Security

University of Cambridge

Ashwin Ahuja

Computer Science Tripos Part IB
Paper 4

May 2019

Contents

1 Introduction **3**
1.1 Security Policy Development in Organisations . . . 4
1.1.1 Human Factors in Security Management . . . 5

2 Access Control **5**
2.1 Access Control Matrix . . . 5
2.2 Discretionary Access Control . . . 6
2.3 Mandatory Access Control . . . 6
2.4 Unix / POSIX Access Control . . . 6
2.4.1 User / Group IDs . . . 6
2.4.2 Process Membership . . . 6
2.4.3 File and Directory Permissions . . . 7
2.4.4 Controlled Invocation / Elevated Rights . . . 7
2.5 Linux Capabilities . . . 8
2.5.1 Examples . . . 8
2.6 POSIX.1e Access Control Lists . . . 8
2.6.1 Compatibility Mapping . . . 9
2.6.2 Design . . . 9
2.6.3 Access Check Algorithm . . . 9
2.7 Windows Access Control . . . 9
2.7.1 Security Descriptor . . . 10
2.7.2 Algorithm . . . 10
2.7.3 Inheritance . . . 11
2.7.4 Auditing . . . 11
2.7.5 Defaults . . . 11
2.7.6 Service . . . 11
2.7.7 LDAP . . . 11
2.8 Mandatory Access Control . . . 12
2.8.1 Bell-LaPadula Model . . . 12
2.8.2 Lattice Model . . . 13
2.8.3 Covert Channel Problem . . . 13
2.8.4 Integrity Models . . . 14
2.9 Commercial Systems . . . 14
2.9.1 Clark / Wilson Framework . . . 14
2.10 Operating System Extensibility for Access Control . . . 14
2.11 Linux Security Modules . . . 15
2.11.1 AppArmor . . . 15
2.11.2 SELinux . . . 15
2.11.3 Android Access Control . . . 16
2.11.4 chroot jails . . . 16

3 Operating-System Security **16**
3.1 Classification . . . 17

4 Software Security **17**
4.1 Malicious Software . . . 17
4.2 Common Vulnerabilities . . . 18
4.3 Buffer Overflows . . . 19
4.4 Missing check of input data - shell meta-characters . . . 19
4.5 Sourcing Secure Randoms . . . 20
4.6 Security Testing . . . 20
4.6.1 Penetration Testing / Flaw Hypothesis Method . . . 20
4.6.2 Fuzz Testing . . . 20

5 Cryptography **20**
5.1 Basic Primitives 20
5.1.1 Standards 21
5.2 Message Authentication Code 21
5.3 Private-Key Encryption Scheme 21
5.4 Authenticated Encryption Scheme 21
5.5 Summary 21
5.5.1 Key Exchange 21
5.5.2 Digital Signature 21

6 Entity Authentication **22**
6.1 Passwords 22
6.2 Authentication Protocols 22
6.2.1 Properties to consider 22
6.2.2 Notation 23
6.2.3 One-pass Authentication 23
6.2.4 Two-pass Authentication 23
6.2.5 Mutual Authentication 24
6.3 Kerberos 24

7 Network Security **24**
7.1 TCP/IP Security 25
7.1.1 Address Spoofing 25
7.1.2 Vulnerabilities 25
7.2 Firewalls 25
7.3 Virtual Private Networks 26
7.3.1 Site-to-Site VPN 26
7.3.2 Remote Access VPN 26
7.4 Distributed Denial of Service Attack 26
7.5 Hyper Text Transfer Protocol 26
7.5.1 Version 0.9 26
7.5.2 Version 1.0 27
7.5.3 Version 1.1 27
7.5.4 Request Headers 27
7.5.5 State and Session Context 27
7.6 Redirects 27
7.7 Basic Authentication 27
7.8 Form Methods 27
7.8.1 GET 27
7.8.2 POST 28
7.9 Cookies 28
7.9.1 Session Cookies 28
7.10 Cross-Site Request Forgery 29
7.11 Website Compromise Solutions 29

1 Introduction

Computer Security: discipline of managing malicious intent and behaviour involving information and communications technology
Security Engineering focuses on intentional rather than accidental behaviour (as in **Safety Engineering**) with the presence of an intelligent adversary

Malicious Behaviour examples:

1. **Fraud**: unauthorised access to money, goods or services
2. **Vandalism**: causing damage for personal reasons - frustration, envy, revenge, curiosity, self esteem, peer recognition
3. **Terrorism**: causing damage, disruption and fear to intimidate
4. **Warfare**: damaging military assets to overthrow a government
5. **Espionage**: stealing information to gain competitive advantage
6. **Sabotage**: causing damage to gain competitive advantage - DOS attack and bot services
7. **Spam**: unsolicited marketing wasting time / resources
8. **Illegal Content**: child sexual abuse images, copyright infringement, etc

Examples of where Information Security is a specific concern:

1. Business environment
2. Military environment
3. Medical environment
4. Households
5. Society at large - utility services, communications

Security Threats $\rightarrow$ Security Policies $\rightarrow$ Security Controls

Information Security Targets

1. **Confidentiality**: ensuring that information is only accessible to authorised people

   Comes in a number of variants:

   (a) Data protection / personal data privacy - fair collection and use of personal data in Europe
   (b) Anonymity / traceability - ability to use a resource without disclosing identity / location - Tor
   (c) Unlinkability - ability to use resource multiple times without others being able to link these uses together - cookies in HTTP for example
   (d) Pseudonymity - anonymity with accountability for actions
   (e) Unobservability - ability to use a resource without revealing this activity to third parties - information hiding for example - example given in lecture notes is bank notes
   (f) Copy protection, information flow control - ability to control the use and flow of information

2. **Integrity**: safeguarding the accuracy and completeness of information and processing methods
3. **Availability**: ensuring that authorised users have access to information and associated assets when required

   Integrity and Availability Protection

   - Rollback: ability to return to a well-defined valid earlier state ($\rightarrow$ backup, revision control, undo function)
   - Authenticity: verification of the claimed identity of a communication partner
   - Non-repudiation: origin and / or reception of message cannot be denied in front of a third party

- Audit: monitoring and recording of user-initiated events to detect and deter security violations
- Intrusion detection: automatically notifying unusual events

Optimistic Security: Temporary violations of security policy may be tolerable where correcting the situation is easy and the violator is accountable - generally applied to integrity and availability. An example of using this is Wikipedia.

Threat Scenarios

1. **Data in Transit**: there is an eavesdropper (Eve) (passive) or someone in the middle (Mallory) (middle person attack) - allows them to both read from and write to the line.
2. **Data at Rest**: attacks a storage device - can be with just read access (Eve) or write access to the data as well (Mallory)
3. **Data in Use**: utilises side channels or fault injections to work out what is happening. For example, heat production or power usage of systems.

Questions of security mechanism:

1. What does security mechanism achieve
2. Do we need confidentiality, integrity, availability and of what?
3. Who will generate / store / have access to keys? Can we lose keys?
4. Will it interfere with other security measures - for example Encryption would be unhelpful for backups
5. Will it introduce new vulnerabilities or can it be used against it?
6. New incentives?
7. What if it breaks

1.1 Security Policy Development in Organisations

1. **Security Requirements Analysis**: (1) identify assets and their value, (2) identify vulnerabilities, threats and risk priorities, (3) identify legal and contractual requirements
2. **Work out suitable security policy**:

   Security Policy: Abstraction of the security requirements - a set of rules that clarifies which are not authorised, required and prohibited activities, states and information flows. **Security Policy Models** are techniques for the precise and formal definition of the protection goals - describe both (1) automatically enforced policies (in an OS for example), (2) a procedure for employees
3. **Security Policy Document**: High level security policy documented as a reference for anyone involved in implementing controls when a good understanding exists of what exactly security means for organisation that needs to be protected or enforced. Lay out (1) overall objectives, (2) principles and (3) underlying threat model to guide choice of mechanisms.
4. **Selection and implementation of controls**: addressing issues in a typical low-level organisational security policies
   - General and specific responsibilities for security
   - Names manager (owns overall policy) and is in charge of (1) continued enforcement, (2) maintenance, (3) review and (4) evaluation of effectiveness
   - Names individual managers who own individual information assets and are responsible for day-to-day security
   - Reporting responsibilities for incidents, vulnerabilities, software malfunctions
   - Mechanisms for learning from incidents

1.1.1 Human Factors in Security Management

Security with respect to dealing with people:

1. Incentives, discipline
2. Training and documentation
3. Background checks
4. Regulation of third-party (suppliers, for example) access - for example, in a satellite hacking system, the people who have the computers (pass non-working computers to nearby random reseller) which have the software to decrypt on them, have the power to reveal all the information.
5. Physical security in office buildings
6. Segregation of duties - reduce effect if people going rogue
7. Audit trails
8. Separation of development and operational facilities
9. Protection against unauthorised and malicious software
10. Document access control, including sensitivity labelling of documents and media
11. Disposal of media
12. Network and software configuration management
13. Duress alarms, terminal timeouts, clock synchronisation
14. Line and file encryption

In general, human users have become the largest security vulnerability as the technical security improves - important to be aware that ancient fraud techniques still work well - the Monty where you are misled that the game is fair by the use of an accomplice who over-bets if you get the correct answer

- Workers are goal oriented - will intuitively take the minimal action required to complete a task - need to be aware of these trusted paths. For example, Ctrl-Alt-Del hands control back to the OS to ensure it is the real login system not a fraudulent application
- Users assess cost / benefit of security measures - for example if the password screen is already open, then they won't bother pressing Ctrl-Alt-Del - need to carefully manage the Compliance Budget
- Must also communicate with users in order to produce usable systems

2 Access Control

2.1 Access Control Matrix

$$M = (M_{so})_{s\in S, o\in O} \ \ with \ \ M_{so} \subset A$$

and

- S = set of subjects
- O = set of objects
- A = set of access privileges

Columns are stored with objects(**access control list**) and rows are stored with subjects (**capabilities**). It is rarely implemented in practise for a number of reasons: (1) scalability, (2) redundancy - many subjects share same capabilities and many objects share the same Access Control Lists, (3) usability - difficult to review and maintain by humans.

Can group equivalent subjects into domains and objects into types - leading to **type enforcement matrices**. Can also distinguish between users (grouped into groups or roles and assigns capabilities to roles) and processes.

Can also utilise existing hierarchical structures - to make grouping subjects or objects more conveniently:

- **Hierarchical File System** - subdirectory paths
- **Process Ancestry Trees** - parent-process relation

2.2 Discretionary Access Control

Access to objects is permitted based on user identity where each object is owned by a user. Owners can specify freely how they want to share their objects with other users, specifying which other users can have which form of access to their objects - implemented in multi-user OSs.

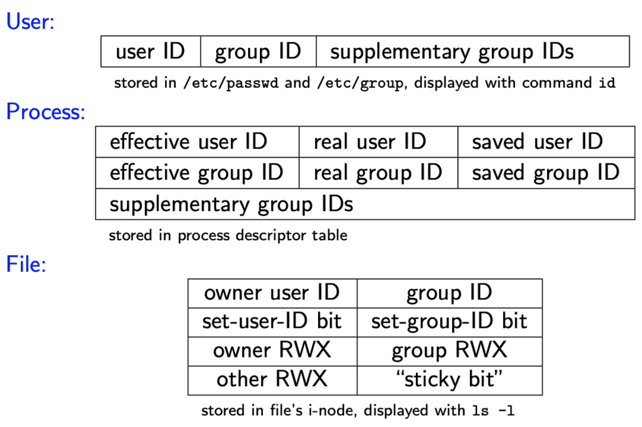
2.3 Mandatory Access Control

Access to objects controlled by a system-wide policy, for example to prevent certain flows of information. System maintains security labels for both objects and subjects based on which access is granted or denied. Labels can change as the result of an access. Security policies enforced without cooperation of users or application programs. Originated in special OS versions for military users - now used widely on mobile devices to sandbox apps.

2.4 Unix / POSIX Access Control

- Small state space -> very easy to display, read and query
- No standardised access-control list - access only restricted to single groups - may need to create a large number of groups
- Small number of user / group identifiers (originally 16-bit integers) -> renumbering is needed when organisations merge

2.4.1 User / Group IDs



User:

user ID	group ID	supplementary group IDs

stored in /etc/passwd and /etc/group, displayed with command id

Process:

effective user ID	real user ID	saved user ID
effective group ID	real group ID	saved group ID
supplementary group IDs		

stored in process descriptor table

File:

owner user ID	group ID
set-user-ID bit	set-group-ID bit
owner RWX	group RWX
other RWX	"sticky bit"

stored in file's i-node, displayed with ls -l

- User identified by a 32-bit integer value
- User belongs to at least one primary group (defined via passwd database) and an optional set of supplementary groups (defined via group database), each of which is identified by a 32-bit integer value
- Kernel only deals with integer user IDs and group IDs
- C library provides support for human-readable names via (1) passwd database query interface that maps between user IDs and user names: getpwnam(3) and getpwuid(3) and (2) a group database query interface that maps between integer group IDs and group names: getgrnam(3), getgrgid(3). *The config file (/etc/nsswitch.conf) defines where these databases reside (/etc/passwd and /etc/group)*
- User and group names occupy separate name spaces, user and group IDs occupy seperate number spaces - same name or ID number can refer to both a user and a group

2.4.2 Process Membership

- **Login**: During login process, look up in passwd and group databases the uid, primary gid and supplementary gid associated with login name and inherits these to the shell process.
- Process started by a user inherits the user and group IDs from the shell process
- Processes store three variants of user ID and group ID: **effective - identity that determines the access rights**, **real - identity of the calling user** and **saved - effective identity when the program was started** - only effective ones are consulted during the file access period
- Root has user id 0 - having full access - and can change its own user IDs and group IDs

2.4.3 File and Directory Permissions

File Permissions: Each file (**inode**) carries three integers: (1) user ID - file owner, (2) group ID, (3) 16 mode bits that define access permissions - contains nine permission bits, which are interpreted in groups of 3:

```
S_IRUSR  00400  owner has read permission
S_IWUSR  00200  owner has write permission
S_IXUSR  00100  owner has execute permission

S_IRGRP  00040  group has read permission
S_IWGRP  00020  group has write permission
S_IXGRP  00010  group has execute permission

S_IROTH  00004  others have read permission
S_IWOTH  00002  others have write permission
S_IXOTH  00001  others have execute permission
```

When a process attempts to access a file, the kernel first decides which of the user class the accessing process falls into - for each class, there are three mode bits, controlling read, write and execute access by that class.

Peripheral devices (and interprocess-communication endpoints - Unix domain sockets, names pipes) are represented by special files with similar access control

Directory Permissions: r - required for reading the names of files contained, w - required to change the list of filenames in the directory (create, delete, rename or move file in it), x - required to access contents or attributes of a file in it (directory traversal), sticky bit (added by Berkeley Unix)

- If set for a writable directory D, then file F in D can be removed or renamed only by the owner of F or the owner of D

Changing Permissions: Only owner of file or directory can change its permission bits - can only change gid to group of which the owner is a member. In order to change the owner of a file or directory, only the root can do this.

New Files: Newly created file has permissions 0666 (rw-rw-rw-) and ¬umask where umask is a field in the process descriptor table that cn be used to disable default permissions of newly created files. The file inherits the **group of the creating process** - if setgid bit is set to directory then (1) any file or directory created in it inherits the group of that directory and (2) any directory created in it inherits the setgid bit

2.4.4 Controlled Invocation / Elevated Rights

Many programs need access rights to files beyond those of the user:

- passwd program allows user to change password and therefore needs write access to /etc/passwd
- Additional permission bits
  1. **set-user-ID**: file owner ID determines process permissions
  2. **set-group-ID**: file group ID determines process permissions
- Normal process started by user U will have the same value U stored as the effective, real and saved user ID and cannot change any of them
- When program file owned by user O and with the set-user-ID bit set is started by user U, then both the effective and the saved user ID of the process will be set to O, whereas real user ID will be set to U. The program can now switch the effective user ID between U (copied from the real user id) and O (copied from the saved user id)
- Set-group-ID bit on program file causes effective and saved group ID of the process to be the group ID of the file and real group ID remains that of the calling user. Effective group ID can then as well be set by the process of any of the values stored in the other two.
- **Proliferation of Root Privileges**: in order to complete any actions for which root is required for, access is needed to be granted which would allow it to do any other action.

2.5 Linux Capabilities

This fixed the proliferations of root privileges by offering more methods of doing 'root' actions outside of the root. Each process is labelled with a set of privileges (under capabilities) implemented as a 64-bit bit mask, which defines which particular privileged operation the process is allowed to execute. Therefore, instead of checking if you are the root user, you check this bit mask. By default, root user has all the actions set to true.

2.5.1 Examples

1. **CAP_CHOWN**: Make arbitrary changes to file UIDs and GIDs
2. **CAP_DAC_OVERRIDE**: Bypass file read, write and execute permission checks
3. **CAP_DAC_READ_SEARCH**: Bypass file read permission checks and directory read and execute permission checks
4. **CAP_FOWNER**: Bypass permission checks on operations that normally require the filesystem UID of the process to match the UID of the file
5. **CAP_KILL**: Bypass permission checks for sending signals
6. **CAP_NET_BIND_SERVICE**: Bind a socket to internet domain privileged ports (with port numbers less than 1024)

Mechanism: Each Linux process p has several capability sets:

1. Permitted P_p - limiting superset of the effective capabilities that the process may assume
2. Effective E_p - used by the kernel to perform permission checks
3. Inheritable I_p - set of capabilities preserved across invocation of another program (with execve)
4. Bounding set X_p - upper limit, inherited, can only ever shrink - can just turn on and off capabilities

Similarly, each executed file f has:

1. Permitted P_f - capabilities automatically permitted when the program is executed
2. Effective E_f - determines if a newly permitted capability from P_f is also added to the effective set
3. Inheritable I_f - inherited capabilities that the program can receive

Therefore, when a process p starts another process p' by executing file f:

1. $P_{p'} := (X_p \cap P_f) \cup (I_p \cap I_f)$
2. $E_{p'} := E_f \cap P_{p'}$
3. $I_{p'} := I_p$

In general, things don't really work this way - it is much more complicated

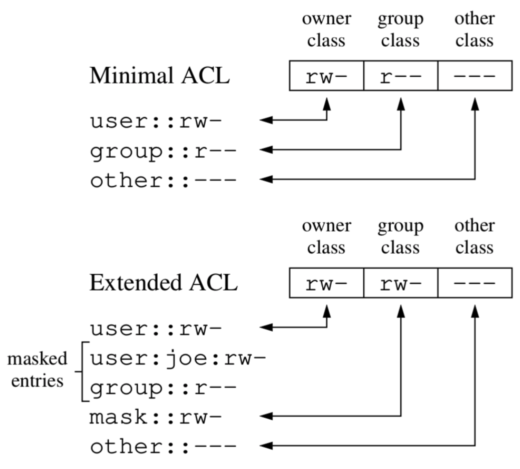
2.6 POSIX.1e Access Control Lists

Consists of a set of entries, of which there are six types:

1. Each file has one (1) owner, (2) owning group and (3) named user entry, which together carry the old nine old mode bits
2. Multiple (4) named group entries can be added to grant access to additional users and groups
3. A single (5) mask entry then limits the permissions that can be granted by the named and owning group entries. If there are named entries, the mask takes over the role of the owning group entry for the old chmod / stat / ls interfaces
4. Is also an (6) others field

2.6.1 Compatibility Mapping

There is a mapping between ACL entries and permission bits:



2.6.2 Design

1. Remain as largely backwards compatible with applications that only understand the nine mode bits
2. **Inheritance**: In addition to access ACL used for access check, directories also have default ACL. It determines the access ACL that a file-system object inherits from its parent directory when it is created.

2.6.3 Access Check Algorithm

```
if   proc.euid = file.owner  ⟹  owner entry determines access
else if  proc.euid matches one of the named user entries  ⟹  matching named user entry
    ↪ determines access
else if  one of the group ids of the process matches the owning group entry and that entry
    ↪ contains the requested permission  ⟹  that entry determines access
else if  one of the group ids of the process matches one of the named group entries and that
    ↪  entry contains the requested permissions  ⟹  that entry determines access
else if  one of the group ids of the process matches the owning group entry or any of the
    ↪ named group, but neither the owning group entry nor any of the matching named group
    ↪ entries contains the requested permissions  ⟹  access is denied
else  the others entry determines access


if  the entry determined above is the owner or other entry, and it contains the requested
    ↪ permissions  ⟹  access is granted
else if   the entry determined above is a named user, owning group or named group entry and
    ↪ it contains the requested permissions and the mask also contains the requested
    ↪ permissions (or there is no mask entry)  ⟹  access is granted
else   access is denied
```

2.7 Windows Access Control

Access is controlled by a Security Reference Monitor - applied to many different object types, where each object type has its own list of permissions. Very fine grained controls, with actions being separated such that we can have different permissions for each of the actions - actions arranged for POSIX compatibility.

Things are separated into easy to use subsets (can have fully grained control over which actions are allowed as well): (1) Full control, (2) Modify, (3) Read and Execute, (4) Read, (5) Write, with the actions being:

	Full Control	Modify	Read & Execute / List Folder Contents	Read	Write
Traverse Folder/Execute File	×	×	×		
List Folder/Read Data	×	×	×	×	
Read Attributes	×	×	×	×	
Read Extended Attributes	×	×	×	×	
Create Files/Write Data	×	×			×
Create Folders/Append Data	×	×			×
Write Attributes	×	×			×
Write Extended Attributes	×	×			×
Delete Subfolders and Files	×				
Delete	×	×			
Read Permissions	×	×	×	×	×
Change Permissions	×				
Take Ownership	×				

Security Identifier (SID): identifier for every use or group - Windows equivalent of the Unix user ID and group ID. Internally a variable-length string of integer values

2.7.1 Security Descriptor

Equivalent of access control information in inode with:

1. SID of object's owner
2. SID of object's primary group
3. Discretionary Access Control List
4. System Access Control List

where each Access Control Entry has:

1. type (AccessDenied, AccessAllowed)
2. SID (representing a user or group)
3. Access Permission Mask (read, write, etc)
4. Five bits to control ACL inheritance

Security Descriptor String Format: for storing or transporting information in a security descriptor.

Displaying Access Control Entries: (1) All non-inherited ACEs appear before all inherited ones, (2) within these categories, GUI interfaces with allow / deny buttons also usually place all AccessDenied ACEs before all AccessAllowed ACEs in the ACL, hence giving them priority.

2.7.2 Algorithm

1. Requesting processes provide a desired access mask - with no DACL present, any requested access is granted. With empty DACL, no access granted
2. All ACEs with matching SID checked in sequence until either all requested types of access have been granted by AccessAllowed entries or one has been denied in an AccessDenied entry

```
foreach Ace in Acl:
        if Ace.SID ∈ PrincipalSids and not Ace.inheritonly:
                if Ace.type = AccessAllowed:
                        Granted := Granted ∪ (Ace.AccessMark - Denied);
                else if Ace.type = AccessDenied
                        Denied := Denied ∪ (Ace.AccessMask - Granted)
                if DesiredAccess ⊆ Granted
                        return SUCCESS
        return FAILURE
```

2.7.3 Inheritance

Windows implements static inheritance for DACLs - only the DACL of the file being accessed is checked during access.

Alternative, dynamic inheritance, also consults that the ACL of ancestor directories along the path to the root, where necessary

New files and directories inherit their ACL from their parent directory when they are created. But, security descriptor can carry a protected-DACL flag that prevents its DACL from inheriting any ACEs. Each ACE has the following five bits:

1. Container inherit - will be inherited by subdirectories
2. Object inherit - will be inherited by files
3. No-propagate - inherits to children but not grandchildren
4. Inherit only
5. Inherited - was inherited from the parent - ensures that all inherited ACEs can be updated without affecting non-inherited ACEs that were explicitly set for that file or directory

Hence, when ACE inherited:

1. inherited is set
2. if container inherit is inherited to a subdirectory, then inherit only cleared
3. if object inherit is inherited to a subdirectory, inherit only is set
4. If no-propagate flag is set, then container inherit and object inherit is cleared
5. If the ACL of a directory changes, it is up to the application making the change to traverse the affected subtree below and update all affected inherited ACEs there

2.7.4 Auditing

SystemAudit ACEs can be added to an object's security descriptor to specify which access requests are audited. Users can also have capabilities that are not tied to a specific object.

2.7.5 Defaults

Default installations of old Windows used no ACLs for application software and every user and any application could modify most programs and operating system components (leads to a virus risk). This was changed in later versions, where users work without administrator rights.

Important to note that Windows has no support for giving elevated privileges to application programs - no equivalent to set-user-ID

2.7.6 Service

Windows program that normally runs (1) continuously from when the machine is booted to its shutdown and (2) independent of any user and has its own SID.

Client programs started by a user can contact a service via a communication pipe - can receive not only commands and data but can also use it to acquire the client's access permissions temporarily.

2.7.7 LDAP

Lightweight Directory Access Protocol is a data model and protocol used for remotely accessing admin information about users, groups and computers - has multiple implementations.

An LDAP server stored a tree of LDAP objects, each of which may be used to represent a user, group, computer, organisational unit, etc. Each LDAP object is a set of attribute, value pairs.

- Each non-root object has a parent and has one attribute (Relative Distinguished Name of the object). The RDNs along the path to the room form the Distinguished Name of the object.
- Each object belongs to an object class, which defines what attributes the object must have

Active Directory Domain Service Service offered by Windows Server - offering LDAP, Kerberos, DNS, DHCP, SMB / DFS and other protocol services. Most admin state can be accessed with LDAP. Uses the same discretionary ACL implementation, security descriptors and SIDs for LDAP objects as Windows uses for NTFS files and directories - enables fine-grained delegation of administrative access, with a few extensions:

1. Access-right bits are different: (1) create child, (2) delete child, (3) delete, (4) delete tree, (5) list contents, (6) list object, (7) read property, (8) write property, (9) write property extended, (10) control access, (11) write owner
2. Order of ACEs is sorted by the server (deny before allow, explicit before inherited)
3. **Object Type**: ACEs can contain an ObjectType field that identifies an attribute name or property set - hence ACE with property-related right only applies to attributes identified by that set not to the entire object
4. **Write property extended** right grants only write access after validation of rules identified by GUID in ObjectType field.

Principle of Least Privilege: Applications should only have access to exactly the objects and resources they need to perform their operation.

Transferable Capabilities: Some operating systems combine notion of object's name that is given to the subject and the access rights that the subject obtains to this object into a single entity:

$$capability = (object - reference, rights)$$

Capabilities can therefore be implemented efficiently as an integer that points to an entry in a tamper-resistant capability table associated with each process. In distributed systems, capabilities can be implemented as cryptographic tokens. Can also have the right to be passed to other subjects.

2.8 Mandatory Access Control

Restrictions to allowed information flows not decided at the user's discretion, but instead enforced by system policies. Aimed in particular at preventing policy violations by untrusted application software, which typically has at least the same access privileges as the invoking user
Air Gap Security: Use completely separate network and computer hardware for different application classes. No communications are allowed between an air-gap security system and the rest of the world. The exchange of storage media must be carefully controlled - must be zeroised before they can be reused on the respective other system.

Data Pump / Data Diode: Like air gap security with a one-way communication link that allows users to transfer data from the low-confidentiality to the high-confidentiality environment (not in the opposite direction)

2.8.1 Bell-LaPadula Model

Mandatory Access-Control for military multi-level security environments. Every subject (process) S and every object O is labelled with a confidentiality level L:

$$UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP\ SECRET$$

Each user has a range of allowed confidentiality levels and can chose at which level L(S) in that range to start a process. That process then cannot read data from above that level and cannot write data into files below that level.
Rules

1. S can only write to O if $L(S) \leq L(O)$
2. S can only read O if $L(O) \leq L(S)$
3. S_1 can only execute S_2 if $L(S_1) \leq L(S_2)$

A selected set of trusted users and processes is allowed to bypass the restrictions, in order to permit the declassification of information

In Practice:

1. Kernel reference-monitor enforces security properties - no read up (ss-property), no write down (*-property)
2. Operating-system components, such as scheduler and memory manage ment must be exempt - **trusted computing base**
3. **Goals**
   (a) Protection against malicious software exfiltrating confidential classified data
   (b) Reducing risk of users accidentally transferring classified data to inappropriate hardware, channels or other users

Issues

1. Information only flows up -> files end up TOP SECRET
2. Applications need to exchange information bidirectionally across security levels
3. Not suited for commercial applications where data integrity (prevention of mistakes and fraud) are the main concern, rather than confidentiality

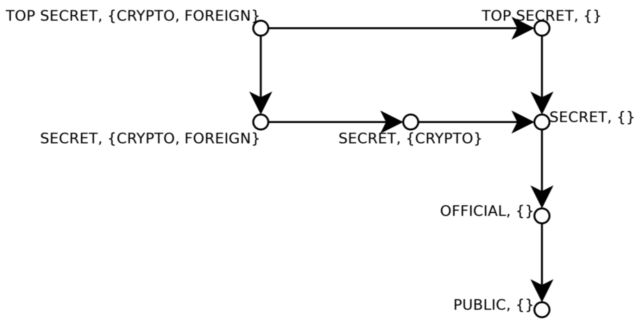
2.8.2 Lattice Model

Intelligence agencies label each document X with both a security level and a set of categories:

$$label(X) = (level(X), \{category_1, ..., category_n\})$$

This partial order allows information flows from Y to X:

$$X \geq Y :\Leftrightarrow \text{ level } (X) \geq \text{ level } (Y) \wedge \text{cat}(X) \supseteq \text{cat}(Y)$$



2.8.3 Covert Channel Problem

Reference monitors see only intentional communications channels - there are also covert channels which are not intended to transfer information. Malicious high-level data can use these to transmit high-level data to a low-level receiving process, who can then leak it to the outside world.

- **Resource Conflicts**: if high-level process has created file F, low-level process will fail when creating file of same name - this is 1 bit information
- **Timing Channels**: processes can use system clock to monitor their own progress and infer the current load, into which other processes can modulate information
- **Resource State**: High-level processes can leave shared resources (disk head position, cache memory content) in states that influence the service response times for the next process
- **Hidden information in downgraded documents**: Can use steganographic techniques
- **Cache-Based Covert Channel**: CPUs obtain performance from on-chip caches with differing time between accesses of different caches.

2.8.4 Integrity Models

Each subject S and object O is labelled with integrity level I

1. **Low-water-mark policy**
   (a) S can only write to O if I(O) $\leq$ I(S)
   (b) After S has read from O: I(S) := min$\{I(S), I(O)\}$
   (c) S_1 can only execute S_2 if $I(S_2) \leq I(S_1)$
   (d) Implemented on FreeBSD in mac-low policy
2. **Biba's Model**
   (a) S can only write to O if I(O) $\leq$ I(S)
   (b) S can only read O if I(S) $\leq$ I(O)
   (c) S_1 can only execute S_2 if $I(S_2) \leq I(S_1)$
   (d) Implemented on FreeBSD in mac-biba policy

2.9 Commercial Systems

Commercial Systems have to maintain both internal consistency (that which can be checked automatically) and external consistency (data accurately describes the real world)

Achieve this with **well-formed transactions** - access to these must be audited and controlled by separation of duty.

2.9.1 Clark / Wilson Framework

- Formalises this idea
- Integrity protected data is referred to as **Constrained Data Items (CDIs)** which can only be accessed via Transformation Procedures
- Also exist **Integrity Verification Procedures** - check validity of CDIs and special **Transformation Procedures** that transform **Unconstrained Data Items** into CDIs

Requires:

1. $\forall CDIs \; \exists \; an \; Integrity \; Verification \; Procedure$
2. All TPs must be certified to maintain the integrity of any CDI
3. All CDI can only be changed by a TP
4. A list of (subject, TP, CDI) triplets restricts execution of TPs

   This access control list must enforce a suitable separation of duty among subjects and only special subjects can change it
5. Special TPs can convert Unconstrained Data Items into CDIs
6. Subjects must be identified and authenticated before they can invoke TPs
7. A TP must log enough audit information into an append-only CDI to allow later reconstruction of what happened
8. Correct implementation of the entire system must be certified

2.10 Operating System Extensibility for Access Control

Mandatory Access Control: proposed in literature but not used - no agreement as to why they are useful

System-Call Interposition: not effective at user level because of time-of-check-to-time-of-use (TOCTTOUT) problems: security extensions need to operate within kernel-lock regions

Both Linux and FreeBSD implemented kernel hooks that allow the addition of different security policies as loadable kernel modules - instead of hard-wiring one mandatory-access-control policy. These can be used to isolate mobile-device applications

2.11 Linux Security Modules

- Adds an opaque security field (void * pointer) to many kernel data structures: *(1) task, (2) superblock, (3) inode, (4) open file, (5) network buffer / packet, () device, (7) IPC objects* - security modules can add arbitrary metadata (ACLs, labels, names)
- Adds global security-ops table of function pointers, called by kernel functions that access kernel objects
- vfs-mkdir() calls *security-ops -> inode-ops -> mkdir(dir, dentry, mode)* before a new directory entry dentry is added to a parent directory dir with permissions mode (a non-zero return value denies the operation) and it also calls:

  security-ops -> inode-ops -> post-mkdir(dir, dentry, mode) after the directory as been created, to allow the security module to add its metadata.
- Called after built-in discretionary access control is granted
- Security modules are stackable if supported by the modules loaded earlier, which provide another LSM interface to the next module

2.11.1 AppArmor

LSM designed to protect system by confining the access of potentially insecure processes - used to protect systems against server processes exposed to a network

1. **Profile**: plain-text file that lists all the resources and privileges that a particular program is allowed to access - can be switched into enforce or complain (logs which system calls would have been blocked in enforce) mode
2. Affects only executable files for which a profile was loaded
3. Enforced restrictions are independent of user identity, group membership or object ownership, and even apply to root

- Executable file /usr/sbin/ntpd is an NTP daemon to synchronise system clocks over the internet - this profile limits the resources and capabilities accessible to this program to those listed, even if /usr/sbin/ntpd is started by root
- Permits (1) read/write access to pulse-per-second (PPS) clock hardware, (2) read access to all standard public OS files, (3) read access to its own config file, (4) read, executable mmap and inherit execute to its own binary, (5) permits access to privileged UDP ports, (6) ability to set system time, (7) include base access rights by most Linux programs

```
/etc/apparmor.d/usr.sbin.ntpd

/usr/sbin/ntpd {
       #include <abstractions/base>

       capability net_bind_service,
       capability sys_time,
       network dgram,
       /dev/pps[0-9]* rw,
       /{,s}bin/ r,
       /usr/{,s}bin/ r,
       /usr/sbin/ntpd rmix,
       /etc/ntp.conf r,
}
```

2.11.2 SELinux

NSA's original LSM for mandatory access control - where each process and object is labelled with a security context: *user:role:type(domain (in the case of processes)):range* - user and role are independent of POSIX user and group identifiers

Supports two independent MAC mechanisms:

1. **Domain / type enforcement**: Policy language for rules to allow operations based on the the user/role/-type/domain assigned to each process and object in its security-context label
2. **Multi-level / multi-category security**: BLP lattice information-flow control policy based on the range field in the security context - contains the security levels and security categories that domains are allowed to access

Policy rules define:

1. Which processes and which objects are labelled with which domain / type
2. Processes in which domains are allowed which access to objects of some type
3. How domains can interact with each other
4. How processes are allowed to transition from one domain to another - **elevated privileges**

Domains and types define equivalence classes (form a layer of abstraction between processes (and objects) and access-control policy on the other): processes in the same domain have the same access rights AND objects of same type can be accessed in the same way by processes in some domain

2.11.3 Android Access Control

- Separate POSIX user identifier allocated for each installed application
- Discretionary access (POSIX) separates applications which are structured into components that interacts via Android-specific message objects (intents) that are passed to a kernel IPC mechanism that mediates all inter-component interactions
- **Application Packages** come with **XML manifest** that requests permissions to access certain system facilities

2.11.4 chroot jails

BSD and Linux, root can call chroot(path) to change location of the root directory of the filesystem for the current process - can be used to constrain files visible to a process in the **following way**:

1. Make list of all files required by an application
2. Create new jail directory
3. Copy all required files into the jail directory, under relative paths from root
4. Call chroot() to change root directory to the jail directory and start application from inside the jail directory

Containers: take chroot jail further and provide support for system wide namespaces: (1) user and group id number spaces, (2) network devices, (3) mount tables.

Can be combined with process-tree resource control to allow applications to be executed in a container environment using their own shared libraries and config files

3 Operating-System Security

Trusted Computing Base: parts of the system that enforce a security policy - a good security design attempts to make TCB as small as possible to minimise the chance for errors in its implementation and to simplify careful verification

Basic OS Security Functions

1. **Domain Separation**: TCB must be protected from external interference and tampering by untrusted subjects
2. **Reference Mediation**: All accessed by untrusted subjects must be validated by TCB before succeeding
3. **Residual Information Protection**: OS must erase any storage resources before they are allocated to a new subject to avoid information leaking from one subject to the next - **object reuse / storage sanitation**

   Residual information can be erased when resource is (1) allocated to subject - *residual information can sometimes be recovered after user believes it has been deleted* or (2) deallocated from subject.

3.1 Classification

Defined by US DoD 1983:

1. **Class D**: **minimal protection** - no authentication, access control or object reuse - *MS-DOS, Windows 98*
2. **Class C1**: **discretionary security protection** with support for (1) discretionary access control, (2) user identification and authentication, (3) tamper resistant kernel, (4) security tested and documented - *classic Unix*
3. **Class C2**: **controlled access protection** - adds (1) object reuse, (2) audit trail of object access, (3) ACLs with user granularity - *Windows NT*
4. **Class B1**: **labelled security protection**, adds (1) confidentiality labels for objects, (2) mandatory access control policy, (3) thorough security testing
5. **Class B2**: **Structured protection**, adds (1) trusted path from user to TCB, (2) formal security policy model, (3) minimum / maximum security levels for devices, (4) well-structured TCB and user interface, (5) accurate high-level description, (6) identifies covert storage channels, (7) estimates bandwidth, (8) system administration functions, (9) penetration testing, (10) TCB source code revision control and auditing
6. **Class B3**: **Security domains**, adds (1) security alarm mechanisms, (2) minimal TCB, (3) covert channel analysis, (4) separation of system admin and security admin
7. **Class A1**: **Verified Design**, adds (1) formal model for security policy, (2) formal description of TCB must be proved to match the implementation, strict protection of source code against unauthorised modification

Common Criteria for Information Technology Security Evaluation: Created in 1999 by TCSEC and ITSEC - provides a framework for defining new product and application specific sets of security requirements (protection profiles). Notably, it separates functional and security requirements from the intensity of required testing - measured by the **evaluation assurance level**

- *EAL1: (1) tester reads documentation, (2) performs some functionality tests*
- *EAL2: developer (1) provides test documentation and (2) vulnerability analysis for review*
- *EAL3: developer (1) uses RCS, (2) provides more test and design documentation*
- *EAL4: (1) low-level design docs, (2) some TCB source code, (3) secure delivery, (4) independent vulnerability analysis (highest level considered economically feasible)*
- *EAL5: (1) Formal security policy, (2) semi-formal high-level design, (3) full TCB source code and independent testing*
- *EAL6: (1) Well-structured source code, (2) reference monitor for access control, (3) intensive pen testing*
- *EAL7: (1) Formal high-level design and (2) proof of correctness of implementation*

4 Software Security

4.1 Malicious Software

1. **Trojan Horse**: software with hidden malicious side effects
2. **Randomware**: trojan that blackmails users
3. **Backdoor**: function in a trojan that enables unauthorised access
4. **Logic bomb**: trojan that executes malicious function only when specific trigger condition is met
5. **Virus**: self-replicating program that can infect other programs by modifying them to include a version of themselves, often carrying a logic bomb as a payload.

   They can only spread in environments where (1) access control policy allows application programs to modify the code of other programs, (2) programs are frequently exchanged in executable form and copied from one machine to another

Was very widespread in MS-DOS but largely disappeared post Vista due to (1) User Account Control and (2) Windows Integrity Mechanism. Also, (3) software generally installed from online software repos - therefore no direct copying and (4) effective malware scanners.

Malware Scanners: use databases with characteristic code fragments of most known viruses and Trojans (3 million) as well as behavioural patterns and monitoring changes in files using cryptographic checksums.

6. **Worm**: self-replicating program that spreads onto other computers by breaking into them via network connections and (unlike a virus) starts itself on the remote machine without infecting other programs
7. **Root kit**: OS modification to hide intrusion
8. **Man-in-the-browser**: Web plugin that intercepts and manipulates traffic
9. **Spyware**: trojan that gathers and sends back information about users without consent
10. **Keylogger**: spyware recording keyboard input
11. **Scareware**: software or web page that pretends to be a diagnostic tool to buy software
12. **Adware**: software that displays unwanted marketing messages
13. **Grayware - potentially unwanted software**: adware that users are tricked into installing which they are unlikely to have installed deliberately despite being legitimate applications

4.2 Common Vulnerabilities

1. Missing checks for **data size**
2. Missing checks for **data content**
3. Missing checks for **boundary conditions**
4. Missing checks for **success / failure of operations**
5. Missing **locks - insufficient serialisation**
6. **Race conditions** - time of check to time of use

   Developers often forget that they work on preemptive multitasking system.

   xterm program is setuid root and allows users to open a log file to record what is being typed. Log file opened by xterm by (1) change subprocess to real uid/gid to test with access(logfilename, W-OK) whether the writable file exists. If not, creates the file owned by the user. (2) Call open(logfilename, O-WRONLY | O-APPEND) to open the existing file for writing

   Exploit provides as logfilename the name of the symbolic link that switches between a file owned by the user and a target file. If access() is called while the symlink points to the user's file and open() is called while it points to the target file, the attacker gains via xterm's log function write access to the target file
7. Incomplete **checking of environment**
   - Semantics of some library functions depends on state of execution environment - for example the PATH environment variable
   - Can fix a number of vulnerabilities by using an absolute path and by using **responsible disclosure**
8. **Unexpected side channels**
9. **Lack of authentication**
10. **Integer overflow**: Can lead to buffer overflows and similar exploits caused by integer overflows
11. **Subtle syntax incompatibilities**: For example, overlong UTF-8 sequences - where first 7 bits are all compatible with ASCII characters - all ASCII characters (U0000-U007F) are represented by ASCII bytes (0x00-0x7f), whereas all non-ASCII characters are represented by sequences of non-ASCII bytes (0x80-0xf7). Only shortest possible UTF-8 sequence is valid for any unicode character, but many UTF-8 decoders accept also the longer variants. Many security applications test strings for absence of certain characters (in ASCII), therefore by using overlong versions, can slip characters past. This led to an IIS vulnerability.

12. **Insufficient Parameter Checking**: example is a read buffer overflow in a smartcard where exchanges include a number of bytes expected - can literally just give a higher value and then receive RAM content beyond the buffer.

4.3 Buffer Overflows

Buffer overflow if you continue to read from after the length of the buffer. In order to exploit a buffer overflow, the attacker typically prepares a byte sequence that consists of:

1. **Landing pad or NOP-sled** - initial series of no-operation (NOP) instructions that allow for some tolerance in the entry jump address
2. Machine instructions that modify a security-critical data structure or that hand control to another application to gain more access
3. Some space for function-call parameters or stack operations
4. Estimated start address of the buffer (landing pad) in the form used for return addresses on the stack

Stopping buffer overflows

- Use programming languages with array bounds checking
- Compiler adds check values between buffers and return addresses on stack
- Shadow stacks
- Let memory-management unit disable code execution on the stack
- Address space layout randomisation

Return-Oriented Programming: If stack execution disabled, exploit existing instructions elsewhere, changing values of the executable where there are useful sequences, overwriting the stack. Therefore, the attacker can still write programs on the stack that load registers and invoke system calls, without executing any machine instructions on the stack.

Can also target other values than return addresses on the stack including: (1) security critical variables, (2) function pointers that called later and (3) frame pointers

Heap Exploits Overflowing buffer - *obtained by malloc() and free()*- sits in a chunk (unit of allocation used by the heap management library, next to other such chunks). The buffer overflows on heap can be exploited by overwriting pointers in the metadata (containing chunk-size information and two pointers, forward and backward - allows us keep deallocated chunks in a doubly-linked list) associated with the next chunk. free() operation will manipulate these value to return a chunk into a doubly-linked list, then after careful manipulation on chunk's metadata, call of free() on a neighbour chunk will perform any desired write operation into memory.

4.4 Missing check of input data - shell meta-characters

Web server allows users to provide email address in form field to receive a file. Address received by a Perl CGI script and stored in $email. Script then attempts to send out the email with the command:
$system(”mail\$email < message”);$
Attacker provides a carefully selected pathological address such as: trustno1@hotmail.com < /var/db/creditcards.log ; echo
and can execute arbitrary commands

Solutions:

- use safe API function instead of constructing shell commands
- Prefix / quote each meta-character handed over to another software with a suitable escape character

SQL Injection: checks for meta characters often forgotten for text strings passed to SQL engines
HTML Cross-Site Scripting: Receive text strings from users that are embedded into HTML code pages

served to other users. Therefore must check for meta-characters or users can inject into your web pages code that is executed by other's web browsers

4.5 Sourcing Secure Randoms

Can combine a number of 'sources' of randomness into a has function to condense the accumulated entropy into a smaller number of good randoms where none of the sources alone provides unbiased random bits

1. Dedicated hardware - thermal noise from reverse-biased diode, unstable oscillators, Geiger counters
2. High-resolution timing of user behaviour - key strokes, mouse movement
3. High-resolution timing of peripheral hardware response times - hard drives
4. Noise from ADC
5. Network packet timing and content
6. High resolution time

Provision of secure source of random bits generally an essential OS service

4.6 Security Testing

4.6.1 Penetration Testing / Flaw Hypothesis Method

1. Put together team of software developers with experience on tested platform and in computer security
2. Study user manuals, design documentation and source code of the target
3. Prepare list of potential flaws (allow users to violate security policy), considering: (1) System control structure (user system interactions), (2) Common programming pitfalls, (3) Historic vulnerabilities and attack strategies, (4) Gaps in documented functionality, (5) Rarely used and recently added functions
4. Sort list of flaws by estimated likelihood AND ease of testing
5. Test until time / budget is exhausted adding new flaw hypotheses as test results provide clues

4.6.2 Fuzz Testing

1. Generate random, invalid and unexpected program inputs until one found that crashes software
2. Then investigate the cause

Aim is to maximise the code coverage in the following ways:

1. **Mutation Fuzzing**: randomly modify existing valid test examples
2. **Structure-Aware Fuzzing**: test generator has syntax description of file formats or network packets and generates tests that contain a mixture of valid and invalid fields
3. **GUI Fuzzing**: send random keyboard and mouse-click events
4. **White-box Fuzzing**: use static program analysis, constraint solving to generate test examples
5. **Gray-box Fuzzing**: use code instrumentation instead of program analysis
6. **Evolutionary Fuzzing**: mutation fuzzing with feedback from execution traces

5 Cryptography

5.1 Basic Primitives

Hash Function h: $\{0,1\}^* \to \{0,1\}^n$ maps variable-length bit strings tp short, fixed-length bit strings using an efficient deterministic algorithm st the probability of a collision h(X) = h(Y) for inputs $X \neq Y$ is minimized. A secure hash function is collision resistant - computationally infeasible for anyone to find any pair of input strings X and Y st h(X) = h(Y) with X $\neq$ Y.

Collisions for any function with 2^n possible outputs can be found in $2^{n/2}$ steps

5.1.1 Standards

1. **SHA-2** where n = 224, 256 (- for 32-bit CPUs) and 384, 512 (- for 64-bit CPUs). On 32-bit CPUs, it takes about 20 cycles to generate a byte and on 64-bit, takes around 10 cycles / byte
2. **SHA-3**: has an arbitrary output length

5.2 Message Authentication Code

Maps a k-bit private key K (uniformly picked at random out of $\{0,1\}^k$) and variable length message M to an n-bit tag st any opponent that does not know key K cannot guess the tag correctly with probability better than 2^{-n} - *existential unforgeability.* Can use a block-cipher based approach or a hash based one.

5.3 Private-Key Encryption Scheme

Pair of functions:

$$\text{Enc} : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^{*+v}$$
$$\text{Dec} : \{0,1\}^k \times \{0,1\}^{*+v} \rightarrow \{0,1\}^*$$

where Enc is an efficient, non-deterministic function that maps a k-bit private key K (picked uniformly at random out of $\{0,1\}^K$) and a variable-length plaintext message to a v bits longer ciphertext message such that the message = $\text{Dec}_K(Enc_K(M))$ where Dec is an efficient deterministic and st an opponent who does not know K must not be able to learn anything new about M (other than the length of M) from being able to observe C = $\text{Enc}_K(M)$

Assumptions for both Message Authentication Code and Private-Key Encryption Scheme

1. Opponent can query for many differeent plain text messages
2. Opponent knows all the definitions for Enc and Dec except for key K - **Kerckhoffs' Principle**
3. Key picked randomly

5.4 Authenticated Encryption Scheme

Private key encryption scheme that also provides ciphertext integrity, i.e. decryption will reject as invalid any ciphertext message that was not generated by the encryption function using the same private key. An opponent will not be able to forge a valid new ciphertext that the decryption function will then accept.

Can be implemented using key-derivation funtion to derive from a private key, two different new keys for use with an encryption function and a message authentication function. Authenticated encryption function first encrypts the plain-text message and then appends the authentication code of the ciphertext.

There are a number of AESD schemes which can be used including AES-GCM, AES-CCM, AES-EAX, AES-OCB

5.5 Summary

N.B. $\leftarrow$ *assigns output of probabilistic algorithm,* := *of a deterministic algorithm*

5.5.1 Key Exchange

1. $(PK_A, SK_A) \leftarrow Gen$ - public/secret key-pair generation by Alice
2. $(PK_B, SK_B) \leftarrow Gen$ - public/secret key-pair generation by Bob
3. $K := DH(SK_A, PK_B) = DH(PK_A, SK_B)$ - key derivation from exchanged public keys

5.5.2 Digital Signature

1. $(PK, SK) \leftarrow Gen$ - public/secret key-pair generation
2. $S \leftarrow Sign_{SK}(M)$ - signature generation using secret key
3. $\text{Vrfy}_{PK}(M,S) = 1, M' \neq M \overset{?}{\Rightarrow} \text{Vrfy}_{PK}(M',S) = 0$ - signature verification using public key

6 Entity Authentication

6.1 Passwords

How to identify humans: should combine a few of these

1. Something they are: **biometric identification**
2. Something they do: signature, keystroke dynamics, voice
3. Access tokens
4. Something they know - passwords, PIN. Unfortunately, hard to create passwords as randomnly picked single words (most common passwords) have low entropy. Requirements:
   (a) Reject delay
   (b) Trusted path (control+alt+delete)
   (c) Confidentiality of password database - store hash rather than plaintext password - use nonce concatenated with passwords to prevent comparison with precalculated hashed dictionary in case of stolen password database
   (d) Monitor failed logins
   (e) Minimum length, inclusion of digits, punctuation and mixed case letters
   (f) Suggest phrases
   (g) Compare passwords to common passwords
   (h) Issue randomly generated PINs and passwords
   (i) Password manager
   (j) One time passwords
   (k) Single sign-on
5. Location

6.2 Authentication Protocols

Allow claimant to prove their identity to a verifier by showing knowledge of a secret key - stops masquerade attacks:

1. Replay attacks - using previously transmitted message
2. Reflection attacks - returning previous message to originator
3. Interleaving attack - using information from one or more ongoing or previous authentication exchanges

Perform challenge-response exchanges of tokens that include an unforgeable cryptographic check values including: (1) message authentication codes and (2) time variant parameters (nonces) including timestamp.

6.2.1 Properties to consider

1. Number of message roundtrips rqd
2. Whether provide unilateral or mutual authentication
3. Key infrastructure that needs to be set up in advance
4. Whether they involve a trusted third party
5. What key entropy is required, whether keys can be revoked
6. Required computational effort
7. Scalability
8. Number of bits exchanged
9. Security properties offered, and challenge in proving these
10. Privacy protections

6.2.2 Notation

- A, B, C, S - Protocol participants (principals)
- T_X - Timestamp generated by participant X
- N_X - Sequence number generated by participant X
- R_X - Random number generated by participant X
- K_{XY} - Symmetric key shared by participants X and Y
- PK_X, SK_X - Public / secret key pair
- M - optional message field
- , - unambiguous concatenation of protocol fields

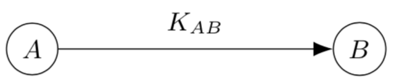
X $_K$ = (X, Mac$_K$(X)) *protect integrity of message X by appending a MAC*

- $\{X\}_K$ - apply authenticated encryption to message X

Y $\{X\}_K$ - apply authenticated encryption to message X, with additional plaintext data Y included for integrity protection

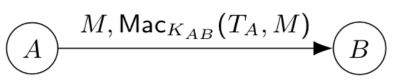
6.2.3 One-pass Authentication

Password or PIN:



Problem: Eavesdropper can replay K_{AB}.

MAC of implicit timestamp:



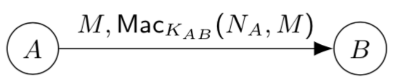
Once B has seen Mac$_{K_{AB}}(T_A)$ it should not accept any Mac$_{K_{AB}}(T'_A)$ with $T'_A \leq T_A$.

Search and clock-offset/frequency tracking needed if clocks are not synchronized.

May include optional message M.

Example: RSA SecurID, TOTP (RFC 6238)

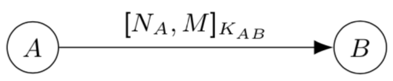
MAC of implicit counter:



Token can be short (for manual entry), but B must remember (or search for) N_A.

Once B has seen Mac$_{K_{AB}}(N_A)$ it should not accept any Mac$_{K_{AB}}(N'_A)$ with $N'_A \leq N_A$.

MAC of explicit counter:



As above, but no search needed if some N_A were lost.

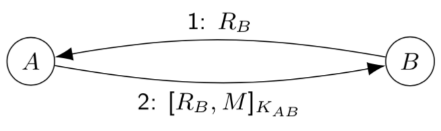
Examples: Chip Authentication Program (e.g., Barclays PINsentry) "Identify" function, some car key remote fobs ($M \in$ {lock, unlock})

- Verifier B has to keep state (last seen timestamp from A) to detect replay attacks. If multiple instances of verifier B, this state must be synchronised between them
- Messages may have been sent long before verifier sees them

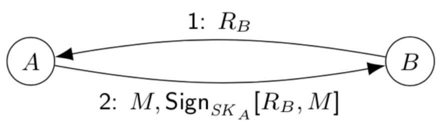
6.2.4 Two-pass Authentication

Issues can be avoided using challenge-response protocols

MAC of random challenge



Signature of random challenge



6.2.5 Mutual Authentication

Allow A to gain assurance of identity of B as well

Two-pass mutual authentication with MAC of time or counter:

$$\begin{tikzcd}
A \arrow[r, bend right=10, "2: \left[{T_A \atop N_A}, M_2\right]_{K_{AB}}"'] & B \arrow[l, bend right=10, "1: \left[{T_B \atop N_B}, M_1\right]_{K_{AB}}"']
\end{tikzcd}$$

Three-pass mutual challenge–response with MAC:

$$\begin{tikzcd}
A \arrow[r, "2: R_A, M_1, \mathsf{Mac}_{K_{AB}}(R_A, R_B, M_1)"] & B \arrow[l, bend right=20, "1: R_B"'] \arrow[l, bend left=20, "3: M_2, \mathsf{Mac}_{K_{AB}}(R_B, R_A, M_2)"]
\end{tikzcd}$$

Reflection Attack: Principals can act as both claimants and verifiers and support concurrent protocol sessions. Allows attacker A' to masquerade as A by returning challenge to B in second session. Fix this by avoiding using same key for multiple purposes.

6.3 Kerberos

Needham Schroeder: User A and server B does not share a secret key initially but server shares key with everyone. A requests session with B from S. S generates session key $K_{A,B}$ and encrypts it separately for A and B - tickets contain timestamp and lifetime

$$\begin{array}{ll}
A \rightarrow S: & A, B \\
S \rightarrow A: & \{T_S, L, K_{AB}, B, \{T_S, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
A \rightarrow B: & \{T_S, L, K_{AB}, A\}\, K_{BS}, \{A, T_A\}_{K_{AB}} \\
B \rightarrow A: & \{T_A + 1\}_{K_{AB}}
\end{array}$$

Kerberos serves the purposes of identifying users, computers and services on computers with a principal name of form (1) user or service/computer@REALM where realm identifies a set of key distribution centers that can authenticate the principal.

- Initial K_{AS} from A's password hashed
- Instead of caching password in memory:
  - **kinit** requests a ticket granting ticket which it decrypts using password hash - then discards password and long term TGT decryption key and stores the short-term session key for future communications with the TGT service kerberostgt/REALM@REALM
  - Then use that cached session each time it needs to request a session ticket for a different server. When key expires - can be auto-renewed or made renewable for longer than the default 10 hours, user must reenter the password
- Pre-authorisation step ensures that key distribution center S does not reply with any TGT encrypted under K$_{AS}$ unless sender has demonstrated knowledge of K$_{AS}$ to hinder offline password guessing
- Mechanism for forwarding tickets allows server to impersonate remote user - access other services on their behalf
- There is also support for a federation of realms

7 Network Security

Problems

1. WAN allows attacks from anywhere often via compromised intermediary machines given international law enforcement is rather difficult

2. Commonly used protocols are not designed for a hostile environment in one of a couple ways: (1) authentication missing or based on source address, cleartext password, or integrity of remote host, (2) missing protection against DOS attacks
3. Use of bus and broadcast technologies, promiscuous mode attack interfaces
4. Vulnerable protocol implementations
5. DDOS attacks

7.1 TCP/IP Security

Transport connections characterised by (1) source IP address, (2) destination IP address, (3) source port, (4) destination port - port numbers $<$ 1024 are privileged, under Unix only root can open them - this is used by some Unix network services to authenticate peer system processes

7.1.1 Address Spoofing

IP source address provided by originating host which can provide wrong information which is verified during the TCP 3-way handshake:

$$\begin{aligned} S \rightarrow D: &\quad \text{SYN}_{\ x} \\ D \rightarrow S: &\quad \text{SYN}_y, ACK_{x+1} \\ S \rightarrow D: &\quad ACK_{y+1} \end{aligned}$$

Only third message starts data delivery, therefore data communications will only proceed after claimed originator has confirmed reception of a TCP sequence number in ACK message. From then on, TCP ignore messages with sequence numbers outside confirmation window - can act like an authentication nonce

7.1.2 Vulnerabilities

1. IP loose source route option allows S to dictate an explicit path to the destination and old specifications require destination machines to use the inverse path for the reply, ensures no need of 3-way TCP handshake
2. UDP is more vulnerable to address spoofing - no sequence numbers
3. TCP implementations *today* randomise start sequence numbers - with predictable start sequence numbers, attacker could without having access to reply packets sent from destination to source: (1) impersonate source without receiving second message - **sequence number attack** and (2) disrupt ongoing communication - **session hijacking**
4. In older TCP implementations, destination allocates a temporary data record for every half-open connection in a small buffer - can flood this - **SYN flooding**
5. DNS issues
   (a) DNS caching issues related issues
   (b) DNS resolvers configured to complete name prefixes automatically

7.2 Firewalls

Dedicated gateways between LANs and WAN - all traffic between LAN and WAN must pass through the firewall and there checked for compliance with local security policy. Work in the following ways:

1. **Port Blocks**: drops or passes packets based on matches with configured sets of IP addresses and / or port numbers. Allows sysadmins to control at a single configuration point which network services are reachable at which host
2. **SYN Blocks**: basic packet filter can distinguish TCP traffic (in both directions) because opening packet lacks ACK bit
3. **Stateful TCP/UDP filters**: require implementation of TCP state machine or track state of UDP-based protocols - this is beyond capabilities of most router hardware

4. **Ingress Filtering**: performs plausibility checks on source IP addresses. Possible iff firewall has interfaces $\{I_1, ..., I_n\}$ and is positioned in the network st all hosts reachable via interface I_k have an IP address from a set S_k, while none of the hosts reachable via another interface I_l have an address in S_k. Firewall can then drop all packets arriving on I_k that does not have source address in S_k and can also drop packets arriving on interface I_l that do have a source address in S_k

5. **Application gateway**: firewalls check for protocol violations above the transport layer and above to protect implementations on the intranet. Some implement entire application protocol stacks in order to sanitise syntax of protocol data units and suppress unwanted content

6. **Logging and Auditing**: record suspicious activity and generate alarms, eg. port scans where single outside host sends packets to all hosts of subnet

Limits of Firewalls:

1. Once host behind firewall has been compromised, attacker can communicate over open protocol

2. Little protection against insider attacks

3. Rigid firewall policies severely disrupt deployment of new systems.

   Newer protocols designed to resemble HTTP

7.3 Virtual Private Networks

Setups apply authenticated encryption to IP packets to achieve security properties similar to those offered by a private direct physical connection

7.3.1 Site-to-Site VPN

Each router at boundary between participating site and rest of internet seperates data packets travelling between participating sites and then applies authenticated encryption to such traffic, along with ingress filtering such as: (1) external eavesdroppers cannot read intra-site traffic, (2) packets with source addresses from one of participating sites are allowed to enter other sites if they have arrived there protected by correct authenticated encryption key for traffic from that site

Hence, source address can be used in packet filters to decide whether data packet originated from one of organisation's sites and network access control applied accordingly

7.3.2 Remote Access VPN

Also activated in a mobile device to route traffic from that device to an organisational intranet. Gives device a tunneled interface with an address on the intranet as if it were located on the inside

7.4 Distributed Denial of Service Attack

Aim: overload internet servers or their connection infrastructure using traffic from a wide range of source addresses

Bot-net Based Attacks: Compromise many computers to run a remote-controlled software that generates high network traffic to a target server that is difficult to distinguish from legitimate traffic

UDP based Amplification Attacks: protocols respond to incoming unauthenticated request packets with significantly longer response packets and send these to the source address found in the request packets

7.5 Hyper Text Transfer Protocol

7.5.1 Version 0.9

Client contacts a server on TCP port 80 sends a request line and received a response file - can be demonstrated via any generic TCP tools

Uniform Resource Locator syntax: scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]

HTTPS: uses TCP port 443 and TLS protocol to authenticate the server and encrypt the HTTP connection

7.5.2 Version 1.0

More flexible and verbose: (1) Header starts with status-code line, (2) headers can carry additional fields, (3) request and response headers each finish with an empty line

7.5.3 Version 1.1

Request header can also have fields and even a message body. Also, requires that the client identifies the server name in a 'Host:' field (for servers with multiple hostnames on the same IP address)

7.5.4 Request Headers

In each request header, web browsers offer information about their software version, capabilities and preferences, eg:

```
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:58.0) Gecko/20100101 Firefox/58.0
Accept: text/html,application/xhtml+xml,application/xml; q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

7.5.5 State and Session Context

- HTTP stateless protocol - TCP connection can terminate after each request / response
- Therefore, have a number of ways of adding context:
  1. **Cookie**: server maintained state indicators in headers
  2. **Referer**: where did URL come from
  3. **Authorisation**: basic password authentication

7.6 Redirects

HTTP server can respond with 3xx status code and a Location field to send the client elsewhere for the requested resource:

- 301 - moved permanently
- 302 - found (temporary new link)
- 303 - see other (go there, but may not know what is expected)

7.7 Basic Authentication

Supports simple password mechanism:

```
$ nc -C www.cl.cam.ac.uk 80
GET /~mgk25/hello-basic.html HTTP/1.0
Authorization: Basic Z3Vlc3Q6Z1VlU3Q=
```

However, not widely used as (1) designer has no control over appearance of pop-up password prompt, (2) clear-text password included in each request and (3) no way to logout except closing browser

7.8 Form Methods

7.8.1 GET

- Meant for operations that have no side effects
- Assumed requests are idempotent
- Field values appended to URL st they can easily be quoted, bookmarked and included in links
- Responses can be cached

7.8.2 POST

- Meant for operations with side effects, especially non-idempotent ones
- Browser must not repeat POST request without explicit confirmation by user
- Form fields kept out of URL bar st users cannot accidentally repeat or reveal then via links, quotations or bookmarks
- Sent as request content type: *application/x-www-form-urlencoded*

7.9 Cookies

Web browsers maintain database table where web servers can store (*name=value*) entries (can set multiple cookies at the same time) as data that the browser will present to the server in future request headers

```
Set-Cookie: sid=hJsndj47Sd8sl3hiu; HttpOnly; Secure
Set-Cookie: lang=en-GB
```

which clients return as:

```
Cookie: sid=hJsndj47Sd8sl3hiu; lang=en-GB
```

- Information can be followed by attributes, seperated by semicola.
- Browsers store attributes with each cookie but do not return them in the Cookie: header
- Secure flag ensures cookie only included in HTTPS requests and omitted from HTTP requests
- HttpOnly flag ensures cookie only visible in HTTP(S) requests to servers but not accessible to client-side JS code via the document.cookie API
- By default, browsers return cookies only to server that set them (recording the hostname used but not the port)
- Servers can also limit cookies to be returned only to certain URL prefixes

  ```
  Set-Cookie: lang=en; Path=/~mgk25/; Secure
  ```

  means browsers will only include in requests to URLs starting with http://www.cl.cam.ac.uk/ mkg25/
- Can also explicitly specify a domain with 'Domain=...'
- If browser receives cookie with same name, Domain value and Path value as a cookie that has already stored, the existing cookie evicted and replaced with new cookie - if different Domain or Path values, browsers store multiple cookies of same name
- Browsers reject Domain values that do not cover origin server's hostname and reject public suffixes
- **Expiration**: by default, expire at end of browser session but can specify an expiry date or max storage duration

7.9.1 Session Cookies

(1) Verify provided password, (2) generate and store in the browsers a cookie to authenticate the rest of the session

Things to consider:

- Passwords linger in browser memory and can be stolen
- Malleable encryption may enable forging of other users' session cookies
- No possibility of logout, the cookie is valid forever

$$S = base64(userid, logintime, Mac_K(userid, logintime, clientip))$$

Unforgeable MAC protects both the user ID and login time, which enables the server-side limitation of the validity period. No need to set the *Expires* attribute - quitting the browser will end the session by deleting the cookie

Stateful Servers: Can simply use a large unguessable random number as a session cookie and compare against a stored copy.

Server-side logout is possible by deleting cookie in the server

Can replace nonce at each HTML request - though causes synchronisation problems when user presses back or reload

Pitfalls:

1. **Leaked MAC key**: can be found through (1) SQL injection attack, (2) from config file through GET request, (3) through stolen backup config files

   Countermeasures

   (a) Append password hash o the cookie and store in the database instead as the value to check the passwords and session cookies against
   (b) Rotate short-term MAC keys
   (c) Hardware security modules

2. **Missing Secure or HttpOnly Flags**: authentication cookies and login passwords can be eavesdropped unless HTTPS is used - can be stole in cross-site scripting attacks

3. **Not renewing session cookie after change of privilege**: Some sites set session cookie before login - if not reset at login, attacker can try to gain access to an account by injecting into victim's browser an unauthenticated session cookie chosen by the attacker, which the victim then elevates through login - **session fixation**

7.10 Cross-Site Request Forgery

Include links aimed at creating unintended side effect which might be executed as browser has a valid session cookie

Countermeasures

1. Check transaction sent as POST
2. Check *Referer:* header where browsers report on which URL a link was clicked, if it shows expected form-page URL
3. Include an invisible MAC of the session cookie into security-critical form fields - adversary unable to anticipate this, must verify that the field has the expected value
4. Short lived sessions

7.11 Website Compromise Solutions

1. Separate strong passwords for each website
2. Store passwords in central server - eg LDAP and Kerberos - but compromised web sites log all passwords entered
3. Single Sign-On