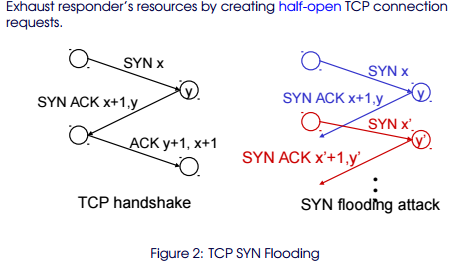
**Network Security**

* Bot and bot controllers
* Net adversary able to:
  + Read messages directly
  + Spoof arbitrary sender addr
  + Try to guess fields in unseen messages

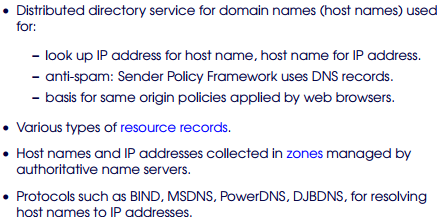
**TCP Session Hijacking**

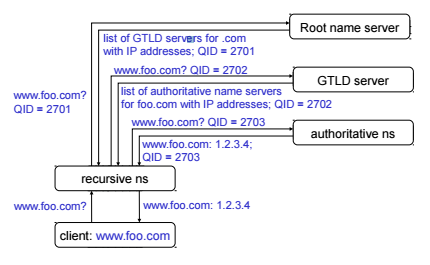
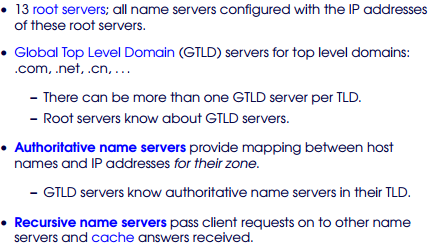


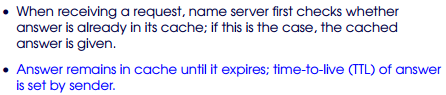
**TCP SYN Flooding Atk**

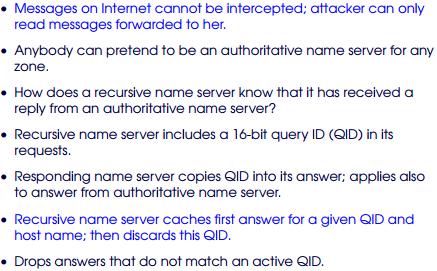


**DNS**



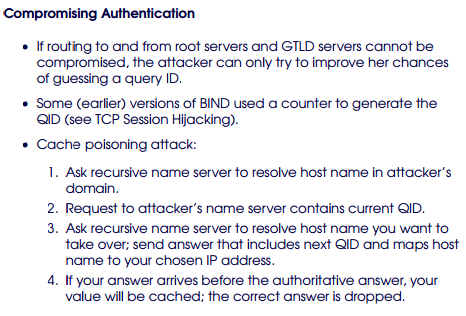




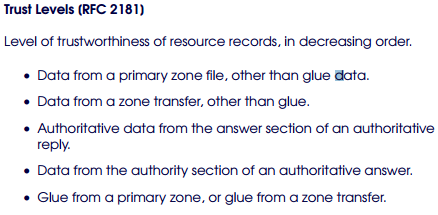
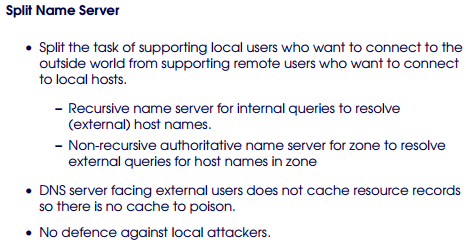
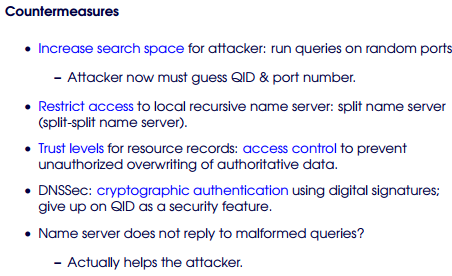
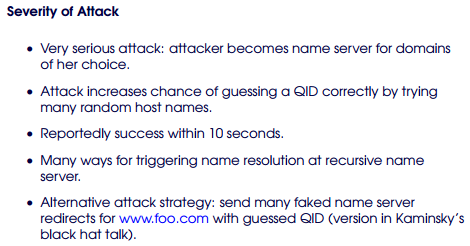
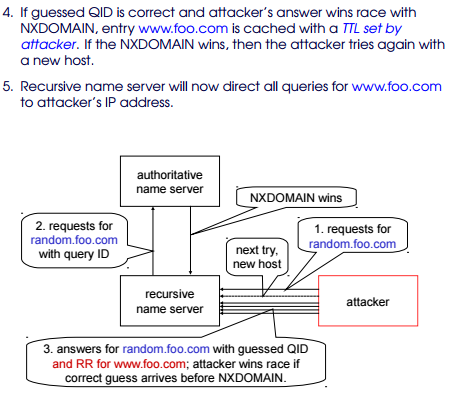
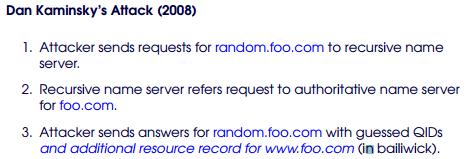
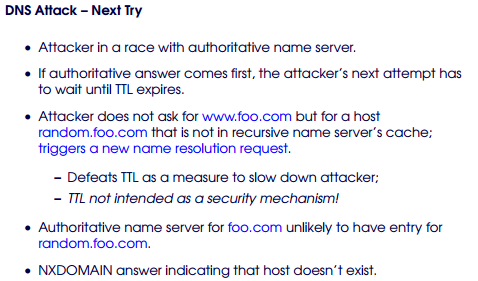
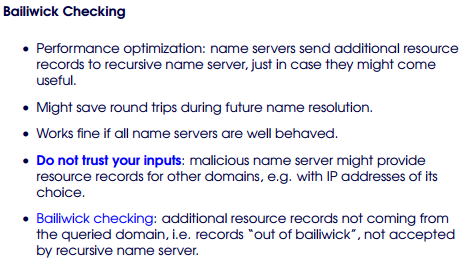
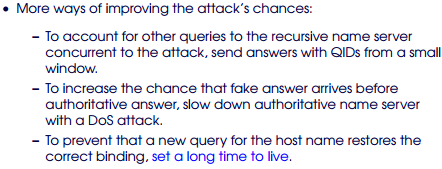
**Lightweight authentication**

**Security? 1 in 2^16**

* Relies on assumption routing from local recursive NS to authorative NS =correct

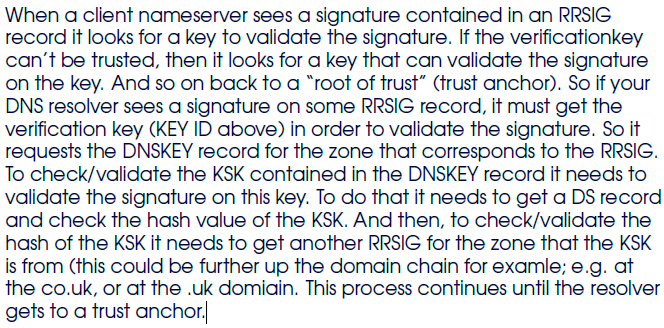
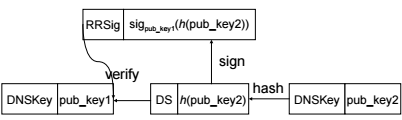


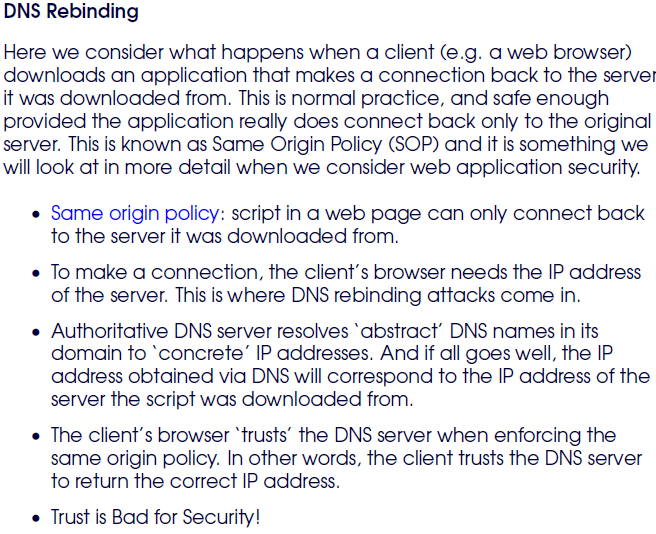
**Don’t use predictable challenges.**



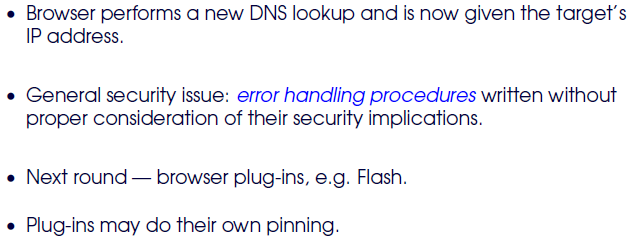
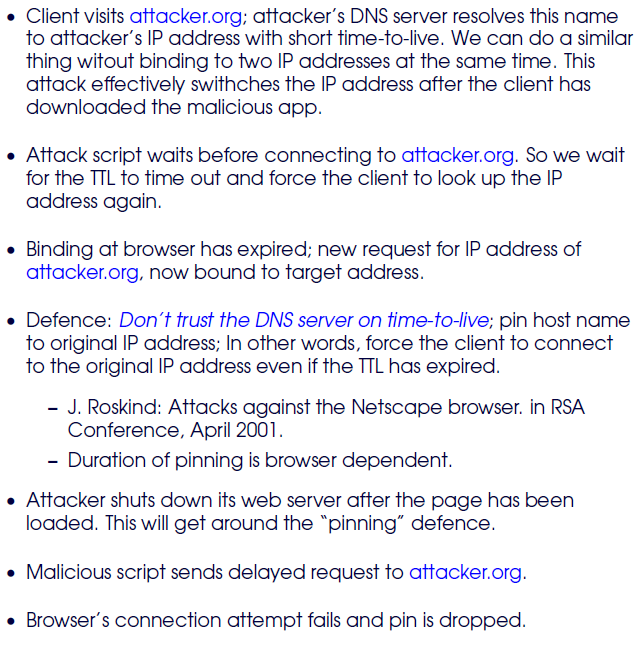
DNS SEC

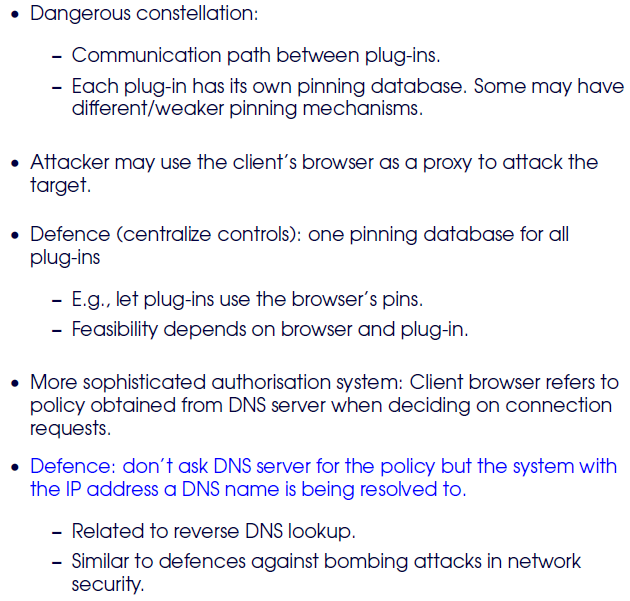
* Protect authenticity + integrity of resource records with Dig Sig
* RRSIG (contain dig sig)
* DNSKEY (public keys of zones)
* DS (delegation signer) = hashes of DNSKEY records
* Authentication chains built = alternating DNSKEY and DS resource records
* Public key in DNSKEY resource record used to verify the sig on next DS ressource record
* Hash in the DS provides link to next DNSKEY resource record
* Verification in resolve find a trust anchor



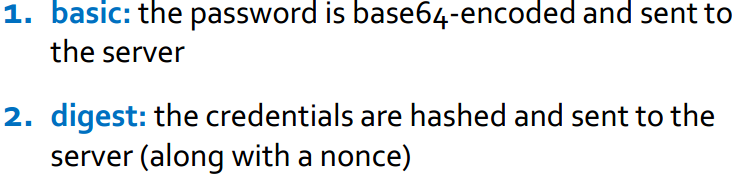
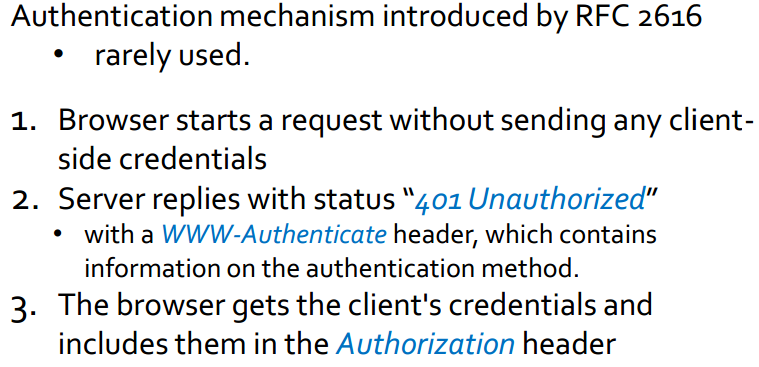
**DNS REBINDING ATTACK**

* Abuse trust – bind 2 ip to domain name
* Download script – connect back using same origin policy
* SOP based on domain name

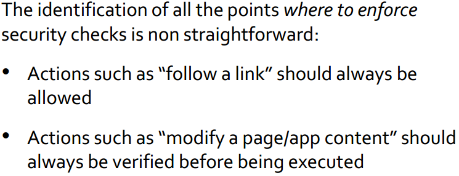




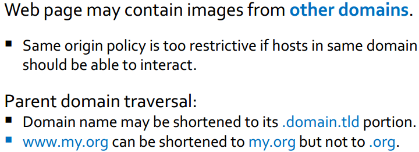
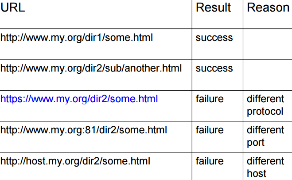
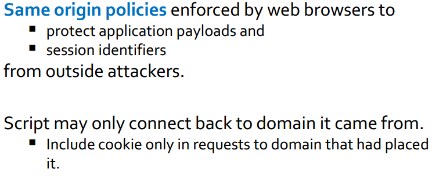
**HTTP Authentication**

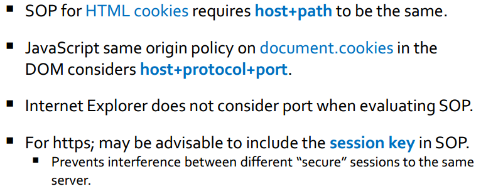
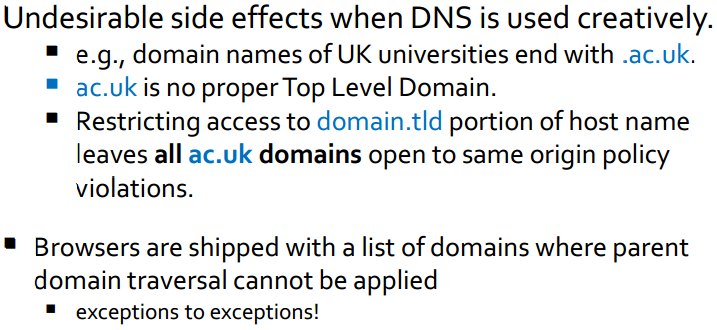
****

**Content Isolation**

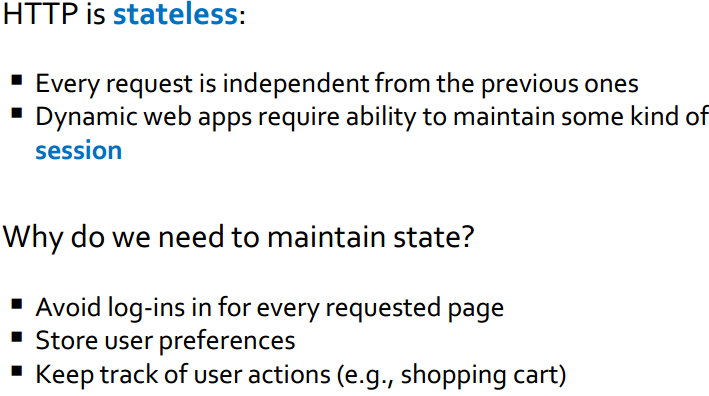
****

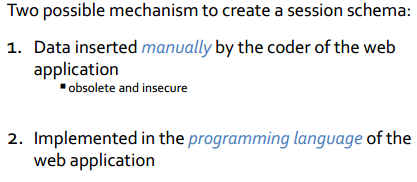
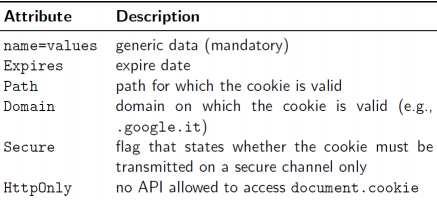
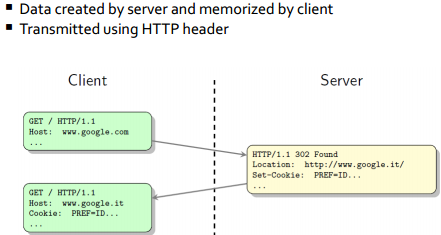
**Same Origin**

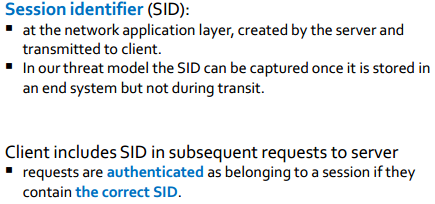
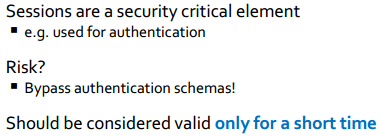
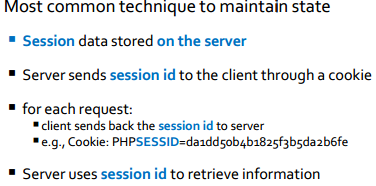
****

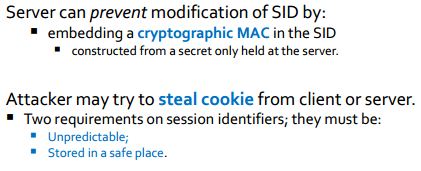
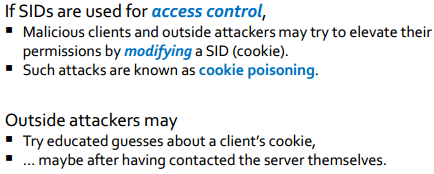
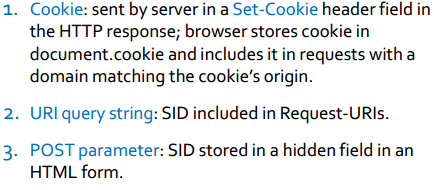
****

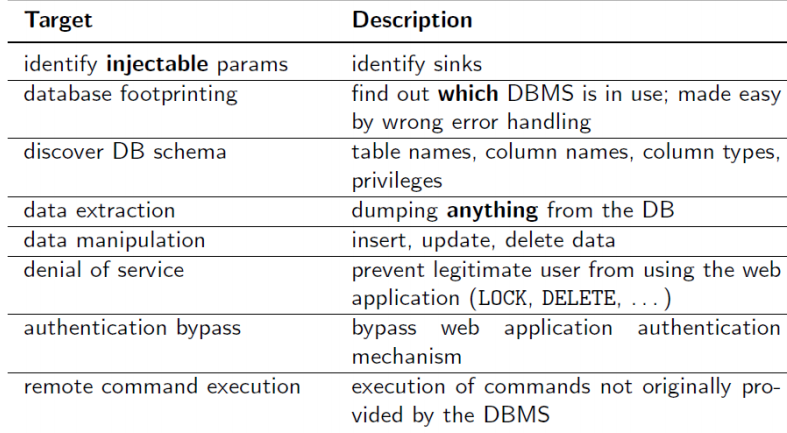
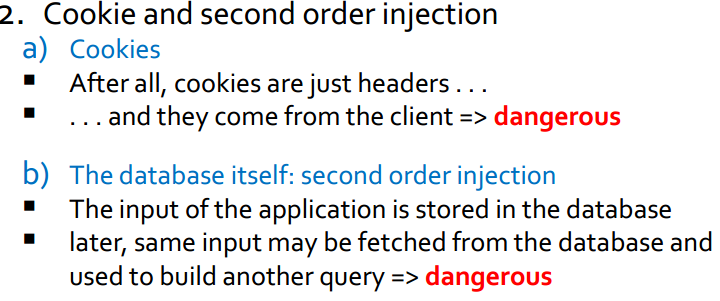
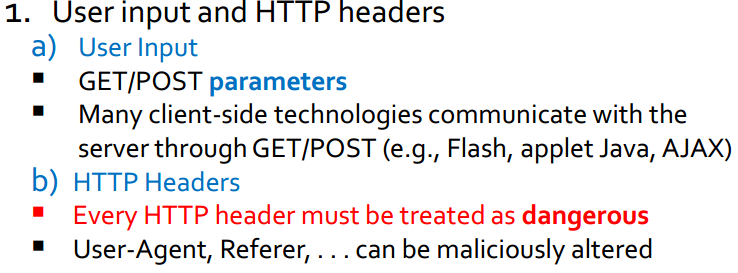
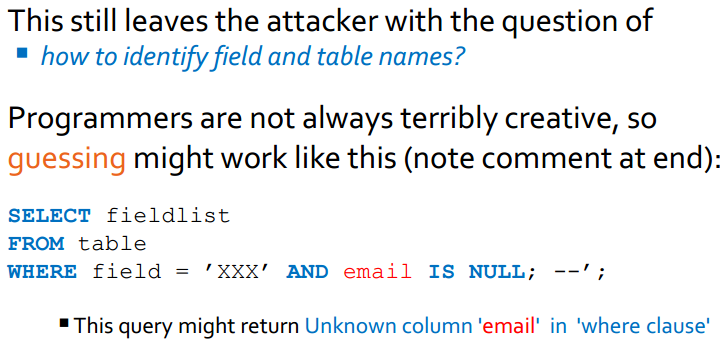
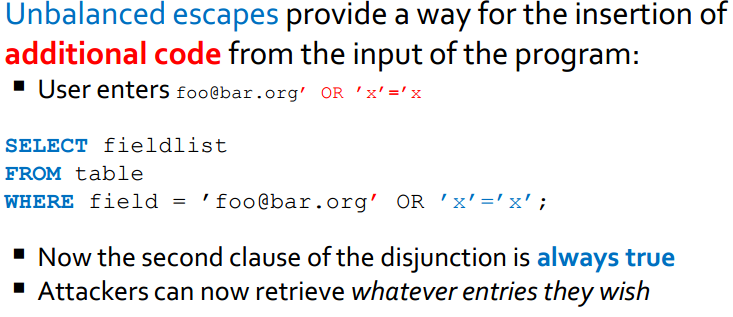
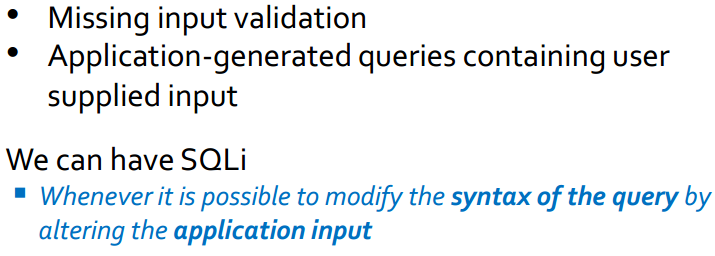
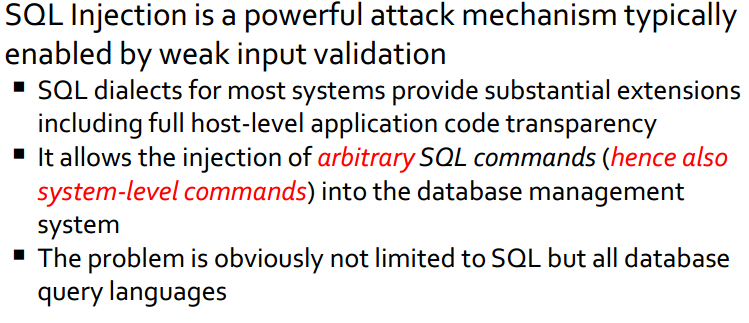
**HTTP Session Cookies**

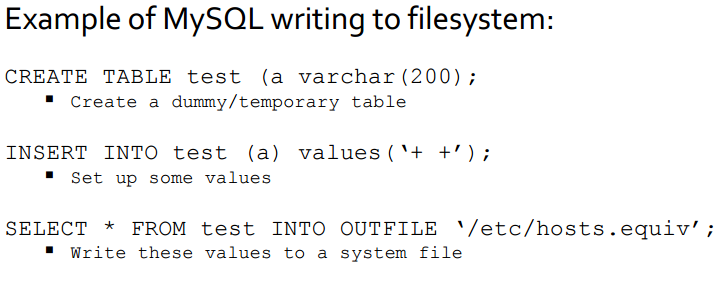
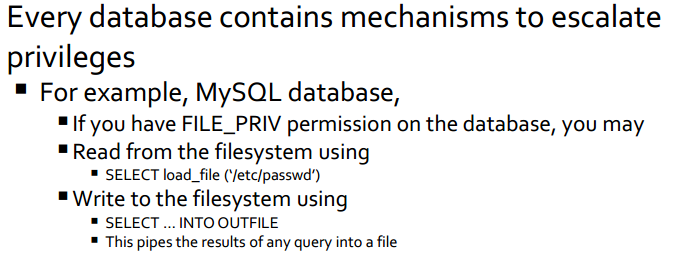
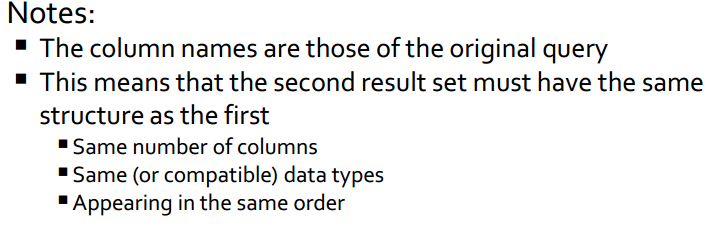
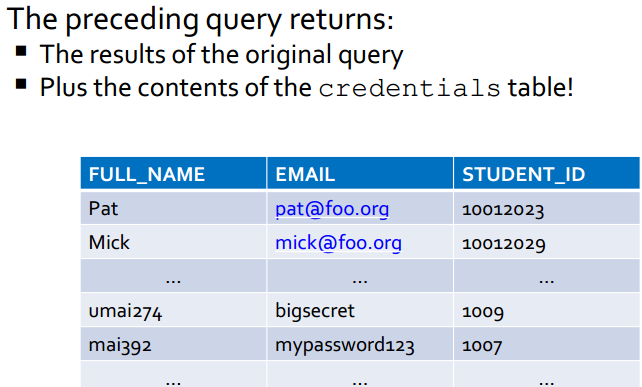
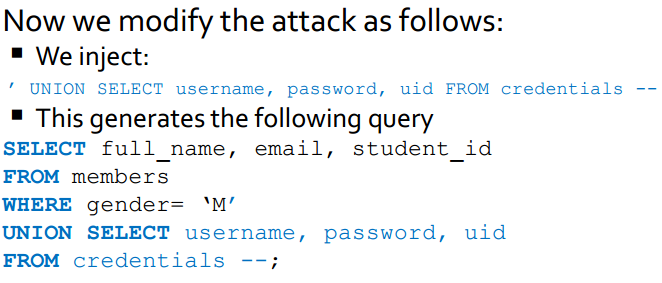
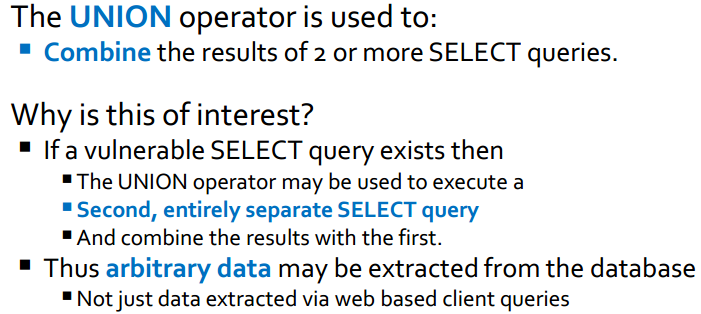
****

****

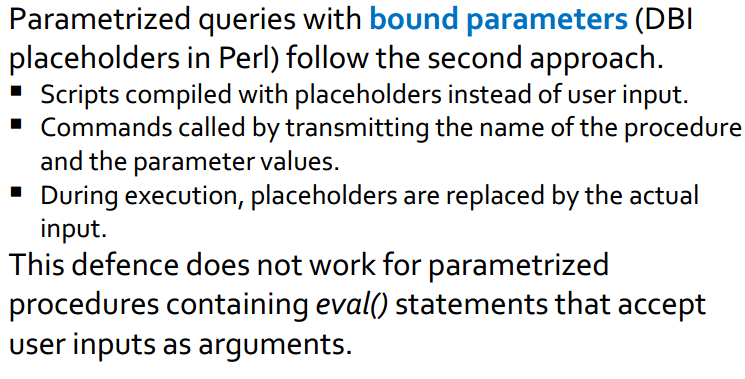
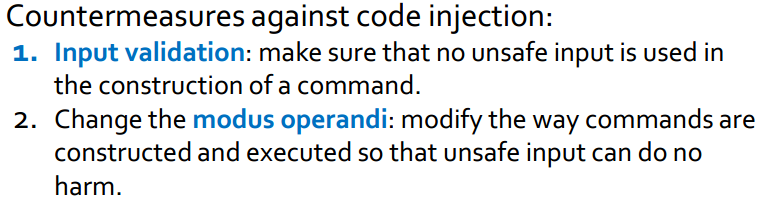
****

****

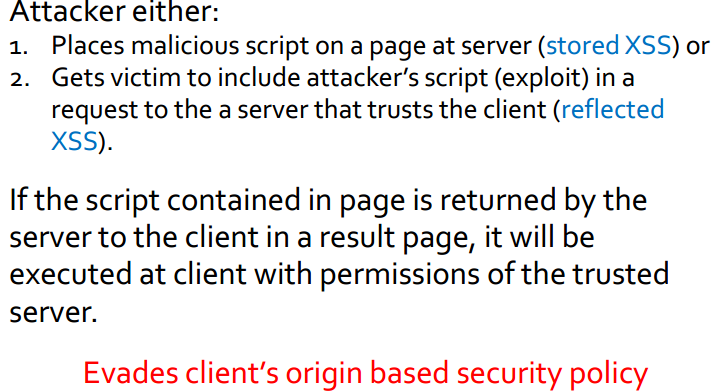
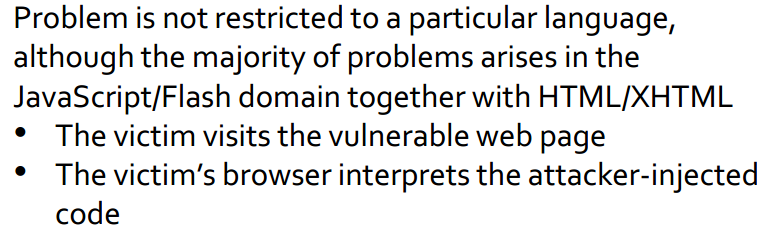
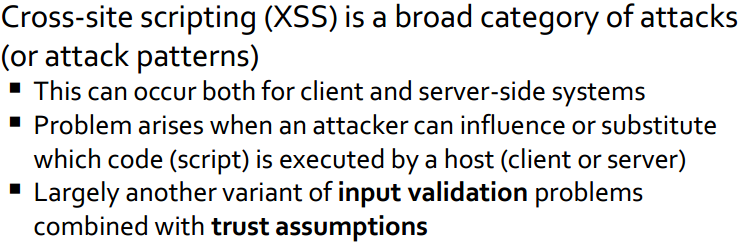
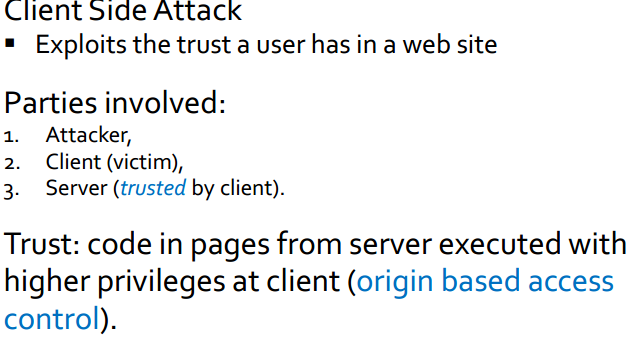
**SQL injection  
Advanced SQLi**

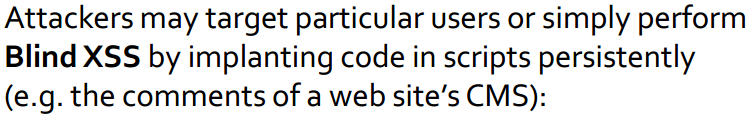
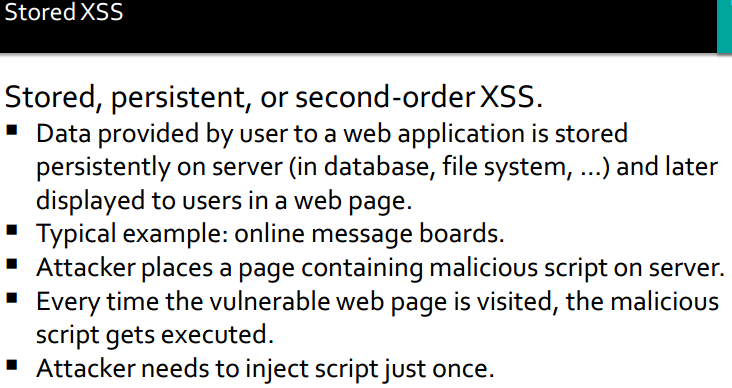
****

**Counter measures**

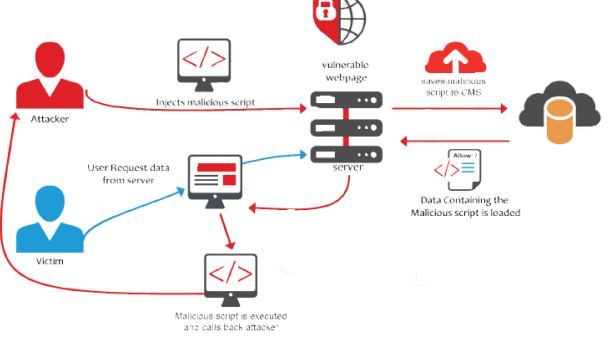
****

**XSS**

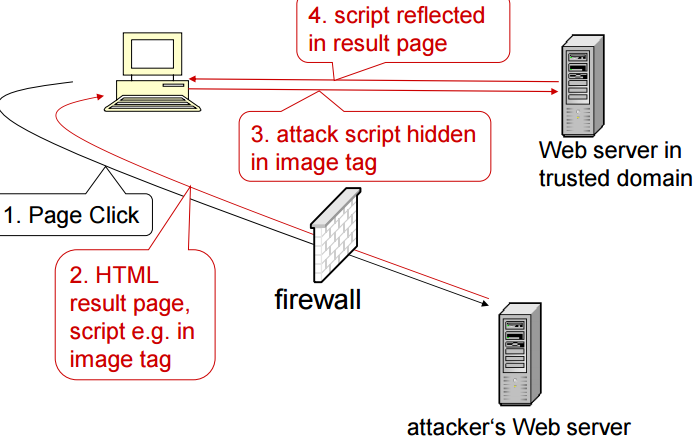
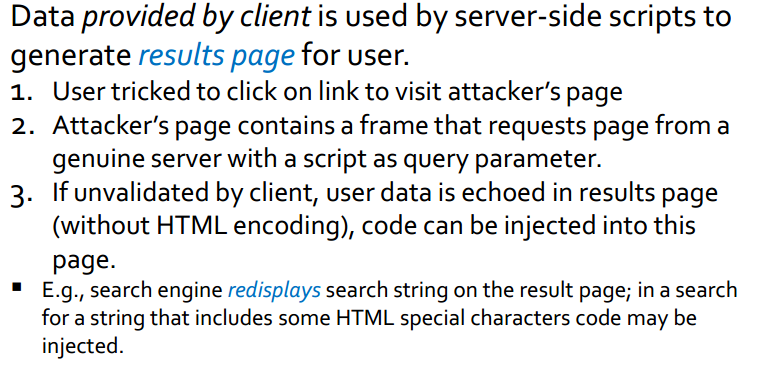
****

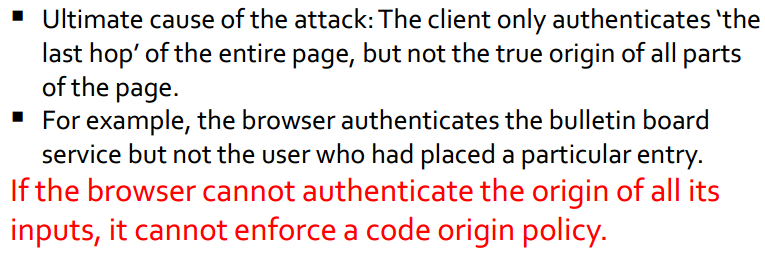
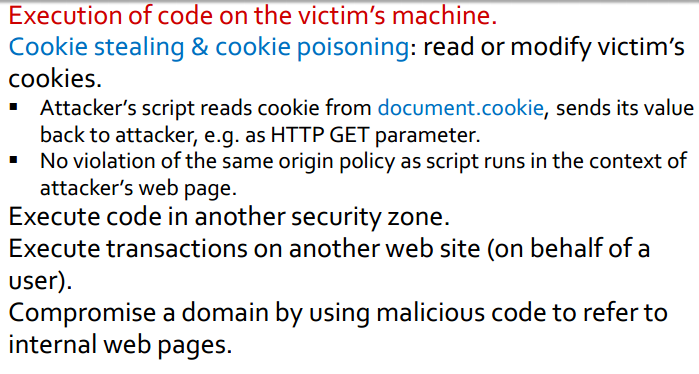
****

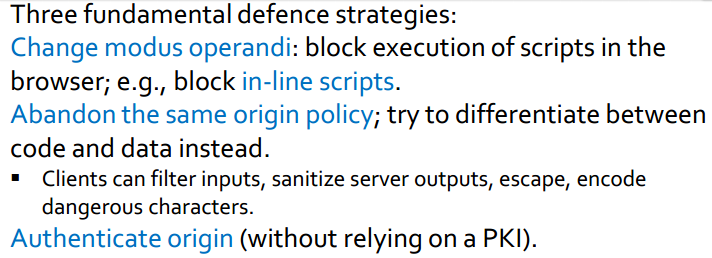
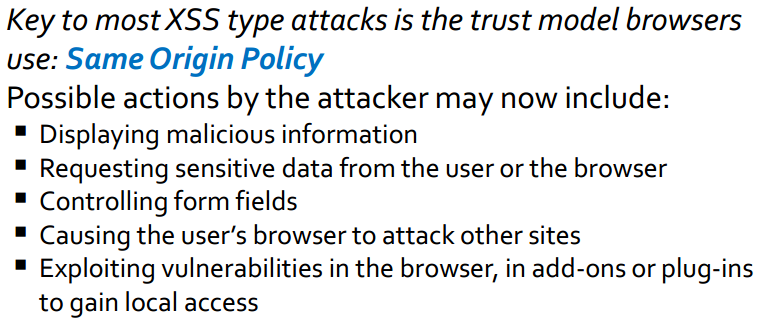
**SOP + input validation = XSS**

****

**Reflected XSS**

****

****

****