**Access Control**

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

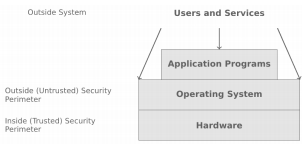
Security Kernel = hardware firmware/software elements of a TCB. Implement RM concept.

* Mediates all access
* Verifiable
* protected from modification

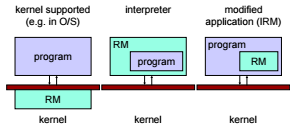
Trusted Computing Base = totality of protection mechanisms within computer

* RESPONSIBLE for enforce security policy
* depends solely by correct input by sys admin

Placing RM



* Hardware e.g. microprocessor
* OS Kernel e.g. hypervisor (VM)
* OS e.g. access control in Unix, Win
* Services Layer: 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 should be 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

* Program in execution

Threads

* Avoid full context switch, but avoid 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:

* Param validation and TOCTOU
* part of stack =copied to stack

TOCTOU = time of check, and time of use

Descriptors

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

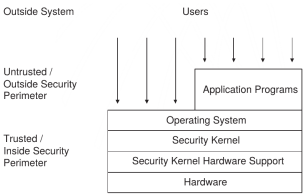
Current Priv Level (CPL)

* 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 outerring priv when copying out

**Security Kernel Concept**

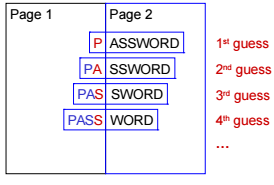
Process Separation (protect those on the same priv level)

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])

**UNIX Security**

Discretionary access control

“secure” version of unix available.

Principals: UIDs and GIDs

UID, superuser = 0, differ from sys to sys

* Cannot write to RO file sys, but can remount as RW
* Cannot decrypt password, but can reset them

/etc/passwd

/etc/group  
groupname:password:GID:listofusers

* Convenient for access control

Subjects

* Process, PID
* Generated with exec/fork
* Real UID/GID = inherit from parent, UID/GID of user that logged in
* Effective UID/GID = inherit from parent process

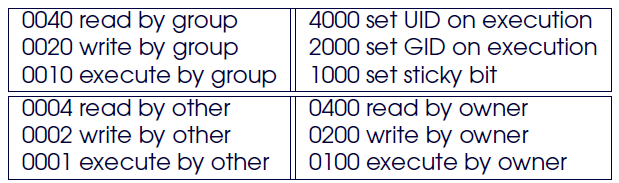
Passwd

* Crypt(3) = one way function
* Modified DES with all-zero block as start, password as key, repeat 25x

OBJECTS

* Files (resources)
* File entry in directory = pointer to data struct (inode)

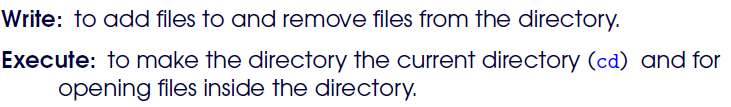
Sticky bit



Umask

* Adjust permission (changing default)
* Logical AND of the bits

Directory Permissions



Look at UID, then GID, then others (when checking permission)

**SECURITY Principles**

Controlled invocation: SUID programs

Physical/logical representation of objects: deleting files

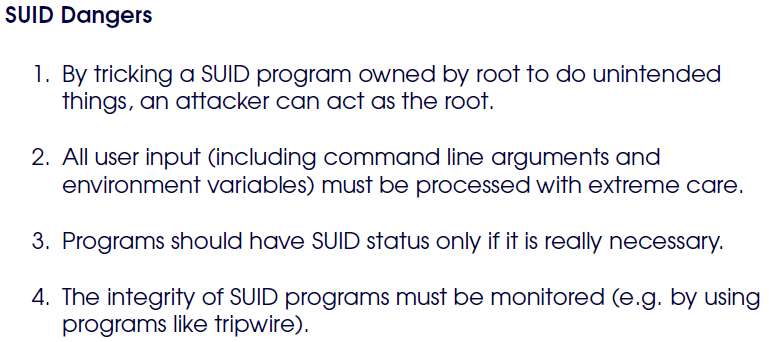
Access to layer below: protecting devices

Search Path

Importing data from outside: mounting FS

Controlled invocation:

* SU priv required to exec OS functions (listen to ports)
* Use sticky bit
* Should only do what the owner intended.



If deny users direct access, need to perform their job = need SUID program

Applying Controlled Invocation

* Set up new UID that owns the resources, and all programs that need access to the resources
* Only owner get access permission
* Define all programs that access the resource as SUID

Logical VS physical memory

* Cp and Ln
* Hardlink = still remains after linked file deleted
* After deletion, needs to be wiped.

Access to layer below:

* Bypass controls set on files and directories if get access to memory devices (/dev/\*)

Process status

* Ps display information about mem usage
* Compromise of ps = attacker with root
* Let group “mem” own memory device and define ps as a SGID program

Terminal Devices

* Owner of file for session (gives RW permission to file, so user can receive messages from other parties)
* But other parties can monitor traffic to/from terminal (can send commands to other terminals)

Mounting FS

* Import objects from other security domain, access control redefined. (use RO mount, nosuid, noexec, nodev)
* Scoping of identifiers. UID and GID = local identifiers. May misinterpret UID or GUID to enforce access ctrl

Environment Var

* Invoker of setuid/setgid program = in control of environment variables given (inherited, transitively)

Changing Root of FS

* Chroot: restricts available part of FS
* Like setting home directory
* System files are ‘expected’ to be in directories

Search PATH

* Exec of programs taken from wrong location
* Hijack PATH by putting malicious file at the start of the PATH
* Use full path /bin/su instead of su

INETD daemon

* Config file maps port num to programs

Telnet Wrapper

* Change inetd config file to run another app instead (IP address filtering)
* Design principle: add another level of indirection

**Management**

Separation of duties

* Create users to deal with networking. If compromised, not all is lost

Root

* Su, /etc/passwd, /etc/ppasswd

Trusted host

* Enter when username matches, no need passwords (specified with /etc/hosts.equiv and rhosts)

Audit Logs

* Lastlog, last user that logged in
* Utmp, accounting info used by who
* Wtmp: records when user logs in/out
* Acct, all executed commands

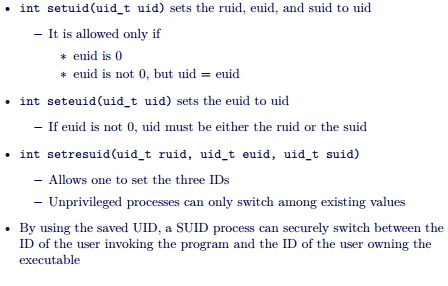
**Application Security**

Violation of CIA

App Vuln = process of identifying vuln in app, deployed in specific ops env

* Deployment
  + Incorrect config
    - Overpriv
    - Faulty security policy/mechanism
  + Implementation vuln
    - Not correctly handle unexpected events
      * Unexpected input/errors/exception/interleaving
* Local
  + Allow one to manipulate behaviour of app through interfaction
    - Require presence on host
  + Allow one to exec ops with priv (diff from attack would have)
  + Easy to perform, attacker = better knowledge of env

SUID

* + Real UID: user who start
  + Effective UID: used to determine if process allowed to do
  + Saved UID: used to drop and regain priv
  + 
  + Most for priv escalation
    - Based on input
      * Startup (command line, env)
      * During exec (dyn linked obj, file input)
    - Interaction with env
      * File sys: creation of files, access to files
      * Processes: signal, invocation of other commands
  + Bad input
    - Mem corruption (buffer overflow)
    - Command injection
  + Faulty exec condition
    - Errornous exception handling
    - Race condition
    - Info leak
* Remote
  + Network-based interaction
    - Unauthenticated remote attacks
  + Allow one to exec ops with priv of vuln app
  + More diff, no prior access

**Attacks**

Env, input, file access, overflow, format string attacks

Environment

* **System() / popen() = dangerous / execlp() execvp()**
* Popen() creating a pipe forking + invoking shell in sys
* Exec of external commands/apps invoked by influencing environment
* Path substitution, modifieds own PATH or HOME car to control exec commands, induce scripts
* IFS Attack (substitute value of IFS), interpreted as space-separated list of local command
* Modern shell reset this variable
* Environment should be known state

**Preserve Attack**

* Used by VI to make backup (can restore from sudden death
  + Ran SUID to protect privacy
  + Used /bin/mail to send email invoked by system
* Change IFS t “/”
  + Create program named bin in current dir, kills VI program

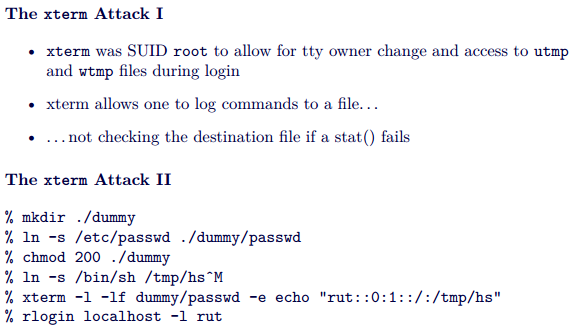
**CLI param**

* CLI params used (no size checking, no sanitization)
* User provided data can used to perform
  + Command injection
  + Directory traversal attacks
  + Overflows
  + Format string
* Check for length, sanitize, general validation problem, sql injection

**Playing with FS**

* App uses temp files for logging/locking
* Do not check if exists/sym link
* Not specified by user, predictable.
* Erroneous handling == bypass of security check
* Creates sym link to file by superuser and invoke app

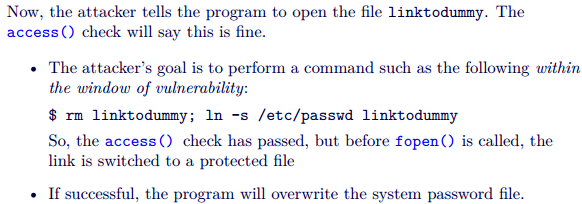
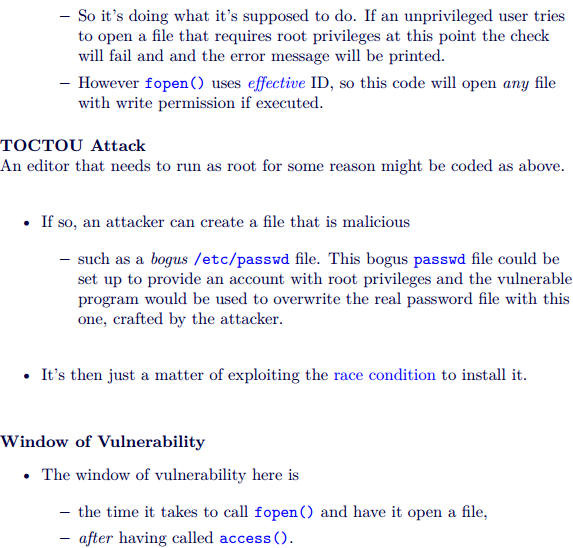
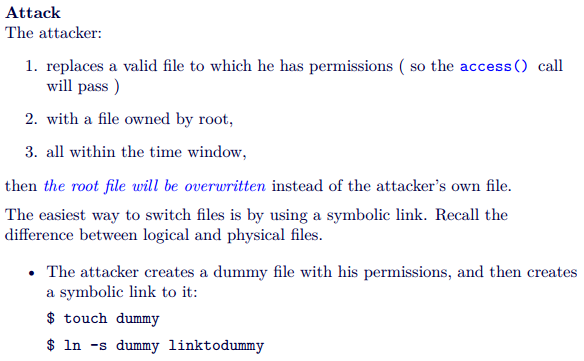
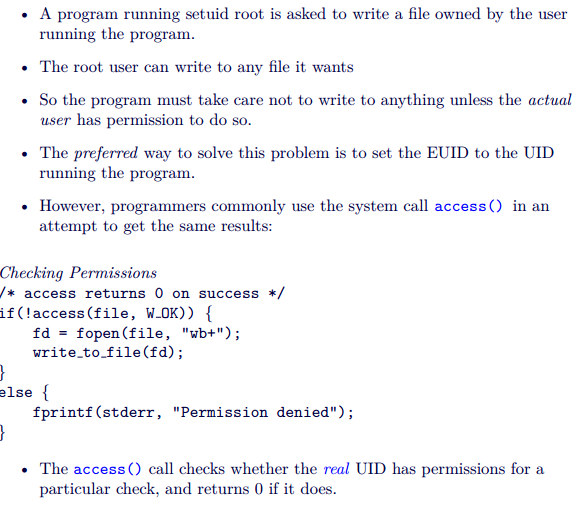
**Example**

* Dtappgather attack
* Xterm attack
* 

**Playing with FS (tocttou)**

* Race conditions attack
* Time of check (t1), time of use (t2), time of attack (t3)
* T1<T3<T2
* Data race condition
  + conflicting access of multiple processes to shared data (1 of them = Write access)

**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

**Attacks**

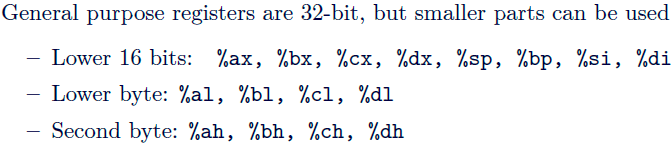
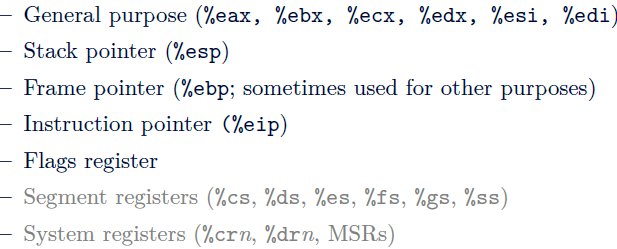
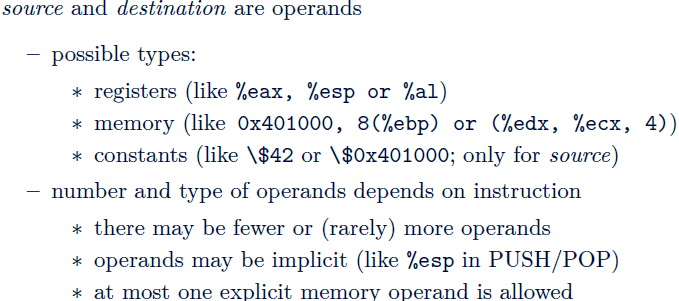
* Stack-based overflows
  + Shell code injection
  + Return into libc
  + ROP
* Heap overflows
* Integer overflows
* User-controlled format strings

**Assembly**

AT&T syntax



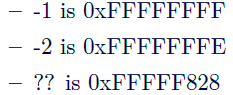
**Mnemonic =** 



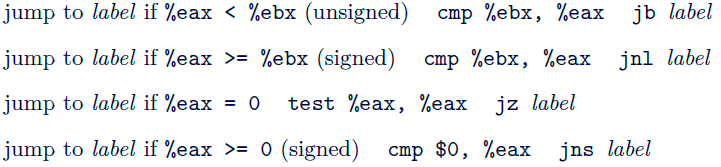
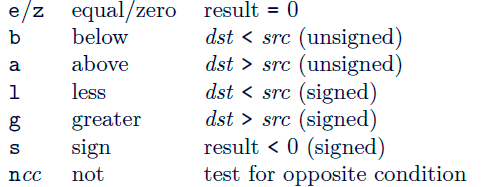
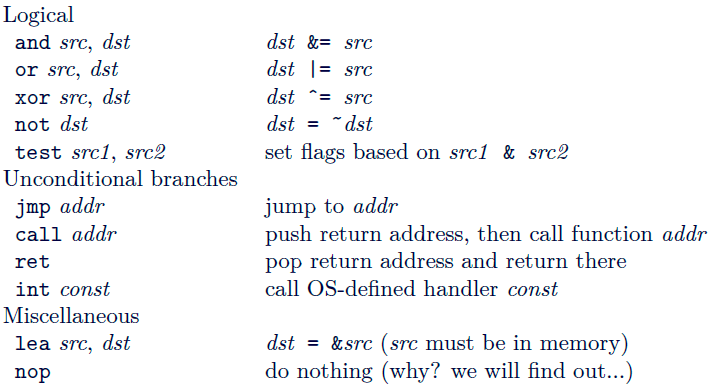
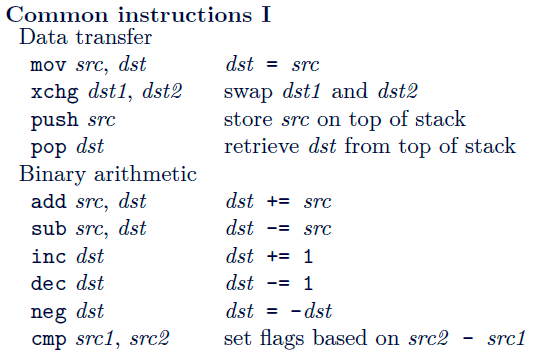
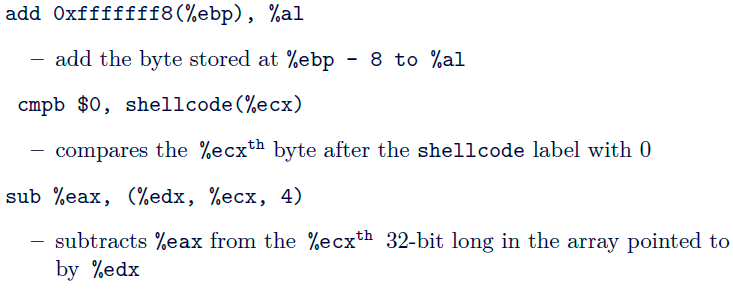


**Intel uses little endian**

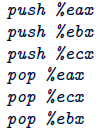
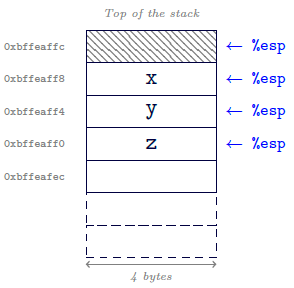
Signed integers = 2’s complement, Sign flip bits



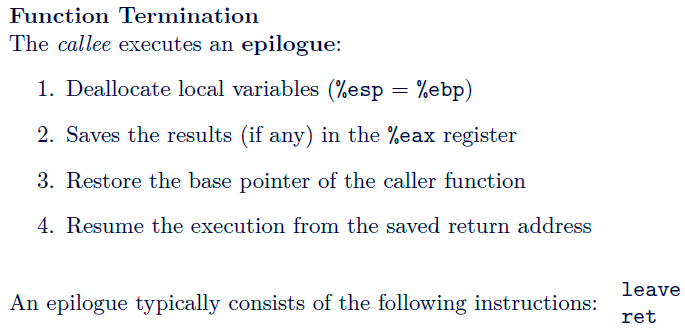
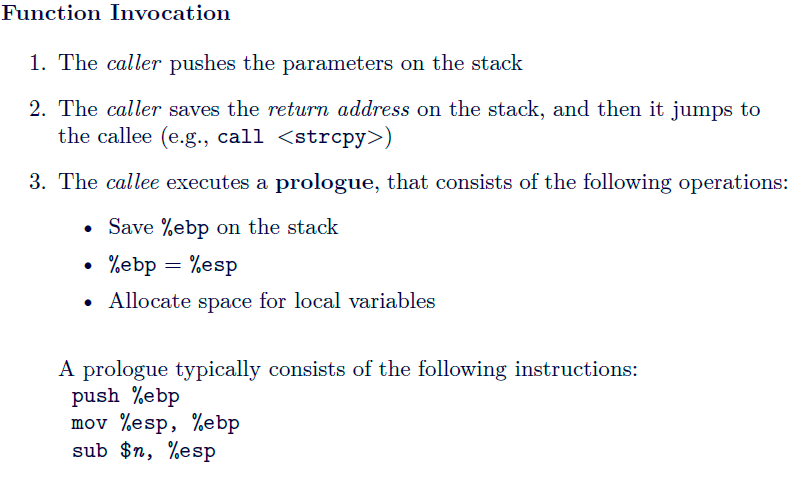
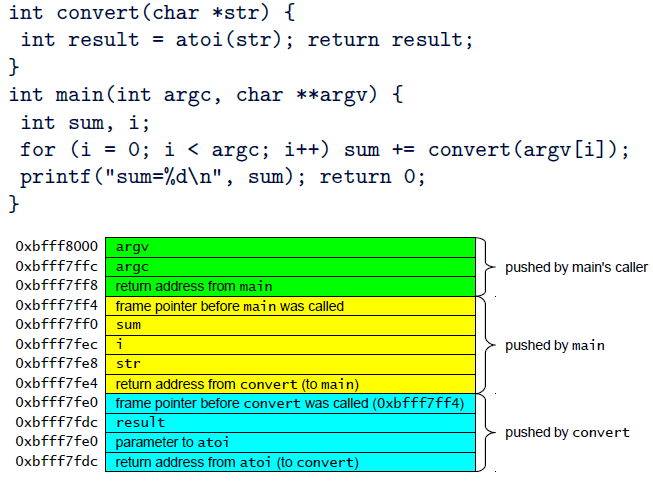
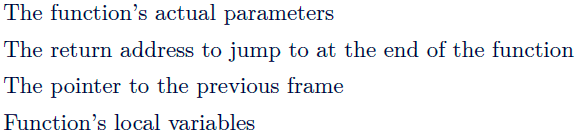
XOR %eax, %eax = zero itself

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

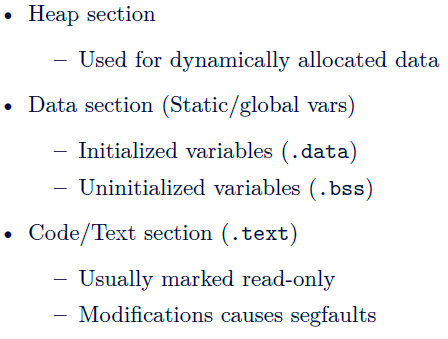
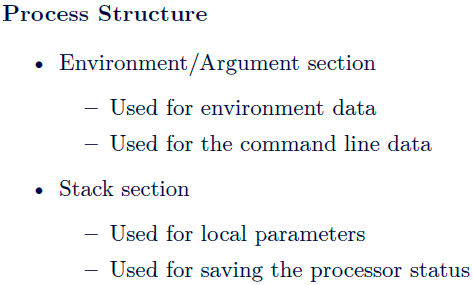
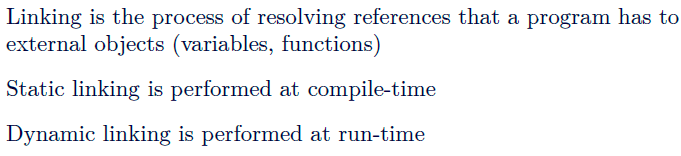
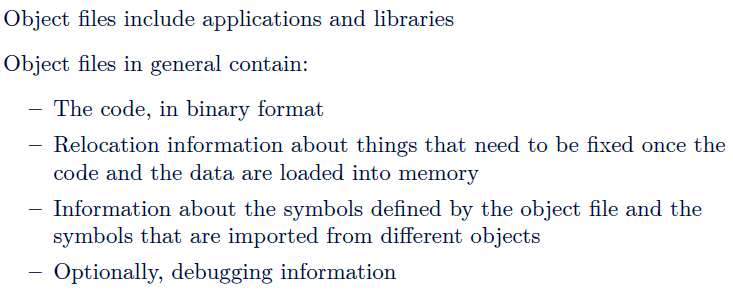
Stacks

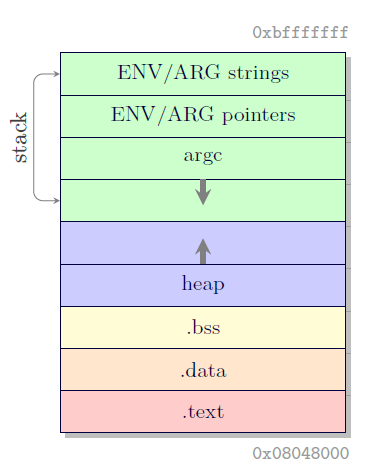


**Stack = composed of frames ( consequence of function calls)  
addr of current frame = %ebp**



**Processes**

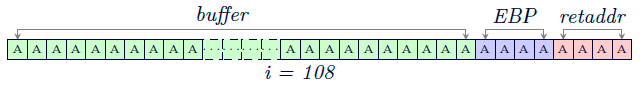


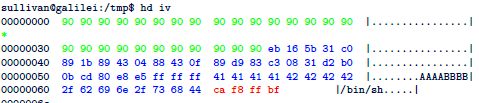
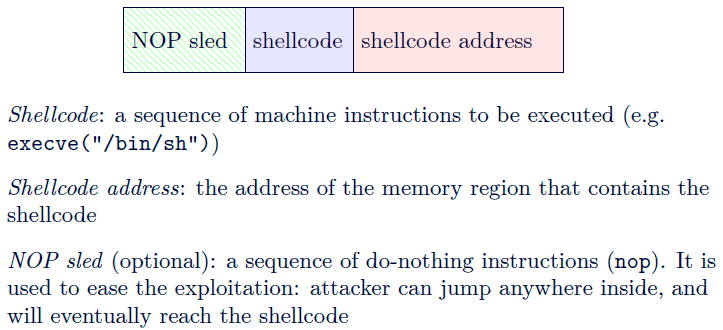


**Stack overflow**

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

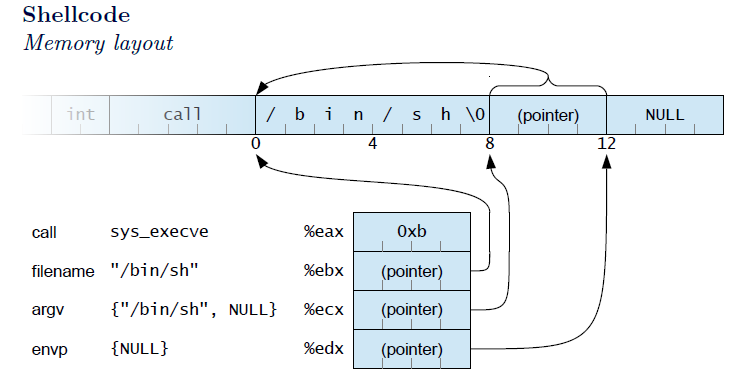
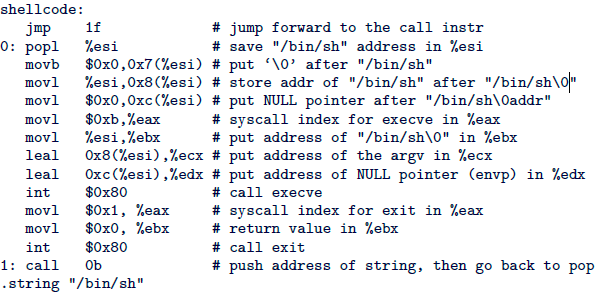
Functions: gets() strcpy(), strcat(), 







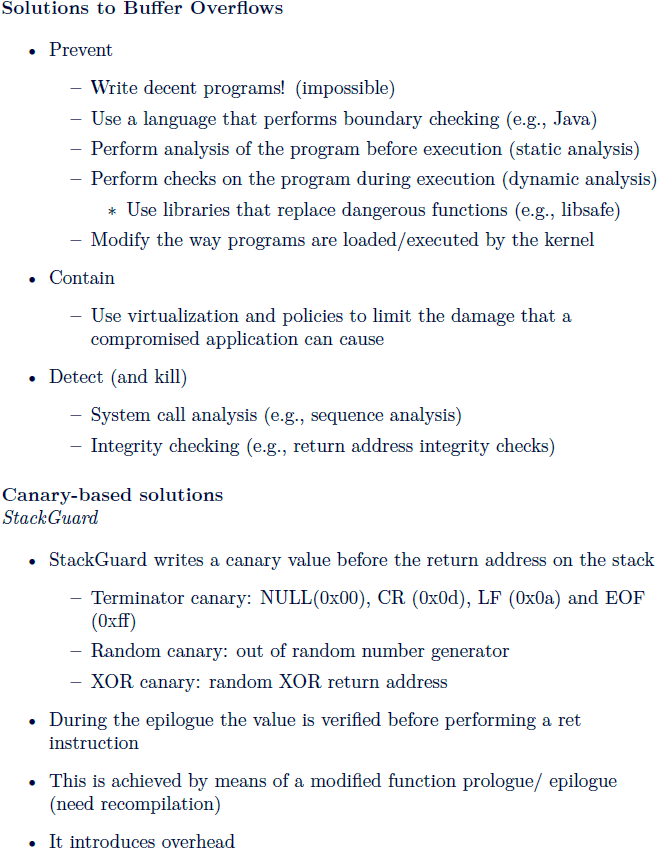
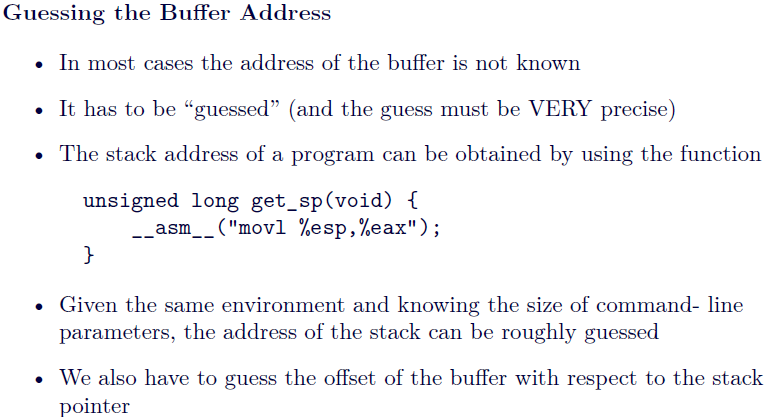
Requires NULL pointer to end the string (reuse null pointer at end of argv)

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

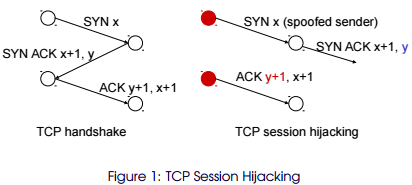
Value of 1 = XOR, then INC



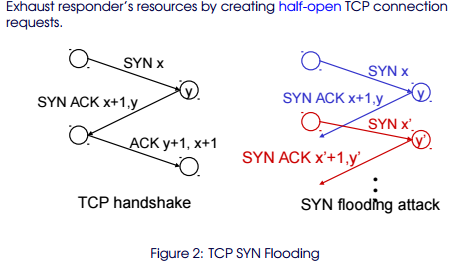
**Network Security**

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

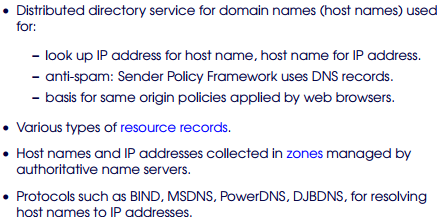
**TCP Session Hijacking**

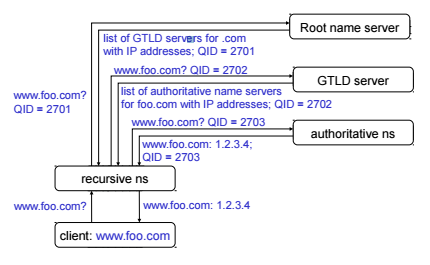
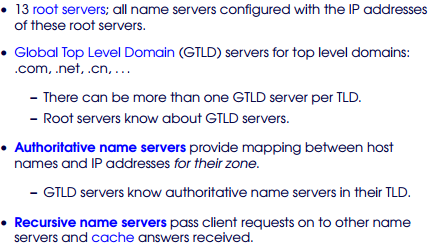


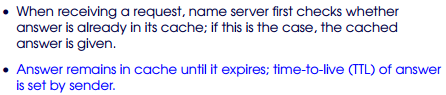
**TCP SYN Flooding Atk**



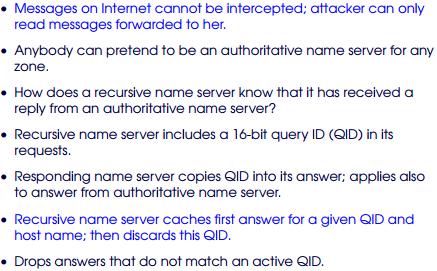
**DNS**





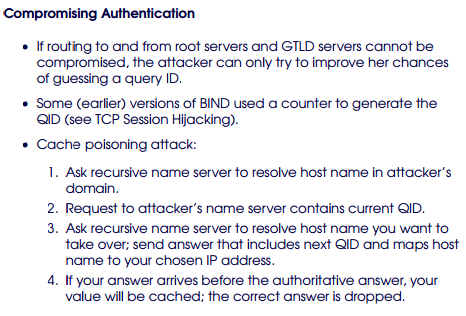


**Lightweight authen**

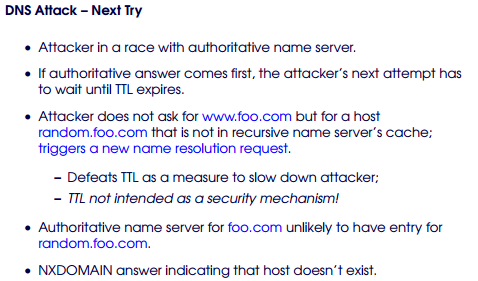
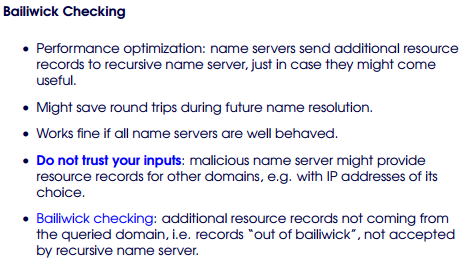
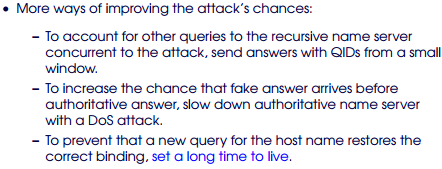


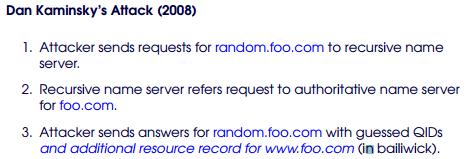
**Security? 1 in 2^16**

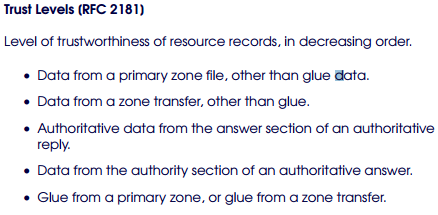
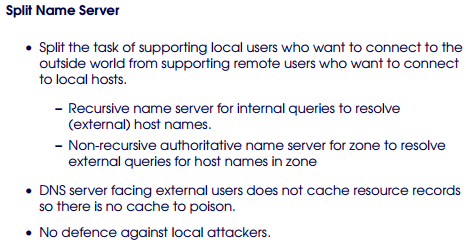
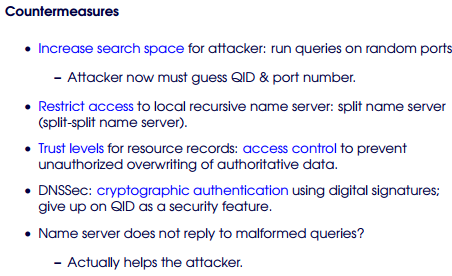
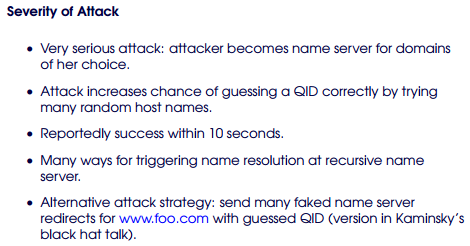
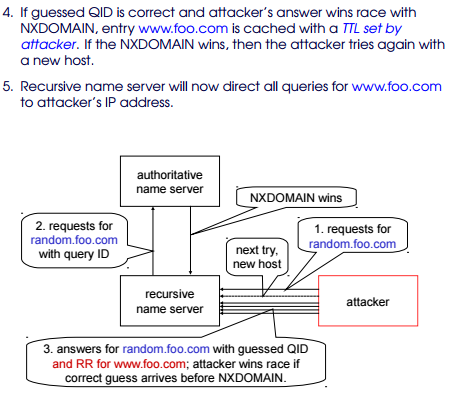
* Relies on assumption routing from local recursive NS to authorative NS =correct



**Don’t use predictable challenges.**

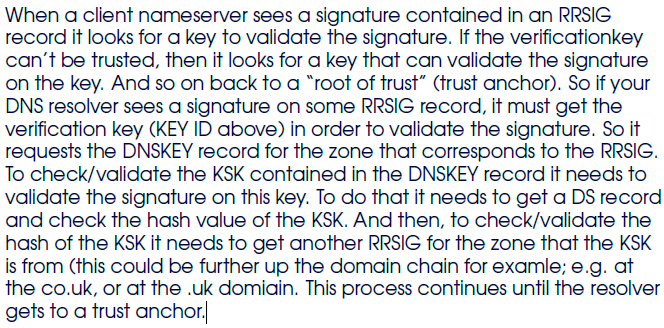
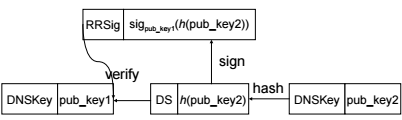


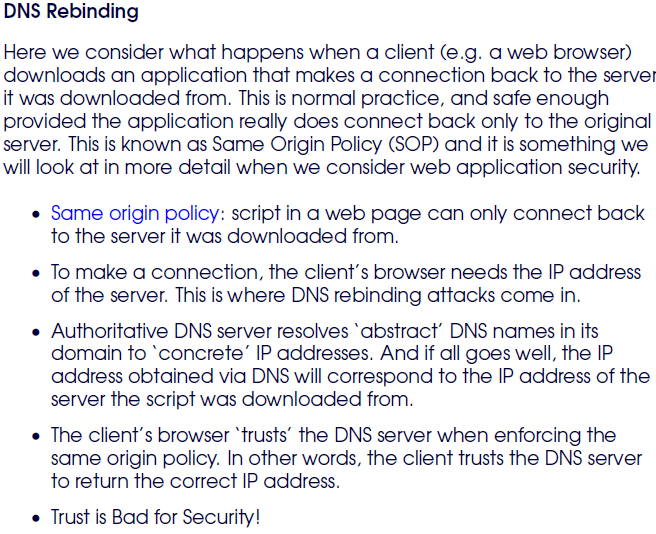


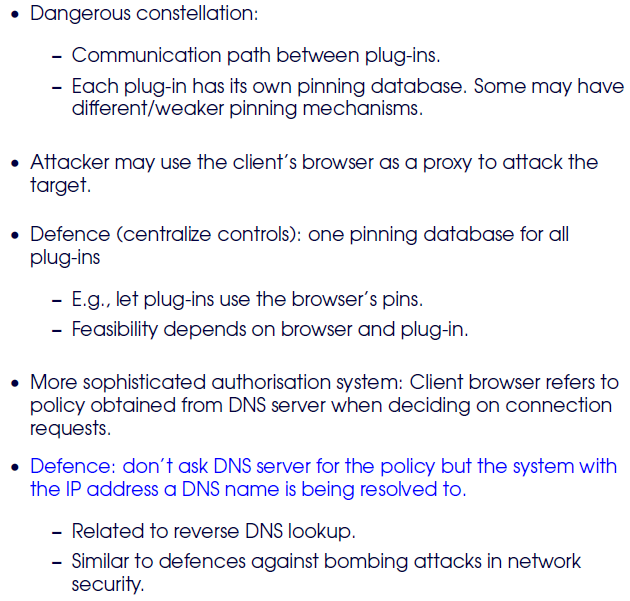
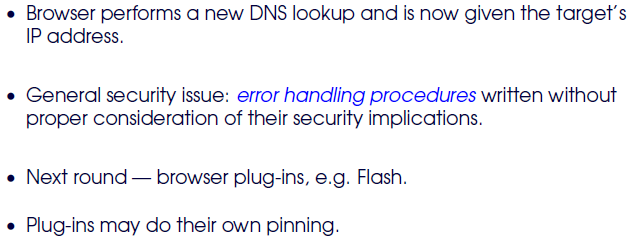
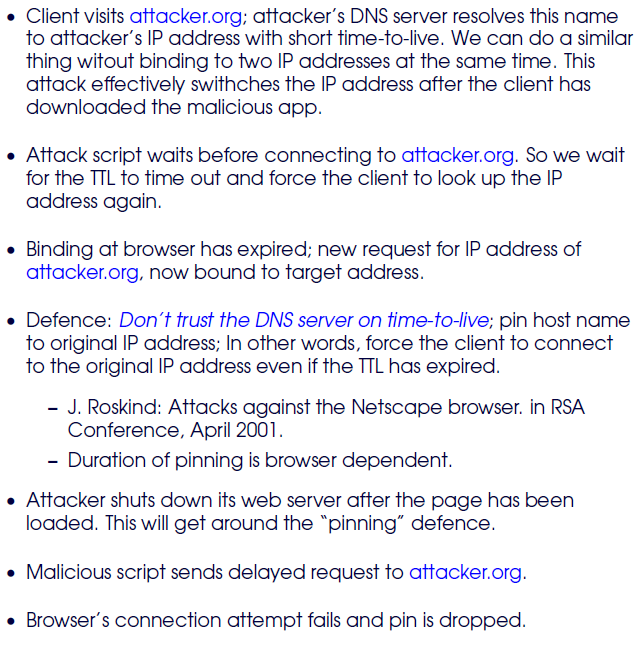


DNS SEC

* 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

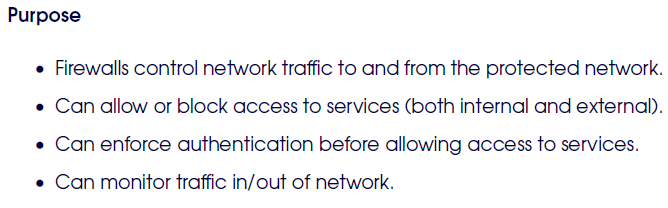
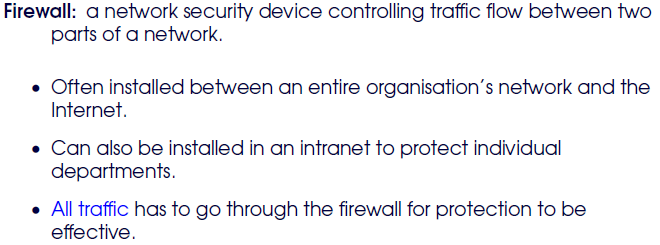


**DNS REBINDING ATTACK**

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

**Firewall**

* **Ingress filtering, egress filtering**
* **ACL based on addr, port, num**

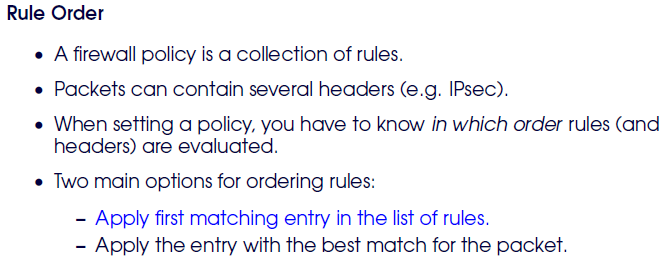
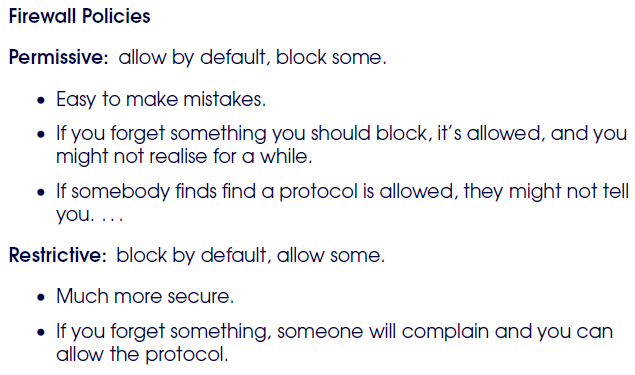
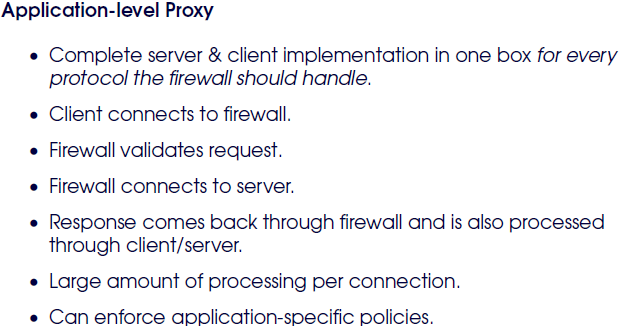


**Packet Filter Firewall ( inspect header, use rules)**

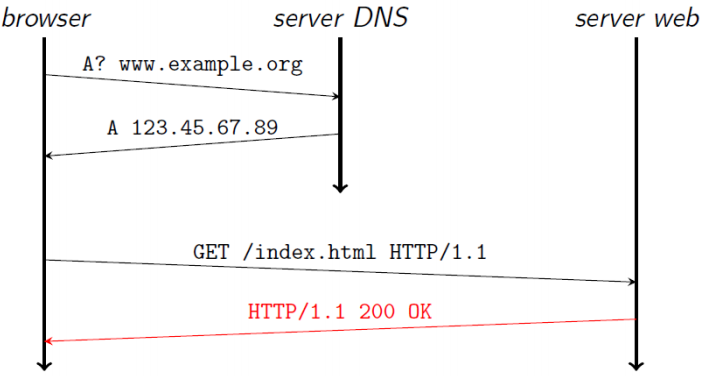
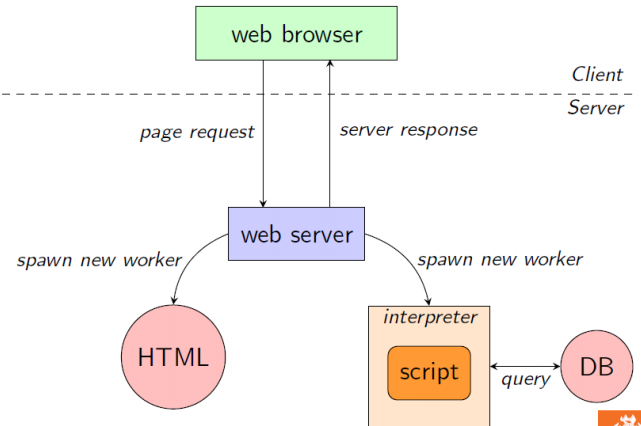
**Stateful packet firewall**

**Circuit level firewall (not used widely) (gateway using IPsec in tunnel mode)**

**App level proxy**

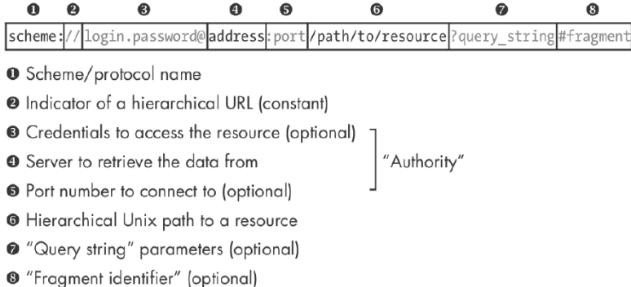


**Web App Sec**

****

**GET, POST**

**URI structure**

****

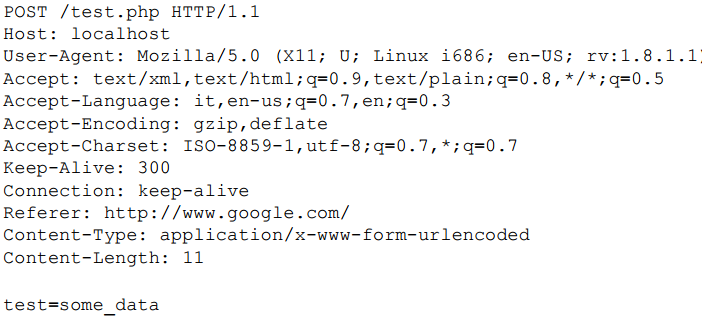
**Attack:** create Hostname that contains a character that looks like a slash

**Defense:** block dangerous characters, display to the user where the browser splits hostname from URI

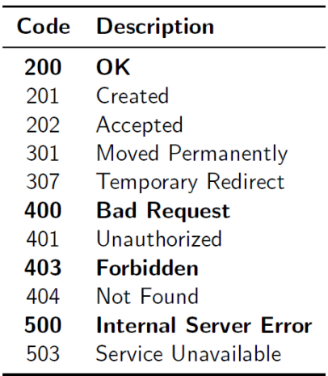
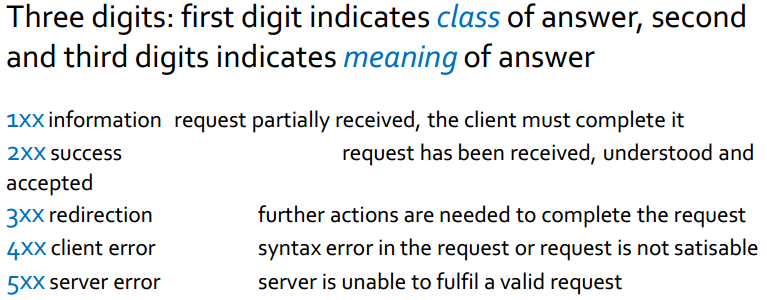
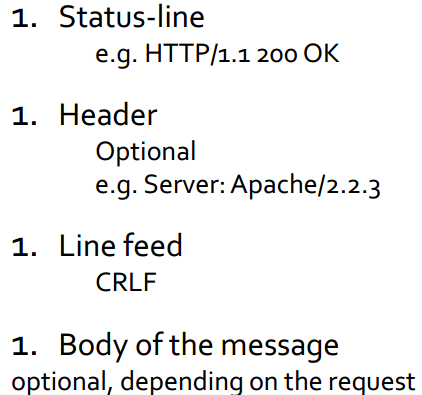
**Web adversary:**  malicious end system, network = “secure”

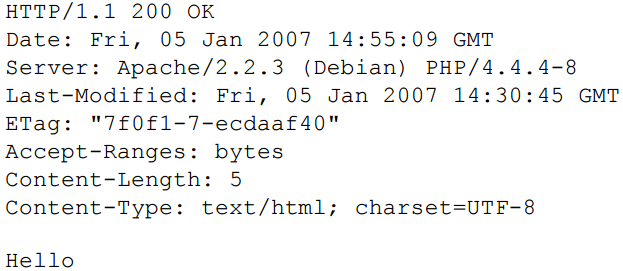
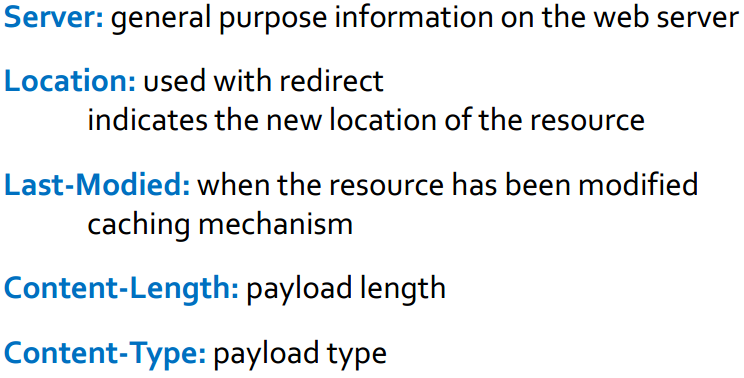
**HTTP Req:**

1. Request line (GET<method /index.html<resource HTTP.1.1<version)  
   proxy = must use absolute URIs
2. Main Req Header (host(mandatory) authorization(cred) if-modified-since (cache) referrer user-agent content-length content-type)
3. Empty line
4. Message body

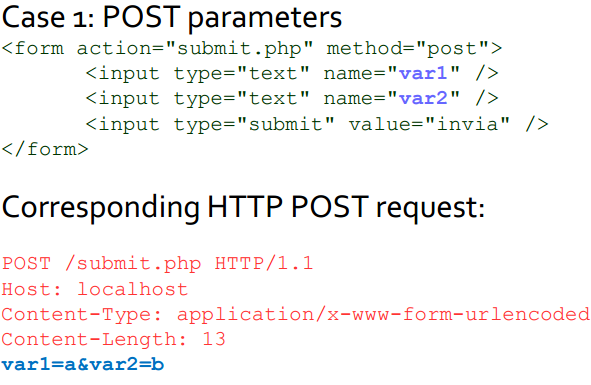
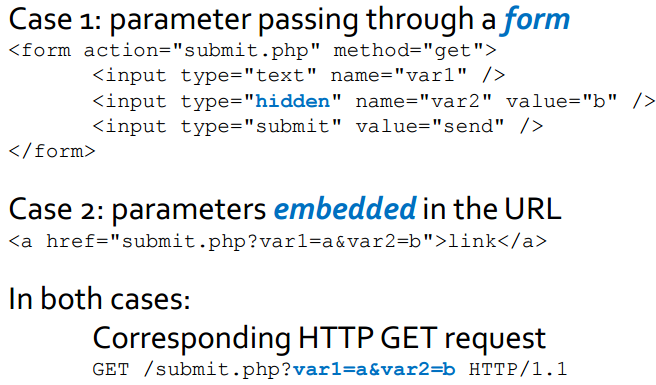


**HTTP Reply**

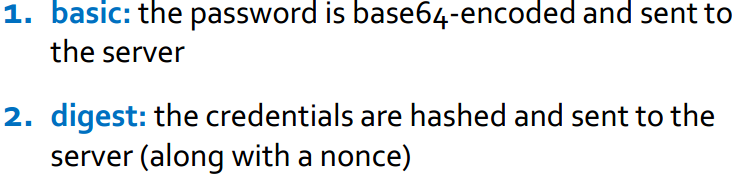
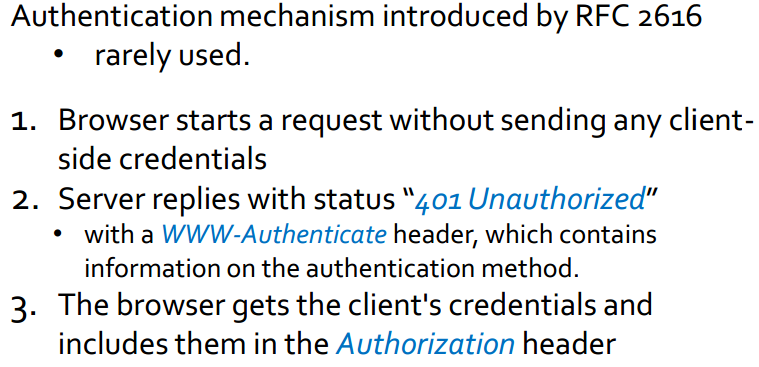




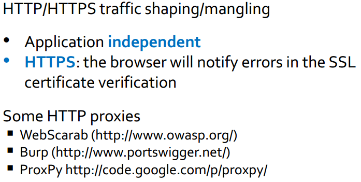
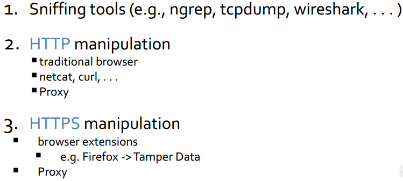
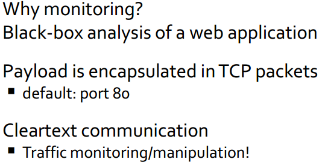
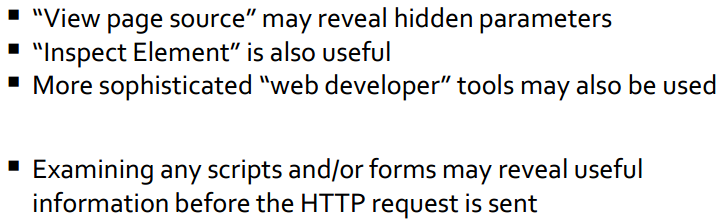
**Parameters**

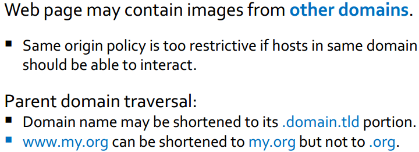
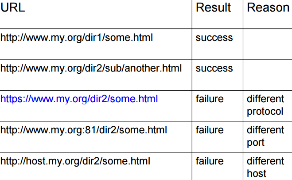
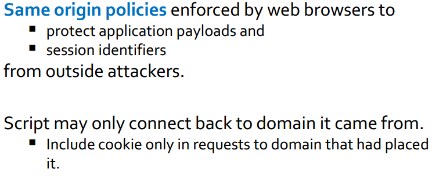
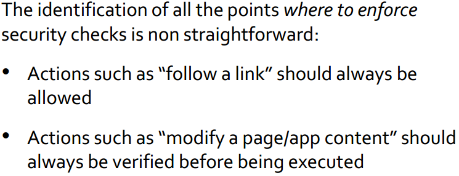
****

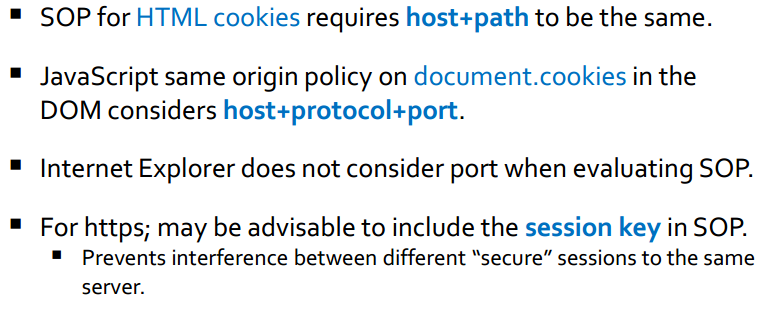
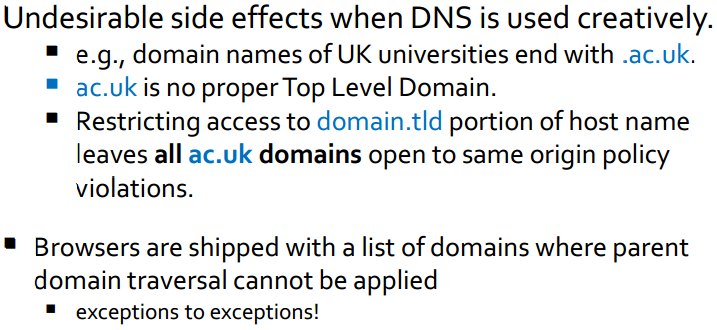
**HTTP Authentication**

****

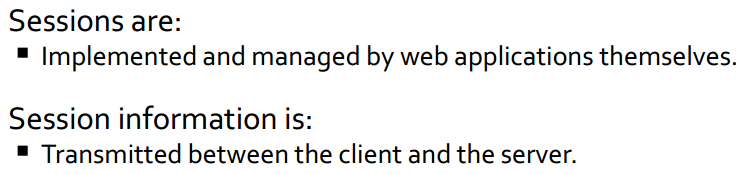
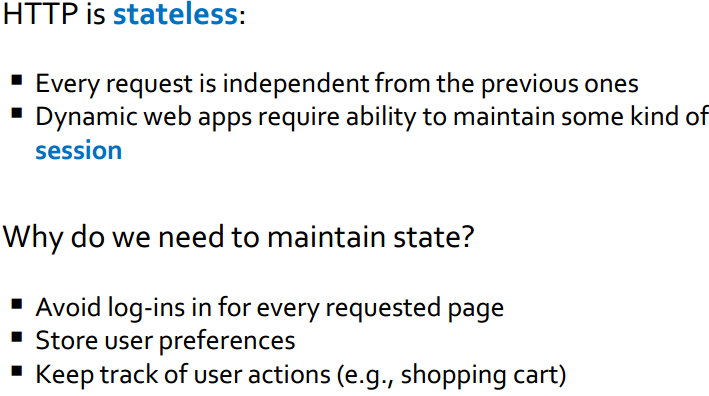
**HTTP Analysis**

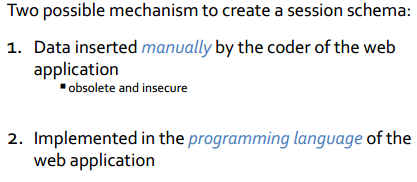
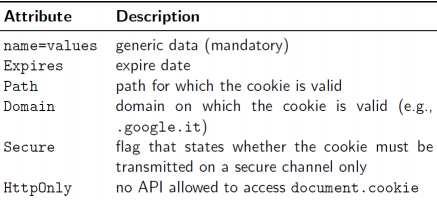
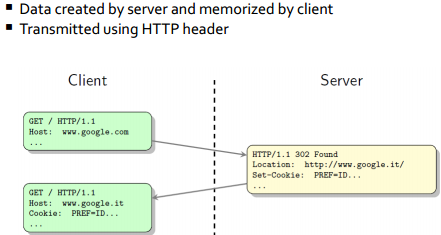
**  
Content Isolation Same Origin**

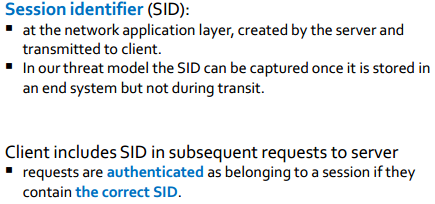
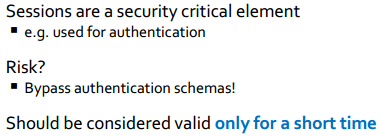
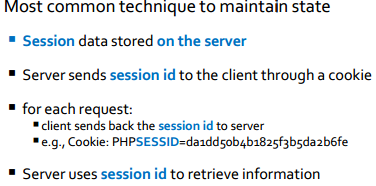
****

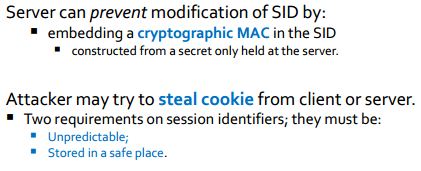
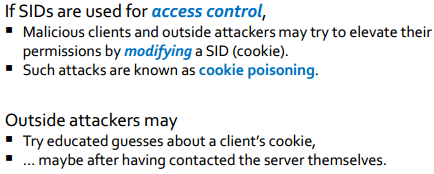
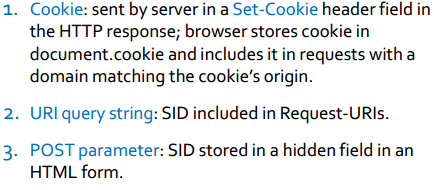
****

**HTTP Session Cookies**

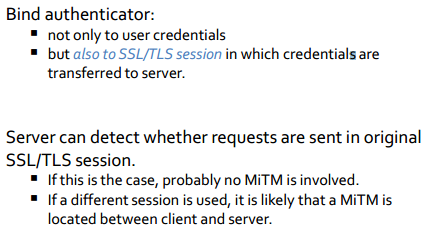
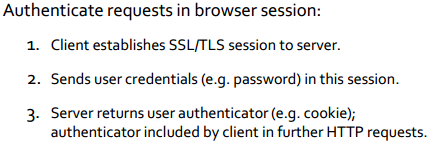
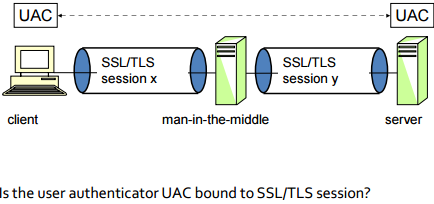
****

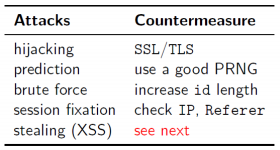
****

****

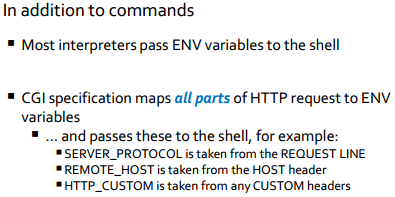
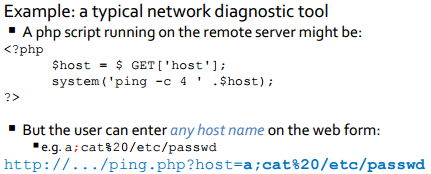
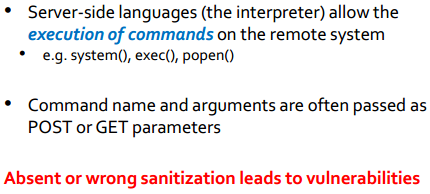
****

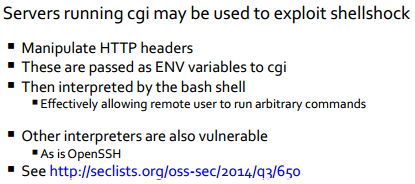
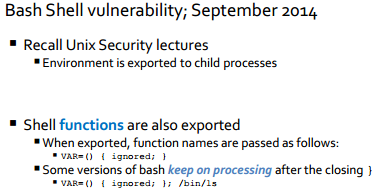
**MITM attack**

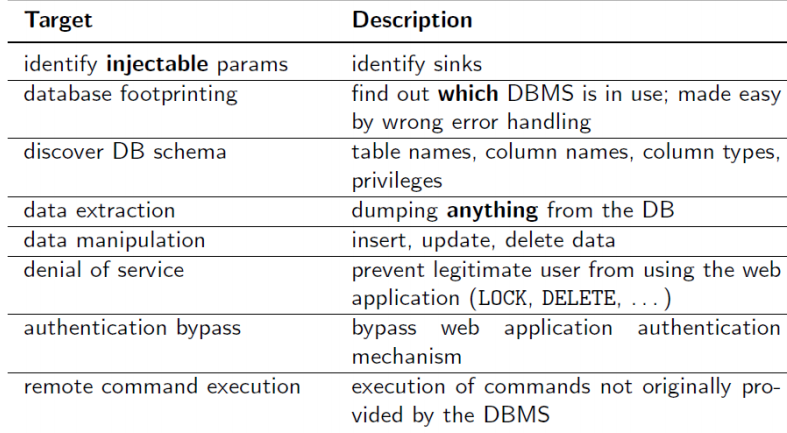
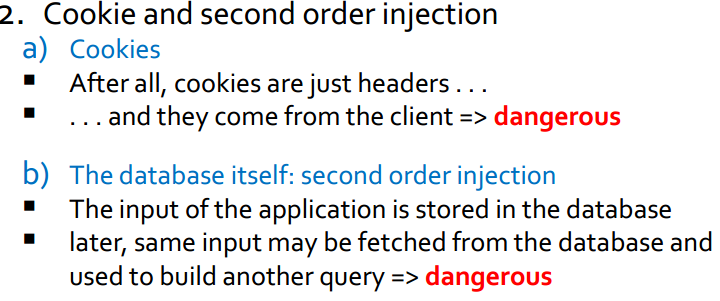
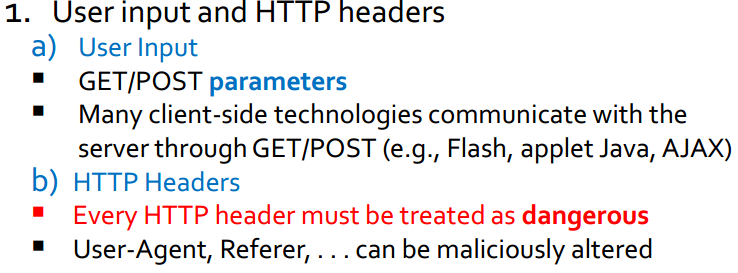
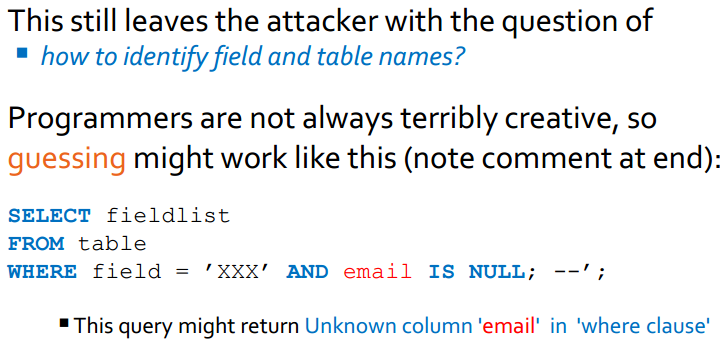
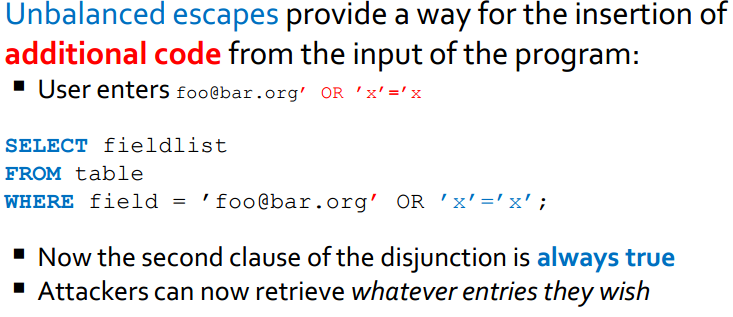
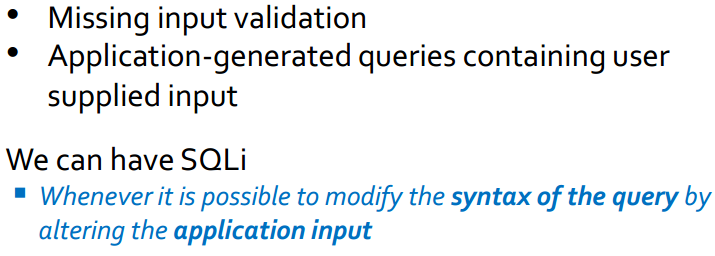
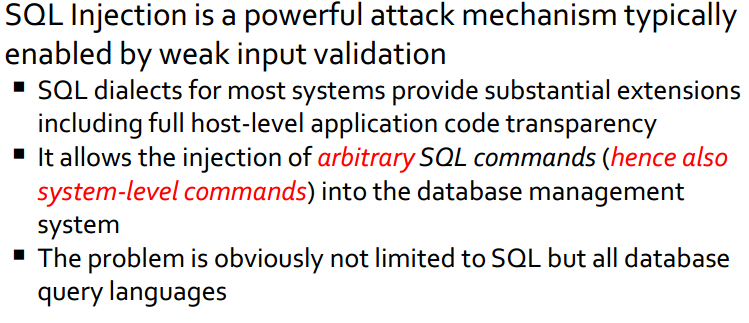
****

****

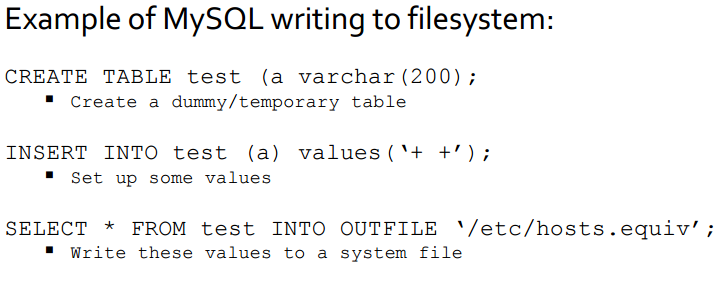
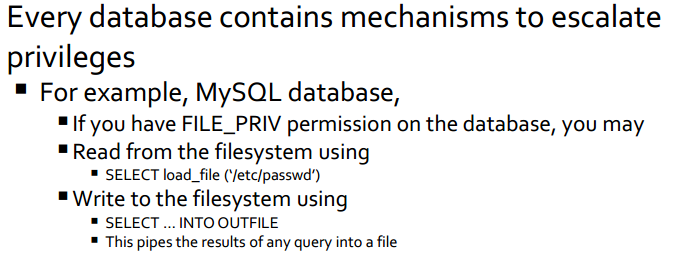
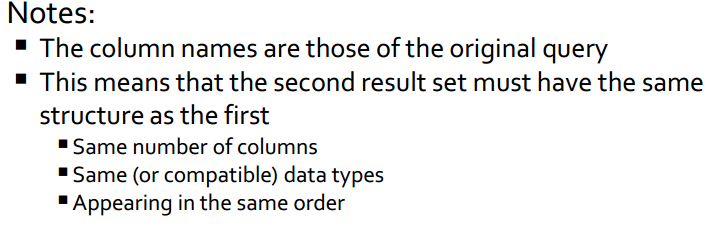
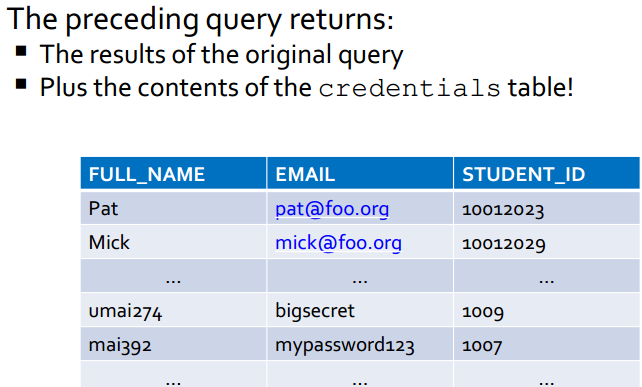
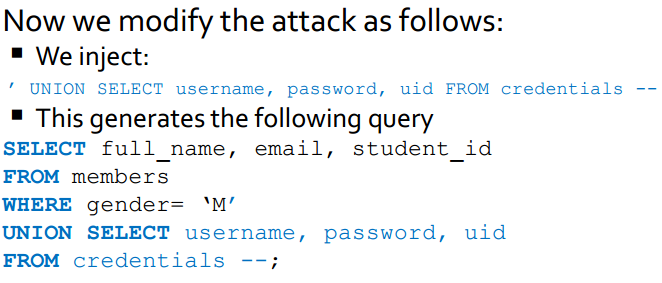
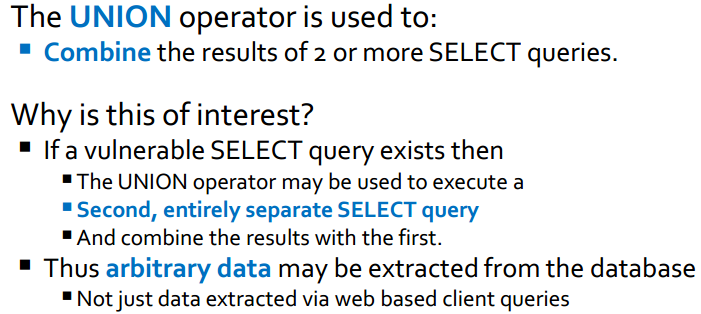
**Command injection**

****

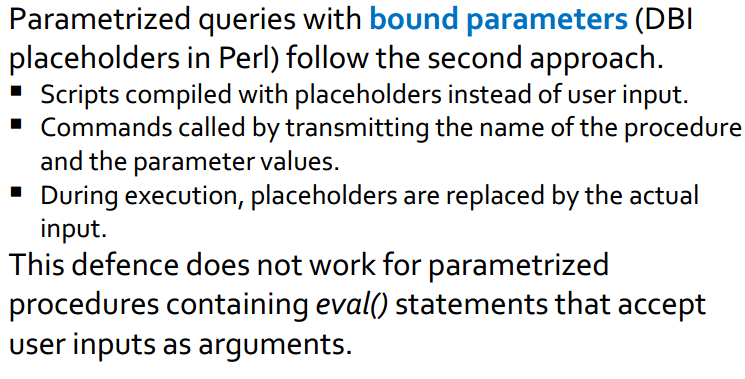
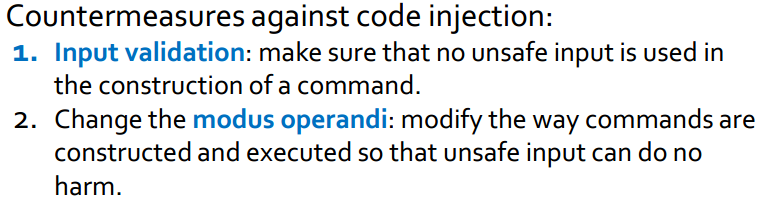
****

**SQL injection  
\**

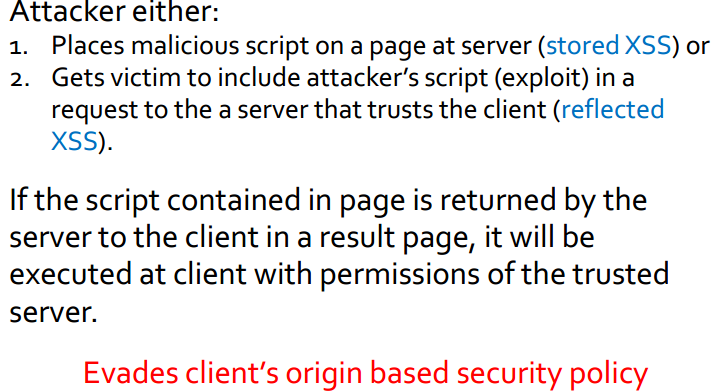
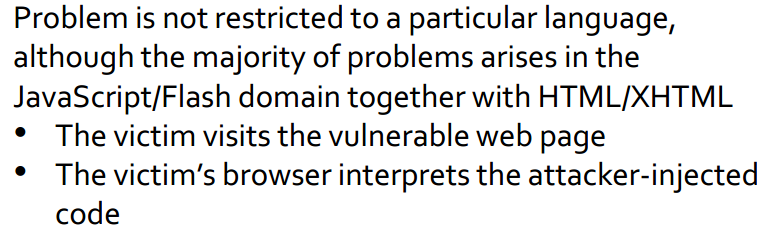
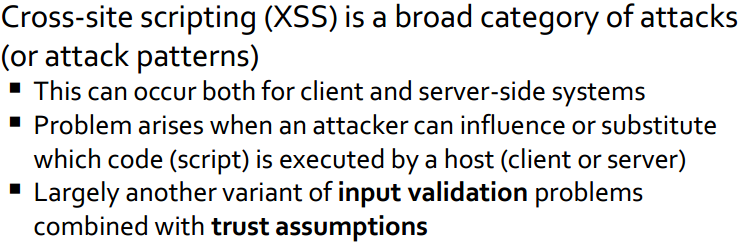
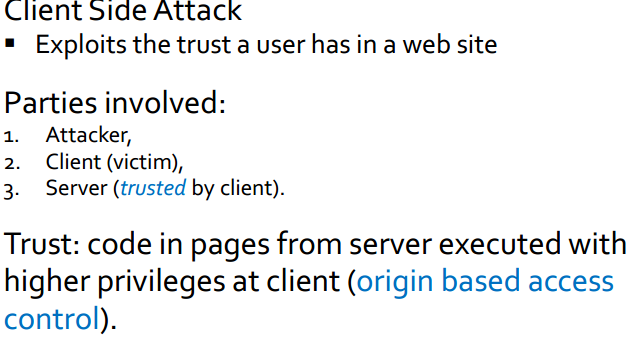
**Advanced SQLi**

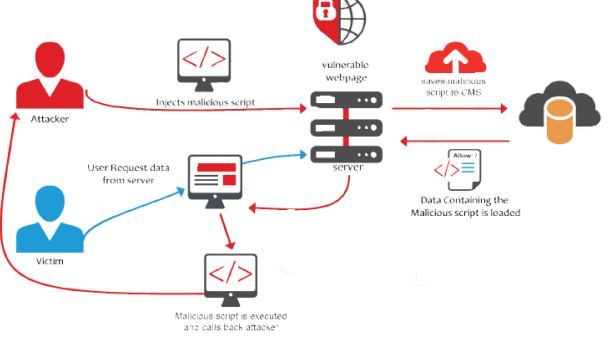
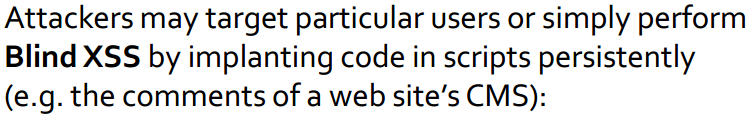
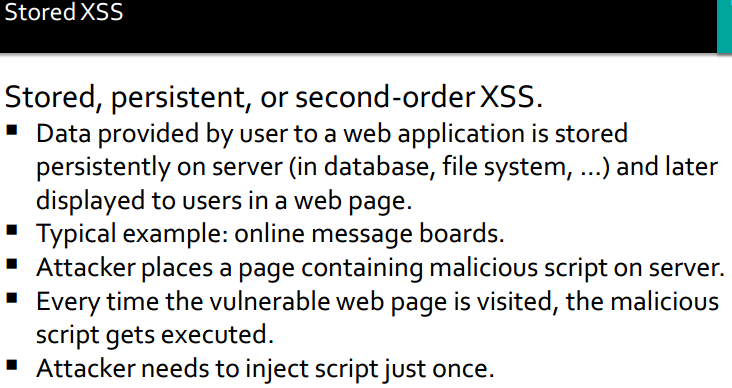
****

**Counter measures**

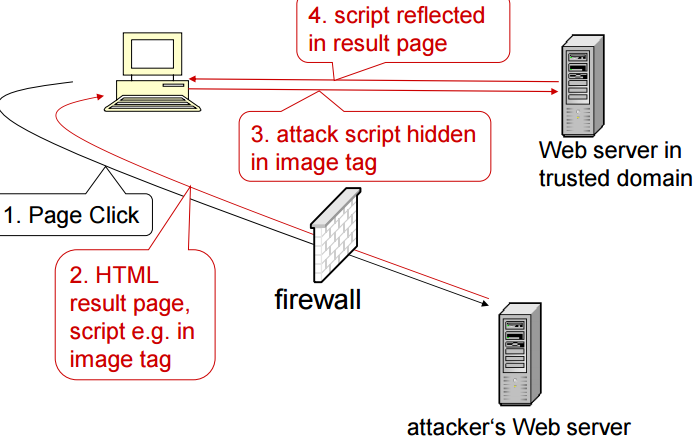
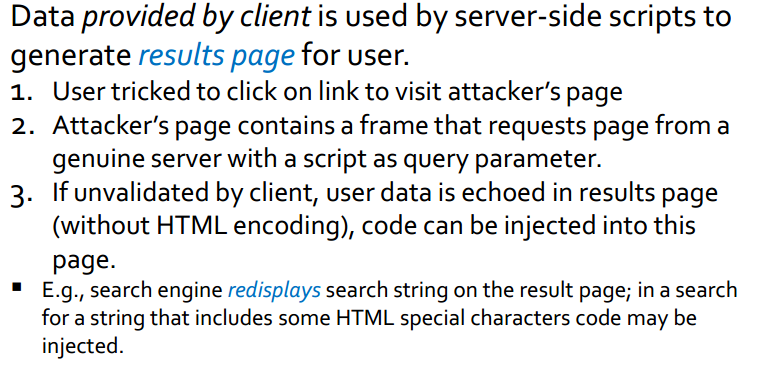
****

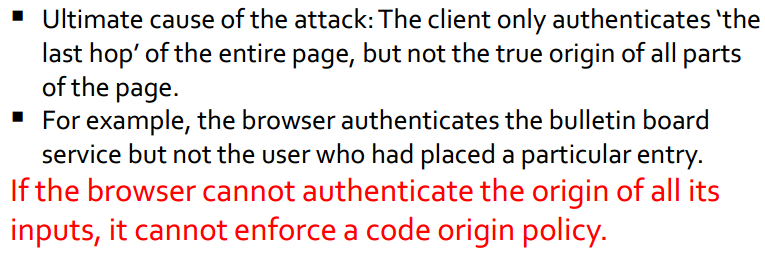
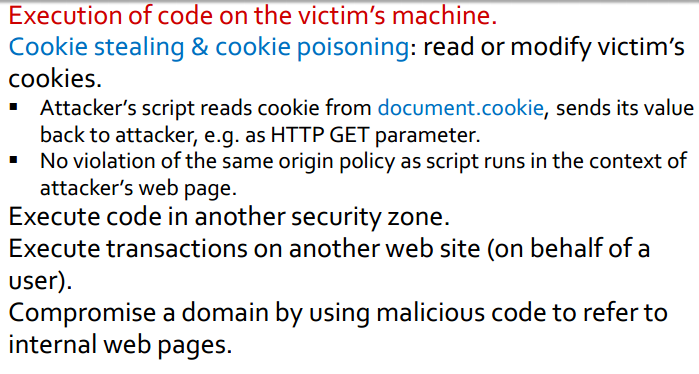
**XSS**

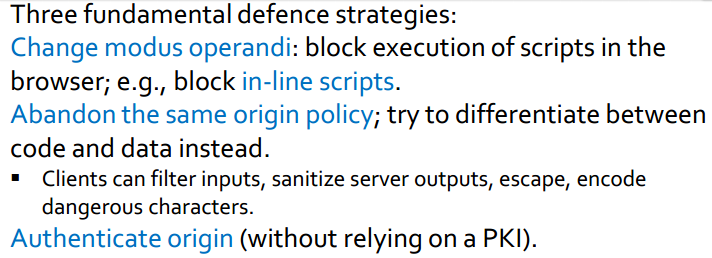
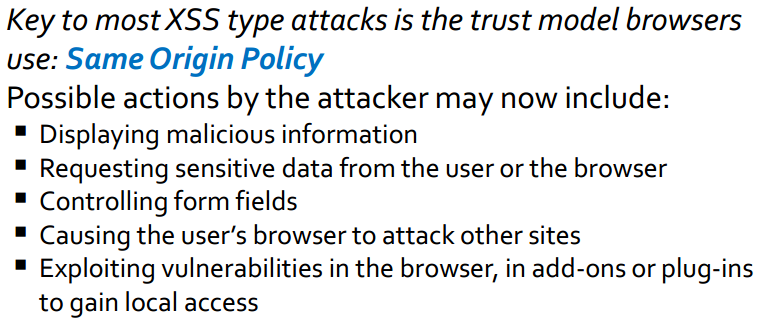
****

****

**Reflected XSS**

****

****

****

**SOP + input validation = XSS**