SQL injections

Current Web Environment

- Current "Web pages" are really Web applications
  - Front-end which may run in browser
  - Server provides execution environment
  - Back-end which provides services
  - Database for persistent storage
- Interfaces connect the different subsystems
  - E.g. HTTP, REST, WebSocket, SQL, etc..
- Multiple technologies and languages used
  - E.g. Javascript, PHP, HTML, CSS

Current Web Environment

- Each subsystem may be vulnerable to attacks
  - Entire application may be compromised if single breach is found
- SQL Injections are just one case
  - Focus in applications using SQL servers
  - There are many other attacks

What?

- Conjunction of several things:
  - Specially crafted input
  - Lack of sanity checks in code
- Injection of an SQL statement into another SQL statement
  - Changing its original purpose
- Most frequent vector: attacker injects special SQL statement into text field

SQL injection

Form provides two values: login and password

Typical validation query:

```
SELECT user FROM users WHERE user='$login' AND password='$password'
```

For login=admin and password=1234, query becomes:

```
SELECT user FROM users WHERE user='admin' AND password='1234'
```

SQL injection: detection

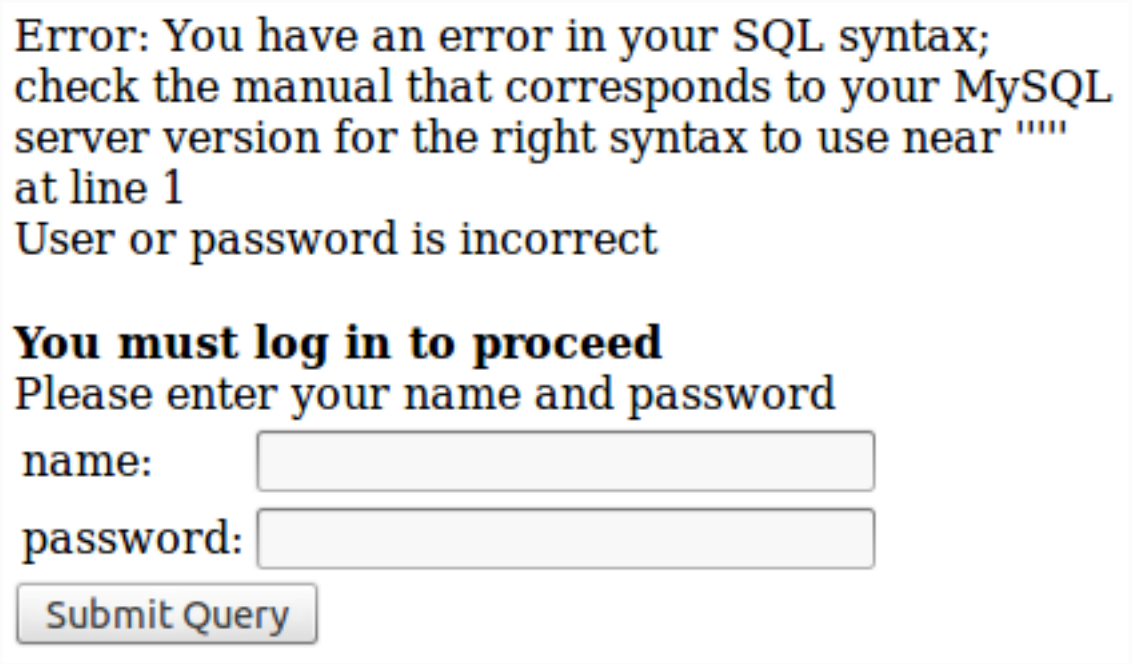
Form provides two values: login and password

What if password is a single quote? '

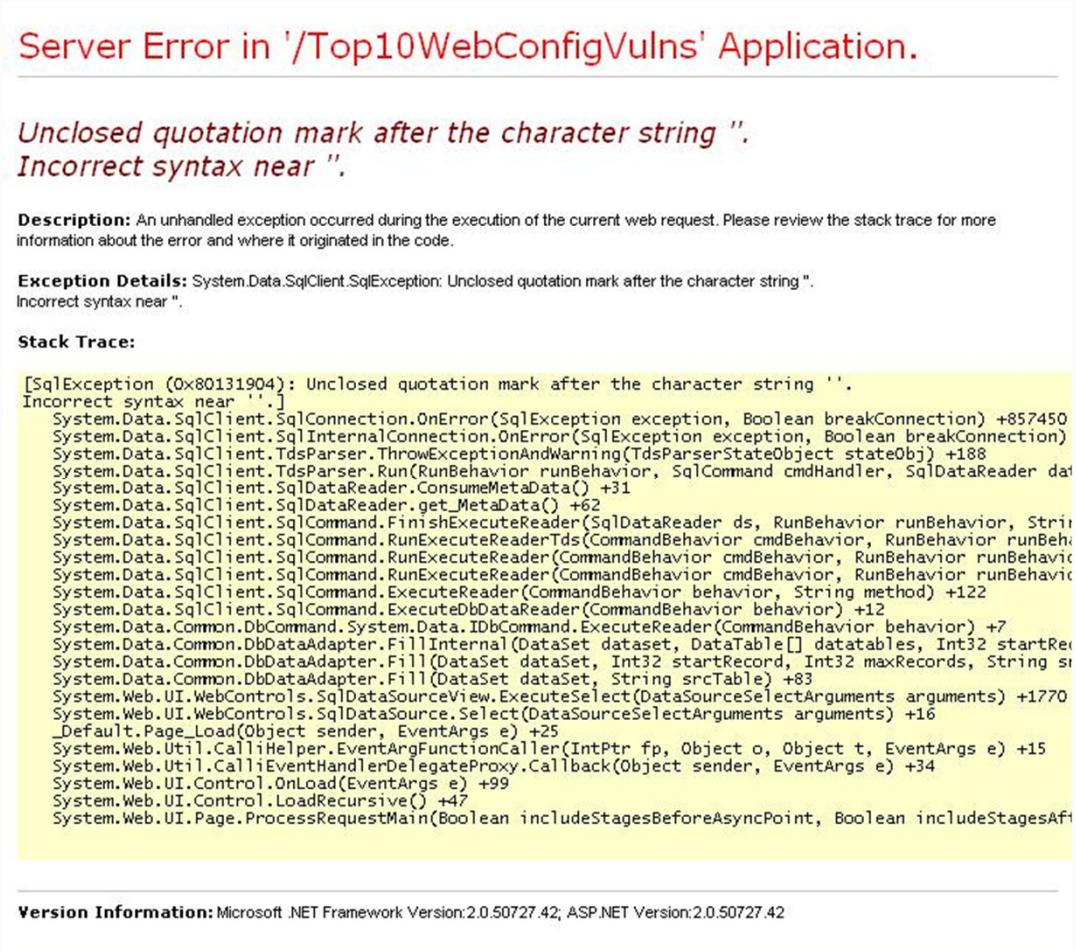
For login=admin and password=', query becomes:

```
SELECT user FROM users WHERE user='admin' AND password='''
```

SQL injection: detection



SQL Injection: Detection



Server Error in '/Top10WebConfigVulns' Application.

Unclosed quotation mark after the character string ''.
Incorrect syntax near ''.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Unclosed quotation mark after the character string ''.
Incorrect syntax near ''.

Stack Trace:

```
[SqlException (0x80131904): Unclosed quotation mark after the character string ''.
Incorrect syntax near ''.]
   System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection) +857450
   System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection)
   System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj) +188
   System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader da
   System.Data.SqlClient.SqlDataReader.ConsumeMetaData() +31
   System.Data.SqlClient.SqlDataReader.get_MetaData() +62
   System.Data.SqlClient.SqlCommand.FinishExecuteReader(SqlDataReader ds, RunBehavior runBehavior, Stri
   System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior, RunBehavior runBeh
   System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavi
   System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavi
   System.Data.SqlClient.SqlCommand.ExecuteReader(CommandBehavior behavior, String method) +122
   System.Data.SqlClient.SqlCommand.ExecuteDbDataReader(CommandBehavior behavior) +12
   System.Data.Common.DbCommand.System.Data.IDbCommand.ExecuteReader(CommandBehavior behavior) +7
   System.Data.Common.DbDataAdapter.FillInternal(DataSet dataset, DataTable[] datatables, Int32 startRe
   System.Data.Common.DbDataAdapter.Fill(DataSet dataSet, Int32 startRecord, Int32 maxRecords, String s
   System.Data.Common.DbDataAdapter.Fill(DataSet dataSet, String srcTable) +83
   System.Web.UI.WebControls.SqlDataSourceView.ExecuteSelect(DataSourceSelectArguments arguments) +1770
   System.Web.UI.WebControls.SqlDataSource.Select(DataSourceSelectArguments arguments) +16
   _Default.Page_Load(Object sender, EventArgs e) +25
   System.Web.Util.CalliHelper.EventArgFunctionCaller(IntPtr fp, Object o, Object t, EventArgs e) +15
   System.Web.Util.CalliEventHandlerDelegateProxy.Callback(Object sender, EventArgs e) +34
   System.Web.UI.Control.OnLoad(EventArgs e) +99
   System.Web.UI.Control.LoadRecursive() +47
   System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAf
```

Version Information: Microsoft .NET Framework Version:2.0.50727.42; ASP.NET Version:2.0.50727.42

http://assets.devx.com/articlefigs/1705

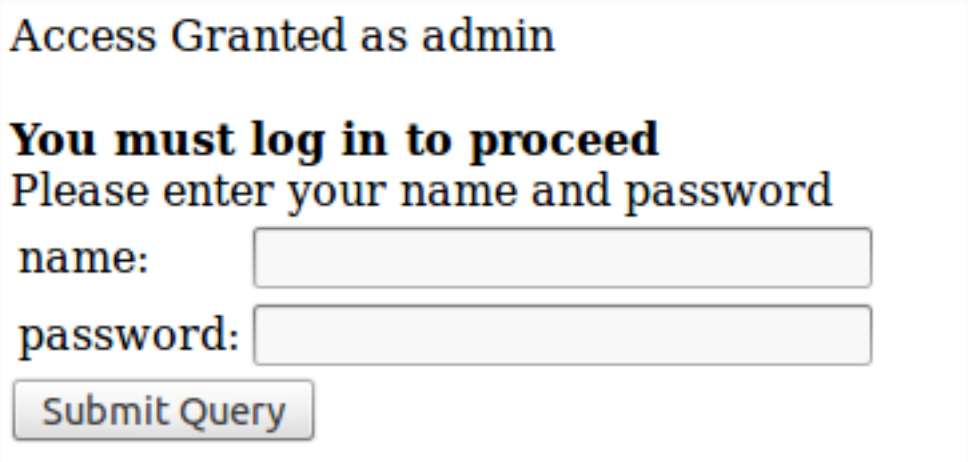
SQL injection: bypass simple password checking

- Form data is used to create an SQL statement
  - Without validation!
  - SQL code in form can be injected
- What if… password is ' or '1'='1

```
SELECT user FROM users WHERE user='admin'
       AND password='' or '1'='1'
```

- SQL statement is valid and always returns a row if the user exists

SQL injection:
bypass simple password checking



SQL Injection: bypass complex password checking

- SQL can store passwords in a ciphered format
  - Uses the PASSWORD function
  - Password stored in database cannot be obtained

- Typical validation query:

```
SELECT user FROM users WHERE user='$login'
    AND password=PASSWORD('$password')
```

- For `login=admin` and `password=') OR ('1'='1`, the query becomes:

```
SELECT user FROM users WHERE user='admin'
    AND password=PASSWORD('') OR ('1'='1')
```

Guess single password

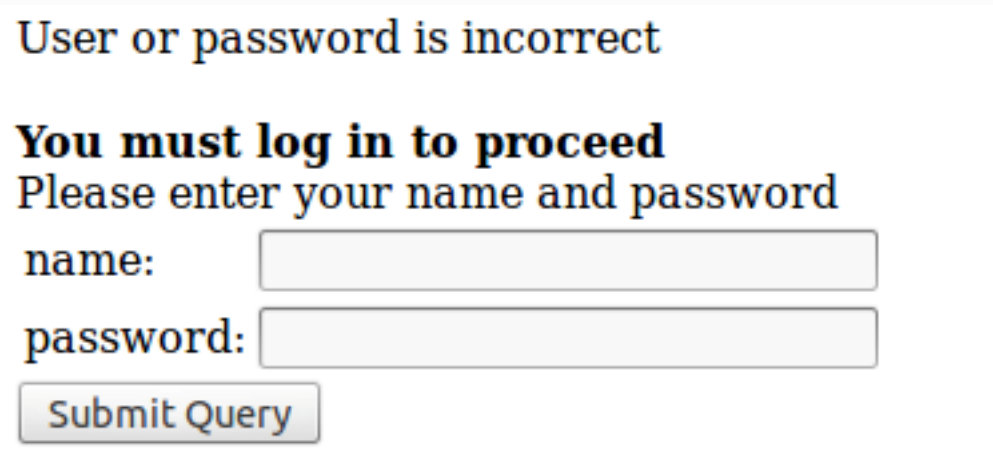
- More complex statement can be included in form fields
- Frequently, the only requirement is that they start and end with single quote (')
  - Because they will be inserted in attribute='injection'

- Does the password starts with 'a'?

```
' OR EXISTS(SELECT user FROM users
    WHERE user='admin' AND
    password LIKE 'a%') AND ''='
```

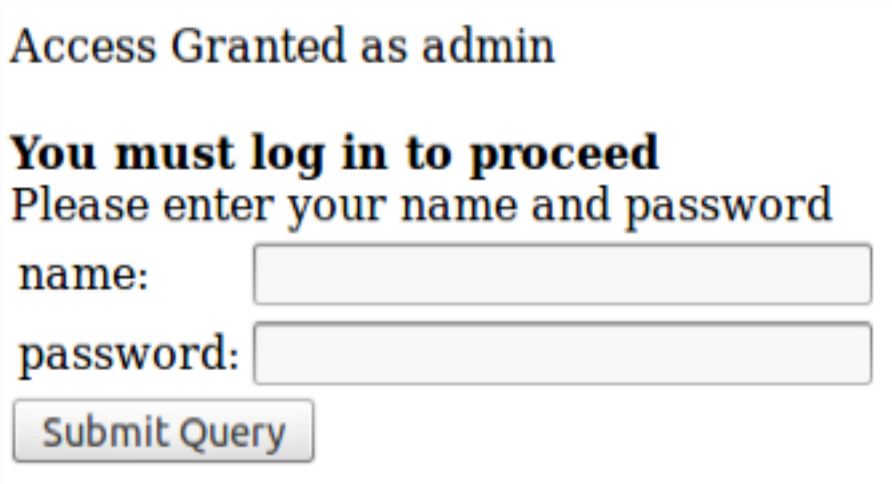
Guess simple password

```
SELECT user FROM users WHERE user='admin' AND
password='' OR EXISTS(SELECT user FROM users WHERE
user='admin' AND password LIKE 'a%') AND '' =''
```



Guess simple password

```
SELECT user FROM users WHERE user='admin` AND
password='' OR EXISTS(SELECT user FROM users WHERE
user='admin' AND password LIKE 'p%') AND '' =''
```



- Then we could try: **pa%** or **pa%a%**, etc..

Other possibilities

- Find table name:
  - Is there a users table in the current db?:

```
' OR EXISTS(SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE
TABLE_SCHEMA='test' AND TABLE_NAME='users') AND ''='
```

  - Is there any table starting by "p" in any db? :

```
' OR (SELECT COUNT(*) FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA LIKE 'p%')>1 AND ''='
```

- Find database name
  - Starts by t?:

```
' OR EXISTS(SELECT 1 FROM users WHERE database() LIKE
't%') AND ''='
```

- Find columns, get columns by index, etc…

SQL injection: terminate query

- Two characters are particularly important
  - ; Terminates current query
    - Allows multiple queries in same request
  - -- terminates processing of all queries
    - Ignores syntax errors which may appear

Query 1

```
SELECT user FROM users WHERE user='admin' AND password='
```

`'; DROP TABLE user;`	`--'`
Query 2	**Ignored after --**

Mitigation: sanitize input data

- Sanitize form input data
  - Filter out dangerous characters
    - Username can only have letters
    - Passwords can only have letters and numbers
    - Emails must comply with RFC 2822
  - Escape dangerous characters
    - Avoid this
- Browser using Javascript
  - Can be bypassed doing direct queries or using tampering proxies
    - e.g. WebScarab
  - Automated tools can easily detect and bypass such methods
    - e.g. WebCruiser
- Server
  - Higher load in server
  - Much more effective!

Mitigation: sanitize input data

- Sanitizing input data based on quotes is insufficient!
  - If form is numeric, no quote is required.
  - e.g. PIN validation

```
SELECT user FROM users WHERE
   user='admin' AND pin=12 or 1=1
```

- Validation must take in consideration actual data
  - Sanitize as much as possible

Mitigation: sanitize input data

- Escaping doesn't really help in all cases
  - e.g. typical escape is ' → ''
- Providing ' OR '1'='1 results in:

```
SELECT user FROM users WHERE user='admin' AND
password='' OR ''1''='1'
```

- The resulting query is invalid, no harm done
- What about \'; DROP TABLE users; --
  - \' is expanded to \'', '\'' is a valid string with just one character (the single quote); the table is dropped!
- MySQL provides own sanitization methods: mysql_real_escape_string()

Mitigation: prepared queries

- Instead of building query string, let SQL libraries compile the query.
  - Separation between Query and Parameters

- Three steps required:
  - Preparation
  - Bind parameters
  - Execution

Mitigation: prepared queries

- Query Preparation:

```
$s = mysql->prepare("SELECT user FROM
users WHERE user=? AND pin=?")
```

- Parameter binding:

```
$s->bind_param("s",$login);
$s->bind_param("i",$password);
```

- Query execution:

```
$s->execute();
```

Mitigation: other methods

- Limit data permissions according to user needs
  - Do not grant DROP, or Write methods for read-only application
- Use stored procedures
- Configure error reporting appropriately
  - Detailed error reporting for developers
  - Limited error reporting for users
- Isolate servers to reduce compromise of neighbor hosts

MAC spoofing & ARP poisoning

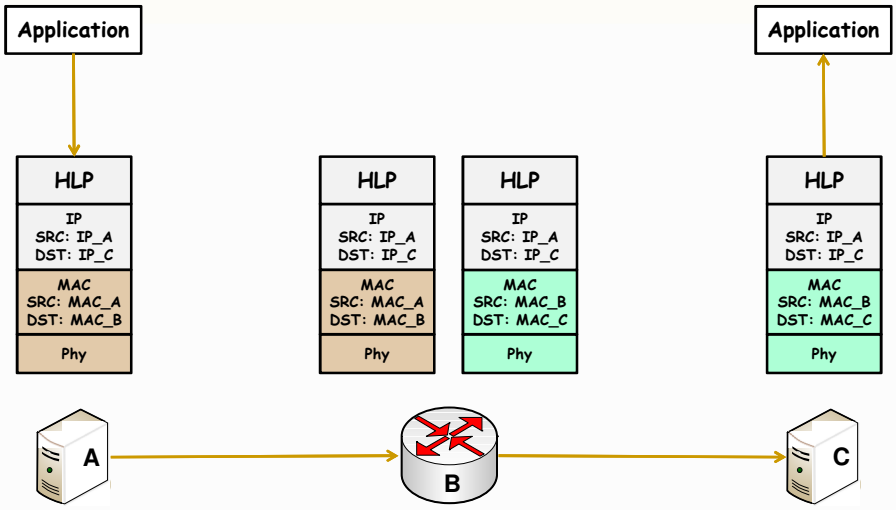
Networking Basics

- Communication in packet networks rely on several layers, with different identifiers
  - Applications use transport (TCP/UDP) ports
  - Hosts use network (IP) addresses
  - Interface Cards use MAC addresses
- Communication is typically made between applications using tuples
  - <IP_Address:Port> and a protocol (TCP, UDP, etc.)

Networking Basics

- When a packet is to be routed, two situations may occur:
  - The destination host is in the same IP network
    - The packet is sent directly to the destination host
  - The destination host is in another IP network
    - The packet is forwarded to a next hop (gateway)
- In both cases, the packet is transmitted between physical interfaces
  - Destination host or gateway

Networking Basics



Networking Basics

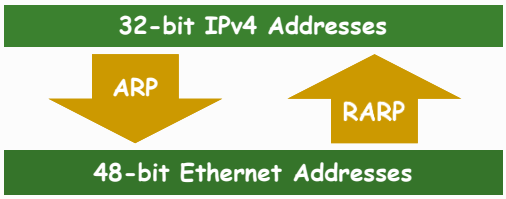
- IP addresses do not change between source and destination
  - End-to-end addressing
- MAC addresses are valid for a single network segment
  - When a packet is routed, the MAC address of the next hop must be found

IP to MAC mapping

- Static configuration
  - MAC entries of all hosts configured statically
    - All hosts "know" the MAC address of all interfaces of all other hosts
  - Doesn't scale!
    - Changing a single interface requires updating all other hosts
- Dynamic configuration
  - ARP (Address Resolution Protocol)

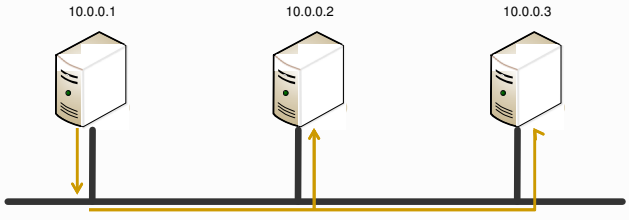
Address Resolution Protocol (RFC 826)

- ARP
  - Find the MAC address of an interface which is in a host with a given IP address
- RARP
  - Finds the IP address of host having an interface with a given MAC



Address Resolution Protocol

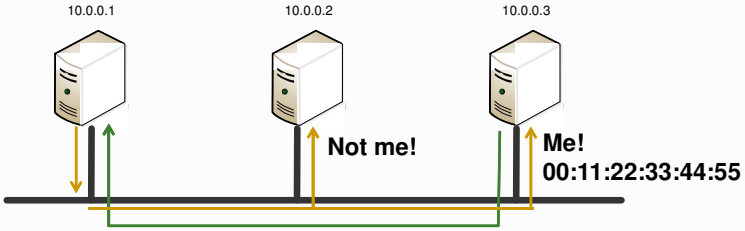
- Send ARP Request using broadcast



Who has 10.0.0.3?

Address Resolution Protocol

- Reply using ARP Response using unicast



Address Resolution Protocol

- Every packet sent requires two MAC address
  - Source Address is known
  - Destination Address must be determined
- ARP Cache increases performance
  - Caches both known and unknown entries
  - Avoids repeating the discovery process per packet
  - Entries have a large lifetime
    - 2 minutes

ARP Cache

```
$ arp -a
fog.av.it.pt     (193.136.92.154) at 00:1e:8c:3e:6a:a6 [ether] on eth0
atnog.av.it.pt   (193.136.92.123) at 00:15:17:e6:6f:67 [ether] on eth0
guarani.av.it.pt (193.136.92.134) at 00:0c:6e:da:19:87 [ether] on eth0
aeolus.av.it.pt  (193.136.92.136) at bc:ae:c5:1d:c6:53 [ether] on eth0
```


MAC Spoofing

- MAC addresses can be modified
  ifconfig eth0 hw ether 00:11:22:33:44:55
- Using a colliding MAC address will allow the reception of network traffic for other hosts
  - Some switches limit MAC addresses to single ports
- Sending ARP packets with spoofed addresses may poison the cache of other stations
  - ARP Poisoning

ARP Poisoning

- Hosts cache information directly from all packets received
  - Besides ARP packets
  - No other verification is done
- New information will replace existing entries
  - Great for allowing network dynamism
  - Very bad for security
- It is possible to send specially crafted packets to create specific entries in remote hosts

ARP Poisoning

- When receiving an ARP Request:

```
Frame 10: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: Apple_1b:1f:42 (e0:f8:47:1b:1f:42), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Apple_1b:1f:42 (e0:f8:47:1b:1f:42)
    Sender IP address: 10.0.0.3 (10.0.0.3)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.0.2 (10.0.0.2)
```

- 10.0.0.2 will send an ARP Response
- But… 10.0.0.2 will also “learn” that 10.0.0.3 is at e0:f8:47:1b:1f:42

ARP Poisoning

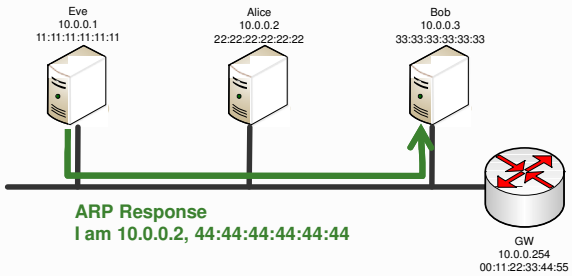
- When receiving an ARP Response

```
▸ Frame 123: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: Tp-LinkT_f2:77:62 (90:f6:52:f2:77:62), Dst: Apple_1b:1f:42 (e0:f8:47:1b:1f:42)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Tp-LinkT_f2:77:62 (90:f6:52:f2:77:62)
    Sender IP address: 10.0.0.246 (10.0.0.246)
    Target MAC address: Apple_1b:1f:42 (e0:f8:47:1b:1f:42)
    Target IP address: 10.0.0.3 (10.0.0.3)
```

- 10.0.0.3 will learn that 10.0.0.246 is at 90:f6:52:f2:77:62
- …. even if no matching request has been made…
  - Gratuitous ARP

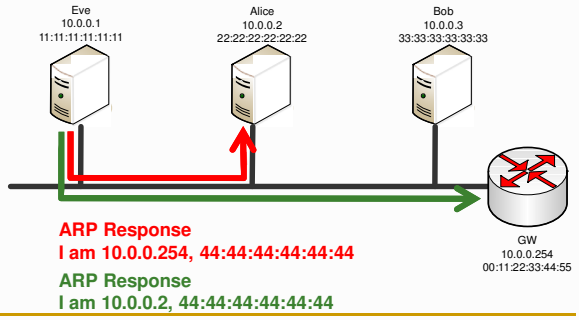
ARP Poisoning: Consequences

- Hosts can be isolated from the network
  - Create fake useless entries for other hosts
- Bob will be fooled to use 44:44:44:44:44:44 for reaching Alice
  - But there's no one with that MAC address …
  - Alice gets isolated



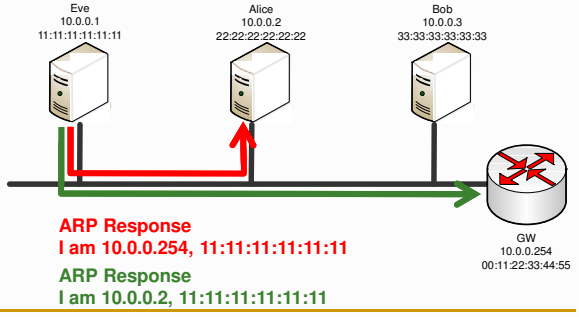
ARP Poisoning: Consequences

- Hosts can be denied communication with the outside world
- Alice an the GW will be fooled about each other
  - Alice gets isolated



ARP Poisoning: Consequences

- Interception of traffic between hosts (MitM)
  - Alice→GW and GW→Alice traffic goes to Eve
- Then Eve can forward the traffic back and forth
  - Or not …



ARP Poisoning: Avoidance

- Static entries
  - No resolution process is triggered
  - Colliding information from ARP packets is discarded
- Port-based packet filtering at switch ingress
  - Spoofed ARP packets are dropped
  - Only possible in static scenarios
- Network segregation
  - VLANs, WiFi client segregation

ARP Poisoning: Avoidance

- Behavior detection w/ monitoring software
  - Detect ARP Responses without Request
  - Detect repeated Requests from same host
  - Detect MAC changes
  - Network administrator is notified
    - But ARP poisoning is not actually avoided!
    - And it may be difficult to find the attacker's host

Authentication protocols

Identity attributes

- Set of attributes for setting apart individuals
  - Name
  - Numerical identifiers
    - Fixed for life
    - Variable with context
  - Address
  - Photo
  - Identity of relatives
    - Usually parents
  - …

Authentication: Definition

- Proof that an entity has a claimed identity attribute
  - Hi, I'm Joe
  - Prove it!
  - Here are my Joe's credentials
  - Credentials accepted/not accepted
  - Hi, I'm over 18
  - Prove it!
  - Here is the proof
  - Proof accepted/not accepted

Authentication: proof types

- Something we know
  - A secret memorized (or written down…) by Joe
- Something we have
  - An object/token solely held by Joe
- Something we are
  - Joe's Biometry
- Multi-factor authentication
  - Join or consecutive use of different proof types

Multi-factor verification jokes

Authentication: goals

- Authenticate interactors
  - People, services, servers, hosts, networks, etc.
- Enable the enforcement of authorization policies and mechanisms
  - Authorization ⇒ authentication
- Facilitate the exploitation of other security-related protocols
  - e.g. key distribution for secure communication

Authentication: requirements

- Trustworthiness
  - How good is it in proving the identity of an entity?
  - How difficult is it to be deceived?
  - Level of Assurance (LoA) (NIST, eIDAS, ISO 29115)

    LoA 1 - Little or no confidence in the asserted identity
    LoA 2 - Some confidence in the asserted identity
    LoA 3 - High confidence in the asserted identity
    LoA 4 - Very high confidence in the asserted identity
- Secrecy
  - No disclosure of secrets used by legitimate entities

Authentication: requirements

- Robustness
  - Prevent attacks to the protocol data exchanges
  - Prevent on-line DoS attack scenarios
  - Prevent off-line dictionary attacks
- Simplicity
  - It should be as simple as possible to prevent entities from choosing dangerous shortcuts
- Deal with vulnerabilities introduced by people
  - They have a natural tendency to facilitate or to take shortcuts

Authentication: Entities and deployment model

- Entities
  - People
  - Hosts
  - Networks
  - Services / servers
- Deployment model
  - Along the time
    - Only when interaction starts
    - Continuously along the interaction
  - Directionality
    - Unidirectional
    - Bidirectional (or mutual)

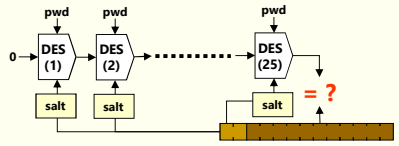
Authentication interactions: Basic approaches

- Direct approach
  - Provide credentials
  - Wait for verdict
  - Authenticator checks credentials against what it knows
- Challenge-response approach
  - Get challenge
  - Provide a response computed from the challenge and the credentials
  - Wait for verdict
  - Authenticator checks response for the challenge provided and the credentials it knows

Authentication of people: Direct approach w/ known password

- A password is matched with a stored value
  - For a claimed identity (username)
- Personal stored value:
  - Transformed by a unidirectional function
    - Key Derivation Function (KDF)
    - Preferably slow!
    - Bcrypt, scrypt, Argon2, PBKDF2
  - UNIX: DES hash + salt
  - Linux: KDF + salt
  - Windows: digest function

DES hash = $DES_{pwd}^{25}(0)$
$DES_k^n(x) = DES_k(DES_k^{n-1}(x))$
Permutation of 12 subkeys´bit pairs with salt (12 bits)



Authentication of people: Direct approach w/ known password

- Advantage
  - Simplicity!
- Problems
  - Usage of predictable passwords
    - They enable dictionary attacks
  - Different passwords for different systems
    - To prevent impersonation by malicious admins
    - But our memory has limits!
  - Exchange along insecure communication channels
    - Eavesdroppers can easily learn the password
    - e.g. Unix remote services, PAP

Top 15 2019 by Splashdata

1 - 123456
2 - 123456789
3 - qwerty
4 - password
5 - 1234567
6 - 12345678
7 - 12345
8 - iloveyou
9 - 111111
10 - 123123
11 - abc123
12 - qwerty123
13 - 1q2w3e4r
14 - admin
15 - qwertyuiop

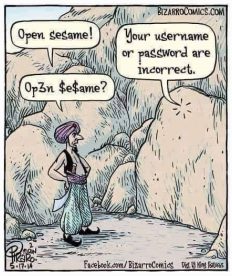
source: https://www.teampassword.com/blog/top-50-worst-passwords-of-2019
Image https://www.pinterest.com/networkboxusa/it-humor

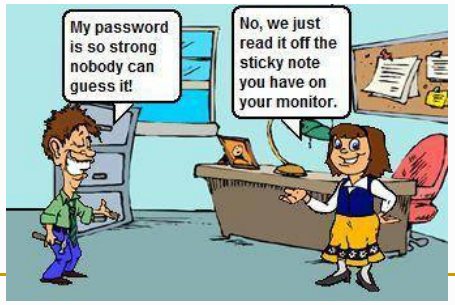
Password selection jokes

Dear IT,

the more "secure" you try to make our passwords by making them impossible to remember, the more likely I am to save them all in a big word doc named "Passwords"

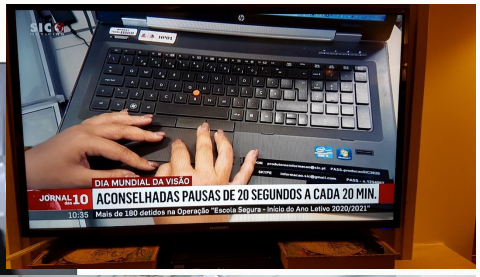
Signed,
Everyone





Sorry, but your password must contain an uppercase letter, a number, a haiku, a gang sign, a hieroglyph, and the blood of a virgin.

Password bloopers



Authentication of people: Direct approach with biometrics

- People get authenticated using body measurements
  - Biometric samples or features
  - Common modalities
    - Fingerprint
    - Facial recognition
    - Palm print
    - Iris scan
    - Voice recognition
    - DNA
- Measures are compared with personal records
  - Biometric references (or template)
  - Registered in the system with a previous enrolment procedure

Biometrics: advantages

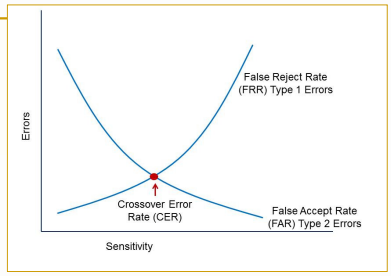
- Convenient: people do not need to use memory
  - Just be their self
- People cannot chose weak passwords
  - In fact, they don't chose anything
- Credentials cannot be transferred to others
  - One cannot delegate their own authentication
- Stealth identification
  - Interesting for security surveillance

Biometrics: problems

- Usability
  - Comfort of people, ergonomic
  - Exploitation scenario
- Biometrics are still being improved
  - In many cases they can be easily cheated
  - Liveness detection
- People cannot change their credentials
  - Upon their robbery
- It can be risky for people
  - Removal of body parts for impersonation of the victim

Image source: https://biometrics.mainguet.org/types/tongue.htm

Biometrics: problems



- Sensitivity tuning
  - Reduction of FRR (annoying)
  - Reduction of FAR (dangerous)
  - Tuning is mainly performed with the target population
    - Not with attackers!
- Not easy to deploy remotely
  - Requires trusting the remote sample acquisition system
- Can reveal personal sensitive information
  - Diseases
- Credentials cannot be (easily) copied to others
  - In case of need in exceptional circumstances

Image source: http://www.pearsonitcertification.com/articles/article.aspx?p=1718488

Authentication of people: Direct approach with OTPs

- One-time password (OTP)
  - Credential that can be used only once
- Advantage
  - OTPs can be eavesdropped
  - Eavesdroppers cannot impersonate the OTP owner
    - True for passive eavesdroppers
    - False for active attackers!

Authentication of people: Direct approach with OTPs

- Problems
  - Interactors need to know which password they should use at different occasions
    - Requires some form of synchronization
  - People may need to use extra resources to maintain or generate one-time passwords
    - Paper sheets
    - Computer programs
    - Special devices, etc.

Authentication of people: OTPs and secondary channels

- OTPs are codes sent through secondary channels
  - A secondary channel is a channel that is not the one were the code is going to be used
    - SMS, email, Twitter, Firebase, QR codes, NFC, etc.
  - The secondary channel provides the synchronization
    - Just-in-time provision of OTP
- Two authentications are possible
  - Confirm a secondary channel provided by a profile owner
    - In order to trust that that channel belongs to the profile owner
  - Authenticate the owner of a profile
    - Which is bound to a secondary channel

Authentication of people: OTPs produced from a shared key

- HOTP (Hash-based One Time Password, RFC 4226)
  - OTP generated from a counter and a shared key
  - Counters are updated independently
- TOTP (Time-based One Time Password, RFC 6238)
  - OTP generated from a timestamp and a shared password
  - TOTP is HOTP with timestamps instead of counters
  - Clocks need a rough synchronization

Token-based OTP generators: RSA SecurID



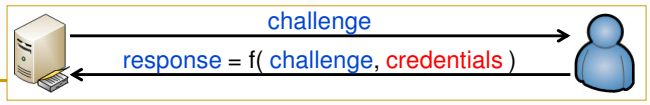
- Personal authentication token
  - Or software modules for handhelds (PDAs, smartphones, etc.)
- It generates a unique number at a fixed rate
  - Usually one per minute (or 30 seconds)
  - Bound to a person (User ID)
  - Unique number computed with:
    - A 64-bit key stored in the token
    - The actual timestamp
    - A proprietary digest algorithm (SecurID hash)
    - An extra PIN (only for some tokens)

RSA SecurID

- OTP-based authentication
  - A user combines their User ID with the current token number
    OTP = User ID, Token Number
- An RSA ACE Server does the same and checks for match
  - It also knows the person's key stored in the token
  - There must be a synchronization to tackle clock drifts
    - RSA Security Time Synchronization
- Robust against dictionary attacks
  - Keys are not selected by people

Challenge-response approach: Generic description

- The authenticator provides a challenge
- The entity being authenticated transforms the challenge
  - With its authentication credentials
- The result (response) is sent to the authenticator
- The authenticator checks the response
  - Produces a similar result and checks if they match
  - Transforms the result and checks if it matches the challenge or a related value



Challenge-response approach: Generic description

- Advantage
  - Authentication credentials are not exposed
- Problems
  - People may require means to compute responses
    - Hardware or software
  - The authenticator may have to have access to shared secrets
    - How can we prevent them from using the secrets elsewhere?
  - Offline dictionary attacks
    - Against recorded challenge-response dialogs
    - Can reveal secret credentials (passwords, keys)

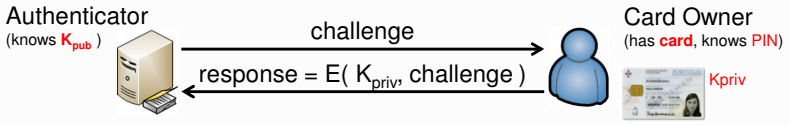
Challenge-response protocols: selection of challenges

- Challenges cannot be repeated for the same entity
  - Same challenge → same response
  - An active attacker can impersonate a user using a previously recorded protocol run
- Challenges should be nonces
  - Nonce: number used only once
  - Stateful services can use counters
  - Stateless services can use (large) random numbers
  - Time can be used, but with caution
    - Because one cannot repeat a timestamp

Authentication of people: Challenge-response with smartcards

- Authentication credentials
  - The smartcard
    - e.g. Citizen Card
  - The private key stored in the smartcard
  - The PIN to unlock the private key
- The authenticator knows
  - The corresponding public key
  - Or some personal identifier
    - Which can be related with a public key through a (verifiable) certificate

Authentication of people: Challenge-response with smartcards



- Signature-based protocol
  - The authenticator generates a random challenge
    - Or a value not used before
  - The card owner ciphers the challenge with their private key
    - PIN-protected
  - The authenticator decrypts the result with the public key
    - If the output matches the challenge, the authentication succeeds
- Encryption-based protocol
  - Possible when private key decryption is available

Authentication of people: Challenge-response with memorized password

- Authentication credentials
  - Passwords selected by people
- The authenticator knows
  - All the registered passwords; or
  - A transformation of each password
    - Preferable option
    - Preferably combined with some local value (salt)
    - Preferable using a tunable function (e.g. iterations)

Authentication of people:
Challenge-response with memorized password

- The authenticator generates a random challenge
- The person computes a function of the challenge and password
  - e.g. a joint digest: response = digest (challenge, password)
  - e.g. an encryption response = $E_{password}$ (challenge)
- The authenticator does the same (or the inverse)
  - If the output matches the response (or the challenge), the authentication succeeds
- Examples
  - CHAP, MS-CHAP v1/v2, S/Key

PAP & CHAP
(RFC 1334, 1992, RFC 1994, 1996)

- Protocols used in PPP (Point-to-Point Protocol)
  - Unidirectional authentication
    - Authenticator is not authenticated
- PPP developed in 1992
  - Mostly used for dial-up connections
- PPP protocols are used by PPTP VPNs
  - e.g. vpn.ua.pt

PAP & CHAP
(RFC 1334, 1992, RFC 1994, 1996)

- PAP (PPP Authentication Protocol)
  - Simple UID/password presentation
  - Insecure cleartext password transmission
- CHAP (CHallenge-response Authentication Protocol)

  Aut → U: authID, challenge
  U → Aut: authID, MD5(authID, pwd, challenge), identity
  Aut → U: authID, OK/not OK
  - The authenticator may require a reauthentication anytime

MS-CHAP (Microsoft CHAP)
(RFC 2433, 1998, RFC 2759, 2000)

- Version 1

  A → U: authID, **C**
  U → A: **R1**, **R2**
  A →U: OK/not OK

  $R1 = DES_{LMPH}(C)$
  $R2 = DES_{NTPH}(C)$

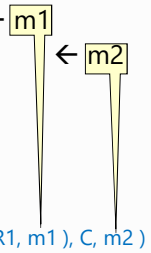
  LMPH = DEShash(pwd')
  NTPH = MD4(pwd)

  pwd' = capitalized(pwd)

- Version 2

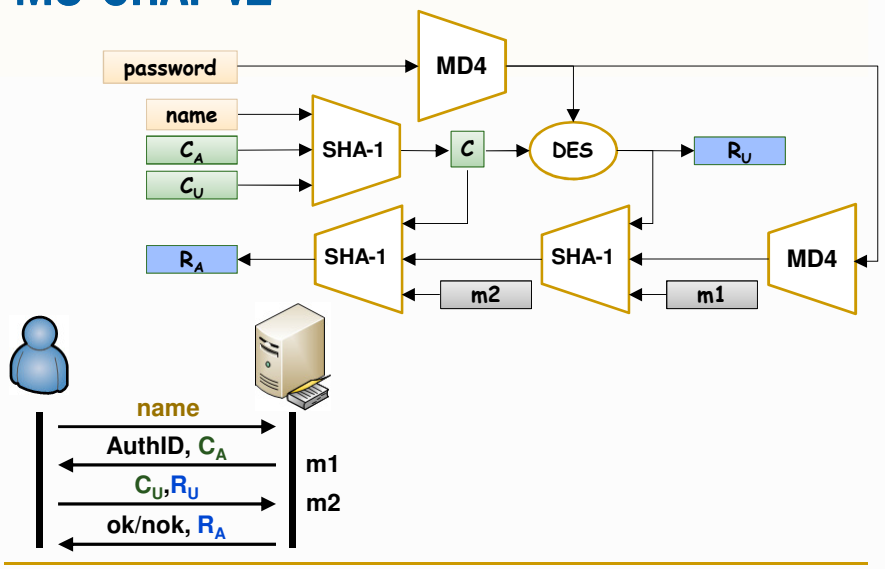
  A → U: authID, $\mathbf{C_A}$
  U → A: $\mathbf{C_U}$, **R1**
  A → U: OK/not OK, **R2**

  $R1 = DES_{PH}(C)$
  $C = SHA(C_U, C_A, username)$
  PH = MD4(password)
  R2 = SHA(SHA(MD4(PH), R1, m1), C, m2)



- Mutual authentication
- Passwords can be updated

MS-CHAP v2

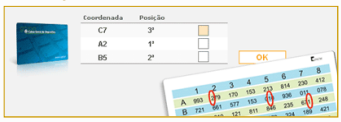


Authentication of people: Generation of OTPs with challenges

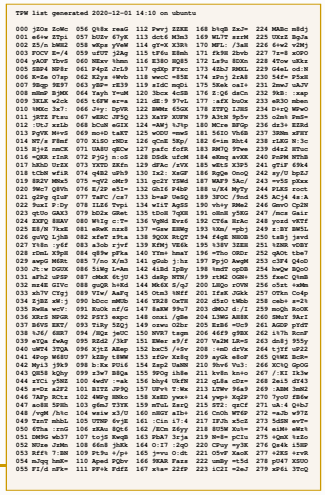
- OTPs can be produced from a challenge received
  - The fundamental protocol is password-based
    - But passwords are OTPs
  - OTPs are produced from a challenge
  - One can use several algorithms to handle OTPs

Authentication of people: OTPs selected from shared data

- Advantage:
  - Shared data can be random
  - No long-term short secrets to protect
- OTPs build from printed data
  - Example: online bank codes



- Selection of an OTP from a printed / saved list



S/Key (RFC 2289, 1998)

- Authentication credentials
  - A password (pwd)
- The authenticator knows
  - The last used one-time password (OTP)
  - The last used OTP index
    - Defines an order among consecutive OTPs
  - An seed value for the each person's OTPs
    - The seed is similar to a UNIX salt

S/Key setup

- The authenticator defines a random seed
- The person generates an initial OTP as:

  $OTP_n = h^n$ (seed, pwd), where h = MD4

  - Some S/Key versions also use MD5 or SHA-1
- The authenticator stores seed, n and OTP_n as authentication credentials

S/Key authentication protocol

- Authenticator sends seed & index of the person
  - They act as a challenge
- The person generates index-1 OTPs in a row
  - And selects the last one as result
  - result = $OPT_{index-1}$
- The authenticator computes h (result) and compares the result with the stored OPT_{index}
  - If they match, the authentication succeeds
  - Upon success, stores the recently used index & OTP
    - index-1 and $OPT_{index-1}$

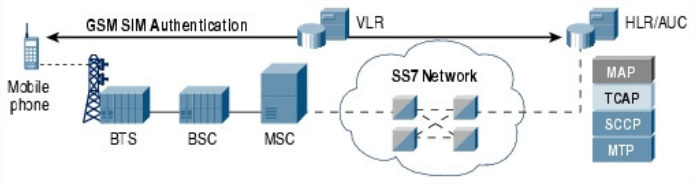
S/Key

- Advantages
  - Users passwords are unknown to authenticators
  - OTPs can be used as ordinary passwords
- Disadvantages
  - People need an application to compute OTPs
  - Passwords can be derived using dictionary attacks
    - From data stored in authenticators
    - From captured protocol runs

Authentication of people: Challenge-response with shared key

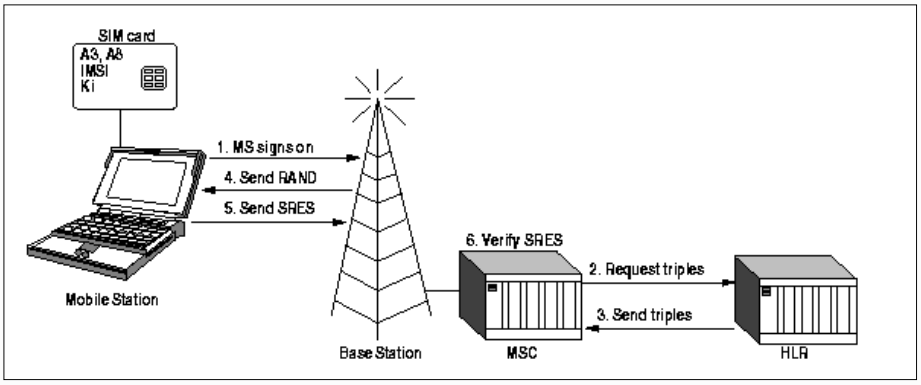
- Uses a shared key instead of a password
  - Robust against dictionary attacks
  - Requires some token to store the key
- Example:
  - GSM

GSM: authentication architecture



- Based on a secret key shared between the HLR and the station
  - 128 Ki, stored in the station's SIM card
  - Can only be used after entering a PIN
- Algorithms (initially not public):
  - A3 for authentication
  - A8 for generating a session key
  - A5 for encrypting the communication
- A3 and A8 implemented by SIM card
  - Can be freely selected by the operator

GSM: mobile station authentication



GSM: mobile station authentication

- MSC fetches trio from HLR
  - RAND, SRES, Kc
  - In fact more than one are requested
- HLR generates RAND and corresponding trio using subscriber's Ki
  - RAND, random value (128 bits)
  - SRES = A3 (Ki, RAND) (32 bits)
  - Kc = A8 (Ki, RAND) (64 bits)
- Usually operators use COMP128 for A3/A8
  - Recommended by the GSM Consortium
  - [SRES, Kc] = COMP128 (Ki, RAND)

Host authentication

- By name or address
  - DNS name, IP address, MAC address, other
  - Extremely weak, no cryptographic proofs
    - Nevertheless, used by many services
    - e.g. NFS, TCP *wrappers*
- With cryptographic keys
  - Keys shared among peers
    - With an history of usual interaction
  - Per-host asymmetric key pair
    - Pre-shared public keys with usual peers
    - Certified public keys with any peer

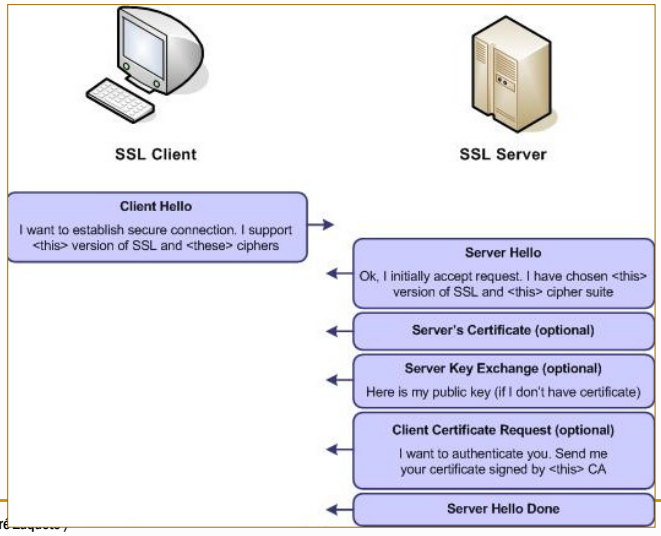
Service / server authentication

- Host authentication
  - All co-located services/servers are indirectly authenticated
- Per-service/server credentials
  - Shared keys
    - When related with the authentication of people
    - The key shared with each person can be used to authenticate the service to that person
  - Per-service/server asymmetric key pair
    - Certified or not

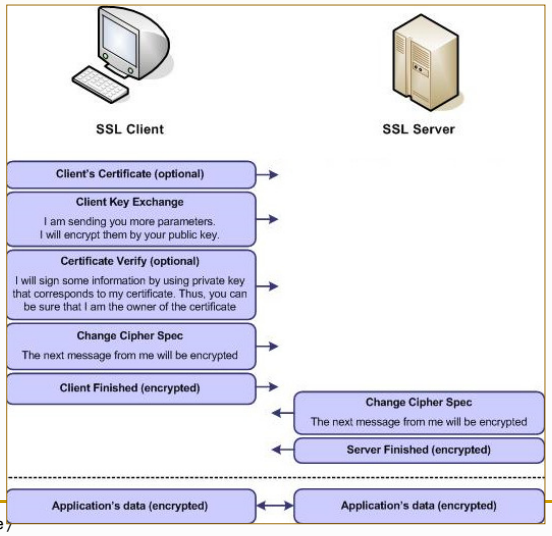
TLS (Transport Layer Security, RFC 5246)

- Secure communication protocol over TCP/IP
  - Created upon SSL V3 (Secure Sockets Layer)
  - Manages per-application secure sessions over TCP/IP
    - Initially conceived for HTTP traffic
    - Actually used for other traffic types
- There is a similar version for UDP (DTLS, RFC 6347)
- Security mechanisms
  - Communication confidentiality and integrity
    - Key distribution
  - Authentication of communication endpoints
    - Servers (or, more frequently, services)
    - Client users
    - Both with asymmetric key pairs and certified public keys

SSL/TLS interaction diagrams (1st part)



SSL/TLS interaction diagrams (2nd part)



SSH (Secure Shell, RFC 4251)

- Alternative to telnet/rlogin protocols/applications
  - Manages secure consoles over TCP/IP
  - Initially conceived to replace telnet
  - Actually used for other applications
    - Secure execution of remote commands (rsh/rexec)
    - Secure copy of contents between machines (rcp)
    - Secure FTP (sftp)
    - Creation of arbitrary secure tunnels (inbound/outbound/dynamic)
- Security mechanisms
  - Communication confidentiality and integrity
    - Key distribution
  - Authentication of communication endpoints
    - Servers / machines
    - Client users
    - Both with different techniques

SSH authentication mechanisms

- Server: with asymmetric keys pair
  - Inline public key distribution
    - Not certified!
  - Clients cache previously used public keys
    - Caching should occur in a trustworthy environment
    - Update of a server's key raises a problem to its usual clients
- Client users: configurable
  - Username + password
    - By default
  - Username + private key
    - Upload of public key in advance to the server

Authentication metaprotocols

- Generic authentication protocols that encapsulate other specific authentication protocols
- Examples
  - EAP (Extensible Authentication Protocol)
    - Used in 802.11 (Wi-Fi)
  - ISAKMP(Internet Security Association and Key Management Protocol)
    - Used in IPSec

Single Sign-On (SSO)

- Unique, centralized authentication for a set of federated services
  - The identity of a client, upon authentication, is given to all federated services
  - The identity attributes given to each service may vary
  - The authenticator is called Identity Provider (IdP)
- Examples
  - SSO authentication at UA
    - Performed by a central IdP (idp.ua.pt)
    - The identity attributes are securely conveyed to the service accessed by the user

Authentication services

- Trusted third parties (TTP) used for authentication
  - But often combined with other related functionalities
- AAA services
  - Authentication, Authorization and Accounting
  - e.g. RADIUS

PAM
(Pluggable Authentication Modules)

Motivation

- Users
  - Unification of authentication mechanisms for different applications
- Manufacturers
  - Authenticated access to services independent of authentication mechanisms
- Administrators
  - Easy orchestration of authentication mechanisms different services requiring client authentication
  - Flexibility to configure specific authentication mechanisms for each host
- Manufacturers and Administrators
  - Flexible and modular approach for integrating novel authentication mechanisms

PAM: features

- Independent authentication protocols / mechanisms
  - Linux password, S/Key, smartcards, biometrics, etc.
  - One module per protocol / mechanism
- Orchestration of protocols / mechanisms
  - Alone or combined
  - AND and OR combinations
  - Application-independent
- Several interface approaches
  - Input from text consoles of graphical windows
  - Access to special devices (smart-cards, biometric readers, etc.)

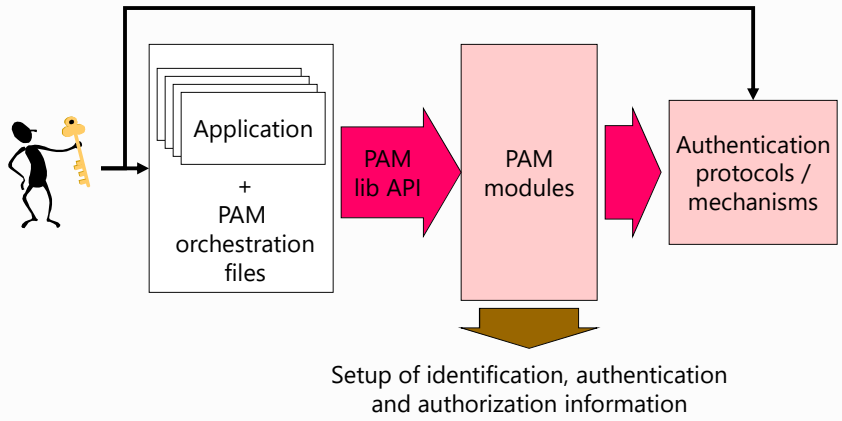
PAM: features

- Modular and extensible architecture
  - Dynamic loading of required modules
  - Handling of several actions besides authentication
    - Password management
    - Accounting management
    - Session management
- Default orchestration per host
  - Defined by the administrator
    - Username/password, biometrics, smart-cards, etc.
- Application-specific orchestrations
  - Each application can use a unique orchestration

Classic Unix authentication

- Requested input: username + password
- Validation
  - Active account for username
    - Entry with the username in the /etc/passwd file
  - Transformed password for that username
    - Entry with the username in the /etc/shadow file
  - Transformation of the provided password with the function and the salt used for that username
  - Comparison with the stored transformation
- Obtained credentials
  - UID + GID [+ list of secondary GIDs]
  - New process descriptor (login shell)

PAM: Architecture



PAM: Actions

- Authentication (auth)
  - Identity verification
- Account Management (account)
  - Enforcement of access policies based on account properties
- Password Management (password)
  - Management of authentication credentials
- Session Management (session)
  - Verification of operational parameters
  - Setup of session parameters
    - max memory, max file descriptions, graphical interface configuration, ...

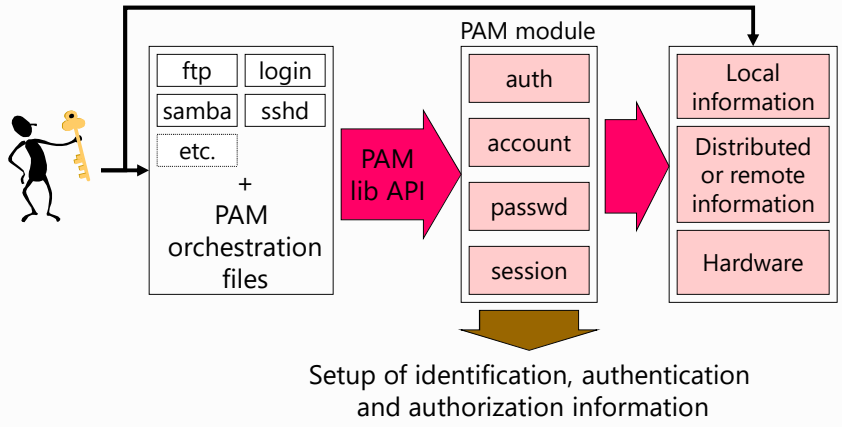
PAM: Modules

- Dynamically loaded (*shared libraries*)
  - /lib/security/pam_*.so
  - /lib/x86_64-linux-gnu/security/pam_*.so
- Standard API
  - Functions provided by the modules that are used
    - C interfaces
  - Decision provided on returned code
    - PAM_SUCCESS
    - PAM_AUTH_ERR, PAM_AUTHINFO_UNAVAIL, etc...
  - Not all functions need to be implemented
    - A module does not need to implement all 4 actions

PAM: orchestration files

- Typically, one per PAM client application
  - e.g. /etc/pam.d/ftp or /etc/pam.d/ssh
  - Can use shared files: /etc/pam.d/common-auth
- Specify how the actions should be applied
  - Their mechanisms (modules)
  - Their parameters
  - Their termination, with or without success
- Each module uses a particular set of resources
  - Local files
    - /etc/passwd, /etc/shadow, /etc/groups, etc.
  - Distributed information or located in remote servers
    - NIS, Kerberos, LDAP, etc.

PAM: Detailed Architecture



PAM APIs: PAM lib

- Start/end of the PAM lib
  pam_start(service, user name, callback, &pam_handle)
  pam_end(pam_handle, status)
- Module specific data
  pam_get_data(), pam_set_data()
  pam_get_item(), pam_set_item()
- "auth" action
  pam_authenticate(pam_handle, flags)
  pam_setcred(pam_handle, flags)
- "account" action
  pam_acct_mgmt(pam_handle, flags)
- "passwd" action
  pam_chauthtok(pam_handle, flags)
- "session" action
  pam_open_session(pam_handle, flags)
  pam_close_session(pam_handle, flags)

Orchestration of PAM actions

- Sequence of module invocations per action
  - By default, modules are executed sequentially
  - Each module has its own parameters and calling semantic
    - Required, requisite, sufficient, optional
    - [...]
  - Execution proceeds until the end, or failure
    - To better hide the source of a failure, module execution can either abort immediately or delay the failure upon executing the entire sequence
  - Applications can recover from failures

PAM APIs: PAM modules

- "auth" action
  pam_sm_authenticate(pam_handle, flags)
  pam_sm_setcred(pam_handle, flags)
- "account" action
  pam_sm_acct_mgmt(pam_handle, flags)
- "passwd" action
  pam_sm_chauthtok(pam_handle, flags)
- "session" action
  pam_sm_open_session(pam_handle, flags)
  pam_sm_close_session(pam_handle, flags)

PAM: Module invocation

- Syntax: action control module [parameters]
- Control is specified for each action and module

requisite
- If the module fails, the result is returned immediately

required
- If the module fails, the result is set but the next modules are invoked

sufficient
- If module fails the result is ignored
- Otherwise, returns success if all previous "required" modules also were successful

optional
- Result is ignored
- EXCEPT: if this is the only module in the action

[success=ok/number default=ignore/die/bad ...]

Configuration files: /etc/pam.d/login

```
auth optional pam_faildelay.so delay=3000000
auth [success=ok new_authtok_reqd=ok ignore=ignore user_unknown=bad default=die] pam_securetty.so
auth requisite pam_nologin.so

session [success=ok ignore=ignore module_unknown=ignore default=bad] pam_selinux.so close
session required pam_loginuid.so
session [success=ok ignore=ignore module_unknown=ignore default=bad] pam_selinux.so open
session required pam_env.so readenv=1
session required pam_env.so readenv=1 envfile=/etc/default/locale

@include common-auth
auth optional pam_group.so

session required pam_limits.so
session optional pam_lastlog.so
session optional pam_motd.so motd=/run/motd.dynamic
session optional pam_motd.so noupdate
session optional pam_mail.so standard
session optional pam_keyinit.so force revoke

@include common-account
@include common-session
@include common-password
```


PAM orchestration files: Advanced decision syntax

- [value=action value=action …]
- Actions:
  - ignore: take no decision
  - bad: continue, but the final decision will be a failure
  - die: terminate immediately with failure
  - ok: continue, so far the decision is success
  - done: terminate immediately with success
  - reset: clear the entire state and continue
  - N (unsigned integer): same as ok + jump over N lines

PAM orchestration files: Advanced decision syntax

- Values (return codes)
  - *success*
  - *open_err*
  - *symbol_err*
  - *service_err*
  - *system_err*
  - *buf_err*
  - *perm_denied*
  - *auth_err*
  - *cred_insufficient*
  - *authinfo_unavail*
  - *user_unknown*
  - *maxtries*
  - *new_authtok_reqd*
  - *acct_expired*
  - *session_err*
  - *cred_unavail*
  - *cred_expired*
  - *cred_err*
  - *no_module_data*
  - *conv_err*
  - *authtok_err*
  - *authtok_recover_err*
  - *authtok_lock_busy*
  - *authtok_disable_aging*
  - *try_again*
  - *ignore*
  - *abort*
  - *authtok_expired*
  - *module_unknown*
  - *bad_item*
  - *conv_again*
  - *incomplete*
  - *default*
    - *Any not specified*

PAM orchestration files: Simplified decision syntax

- High-level decisions definitions
  - requisite
    - [success=ok new_authtok_reqd=ok ignore=ignore default=die]
  - required
    - [success=ok new_authtok_reqd=ok ignore=ignore default=bad]
  - sufficient
    - [success=done new_authtok_reqd=ok default=ignore]
  - optional
    - [success=ok new_authtok_reqd=ok default=ignore]

Access control models

Access types

- Physical access
  - Physical contact between a subject and the object of interest
    - Facility, room, network, computer, storage device, authentication token, etc.
  - Out of scope of this course …
- Informatic or electronic access
  - Information-oriented contact between a subject and the object of interest
    - Contact through request-response dialogs
  - Contact is mediated by
    - Computers and networks
    - Operating systems, applications, middleware, devices, etc.

Access control

- Definition
  - The policies and mechanisms that mediate the access of a subject to an object
- Interaction model

- Normal requirements
  - Authentication
    - With some Level of Assurance (LoA)
  - Authorization
  - Accountability

AAA

Access control

- Subjects and objects
  - Both digital entities
  - Subjects can be something exhibiting activity :
    - Processes
    - Computers
    - Networks
  - Objects can be the target of an action :
    - Stored data
    - CPU time
    - Memory
    - Processes
    - Computers
    - Network
- An entity can be both subject and object

Least privilege principle

> Every program and every user of the system should operate using the least set of privileges necessary to complete the job
> J. H. Saltzer, M. D. Schroeder,
> The protection of information in computer systems, Proc. of the IEEE, 63(9) 1975

- Privilege:
  - Authorization to perform a given task
  - Similar to access control clearance
- Each subject should have, at any given time, the exact privileges required to the assigned tasks
  - Less privileges than the required create unsurpassable barriers
  - More privileges than the required create vulnerabilities
    - Damage resulting from accidents or errors
    - Potential interactions among privileged programs
    - Misuse of a privileges
    - Unwanted information flows
      - "need-to-know" military restrictions

Access control models

	O1	O2	…	Om-1	Om
S1		Access rights			
S2					
…					
Sn-1					
Sn					

- Access control matrix
  - Matrix with all access rights for subjects relatively to objects
  - Conceptual organization

Access control models

	O1	O2	...	Om-1	Om
S1		Access rights			
S2					
...					
Sn-1					
Sn					

- ACL-based mechanisms
  - ACL: Access Control List (matrix column)
    - List of access rights for specific subjects
    - Access rights can be positive or negative
    - Default subjects may often be used
  - Usually ACLs are stored along with objects
    - e.g. for file system objects.

Access control models

	O1	O2	...	Om-1	Om
S1		Access rights			
S2					
...					
Sn-1					
Sn					

- Capability-based mechanisms
  - Capability: unforgeable authorization token (matrix row)
    - Contains object references and access rights
  - Access granting
    - Transmission of capabilities between subjects
  - Usually capabilities are kept by subjects
    - E.g. OAuth 2.0 access tokens

Access control kinds: MAC and DAC

- Mandatory access control (MAC)
  - Access control policy statically implemented by the access control monitor
  - Access control rights cannot be tailored by subjects or object owners
- Discretionary access control (DAC)
  - Some subjects can update rights granted or denied to other subjects for a given object
    - Usually this is granted to object owners and system administrators

Access control kinds: Role-Based Access Control (RBAC)

D.F. Ferraiolo and D.R. Kuhn, "Role Based Access Control", 15th National Computer Security Conference, Baltimore, October 1992

- Not DAC or MAC
  - Roles are dynamically assigned to subjects
    - For access control it matters the role played by the subject and not the subject's identity
- Access control binds roles to (meaningful) operations
  - Operations are complex, meaningful system transactions
    - Not the ordinary, low-level read/write/execute actions on individual objects
  - Operations can involve many individual lower-level objects

Access control kinds: RBAC rules (1/2)

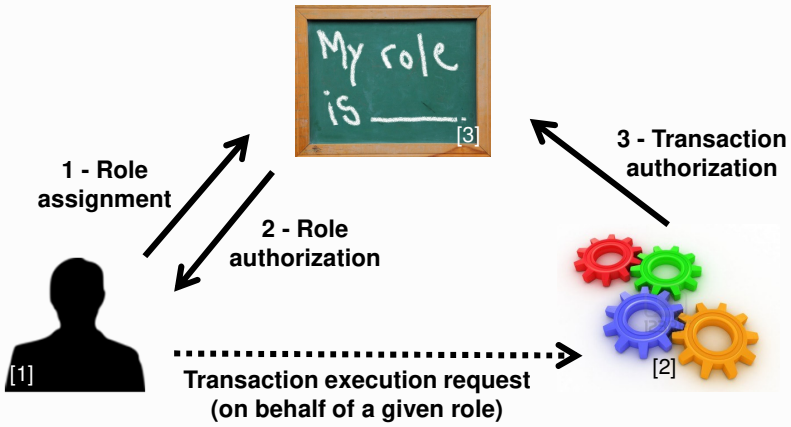
- Role assignment:
  - All subject activity on the system is conducted through transactions
    - And transactions are allowed to specific roles
    - Thus all active subjects are required to have some active role
  - A subject can execute a transaction **iff** it has selected or been assigned a role which can use the transaction

Access control kinds: RBAC rules (2/2)

- Role authorization:
  - A subject's active role must be authorized for the subject
- Transaction authorization:
  - A subject can execute a transaction **iff**
    - the transaction is authorized through the subject's role memberships

    and
    - there are no other constraints that may be applied across subjects, roles, and permissions

RBAC rules

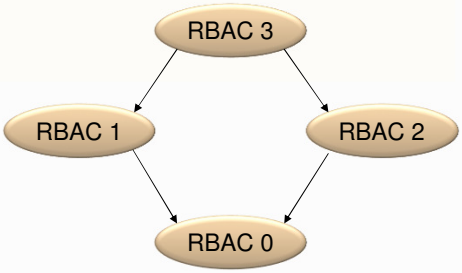


[1] From http://www.clker.com/clipart-24011.html
[2] From http://www.123rf.com/photo_12115593_three-dimensional-colored-toothed-wheels.html
[3] From http://www1.yorksolutions.net/Portals/115255/images/MyRoleIs.jpg

RBAC: Roles vs. groups

- Roles are a collection of permissions
  - The permissions are granted to the subjects that, at a given instant, play the role
  - A subject can only play a role at a given time
- Groups are a collection of users
  - And permissions can be granted both to users and groups
  - A subject can belong to many groups at a given time
- The session concept
  - Role assignment is similar to a session activation
  - Group membership is ordinarily a static attribute

RBAC variants



- RBAC 0
  - No role hierarchies
  - No role constraints
- RBAC 1
  - RBAC 0 w/ role hierarchies (privilege inheritance)
- RBAC 2
  - RBAC 0 w/ role constraints (separation of duties)
- RBAC 3
  - RBAC 1 + RBAC 2

NIST RBAC model

- Flat RBAC
  - Simple RBAC model w/ user-role review
- Hierarchical RBAC
  - Flat RBAC w/ role hierarchies (DAG or tree)
  - General and restricted hierarchies
- Constraint RBAC
  - RBAC w/ role constraints for separation of duty
- Symmetric RBAC
  - RBAC w/ permission-role review

Access control kinds: Context-Based Access Control (CBAC)

- Access rights have an historical context
  - The access rights cannot be determined without reasoning about past access operations
  - Example:
    - Stateful packet filter firewall
- Chinese Wall policy
  - Conflict groups
  - Access control policies need to address past accesses to objects in different members of conflict groups

D.F.C. Brewer and M.J. Nash, "The Chinese Wall Security Policy ", IEEE Symposium on Security and Privacy, 1989

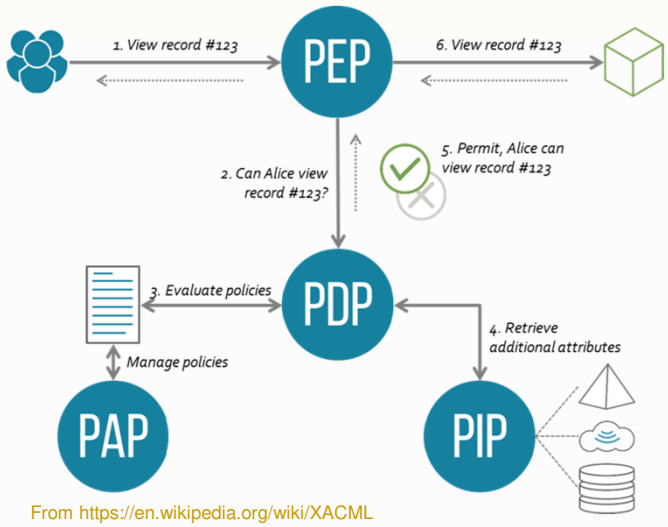
Access control kinds: Attribute-Based Access Control (ABAC)

- Access control decisions are made based on attributes associated with relevant entities
- OASIS XACML architecture
  - Policy Administration Point (PAP)
    - Where policies are managed
  - Policy Decision Point (PDP)
    - Where authorization decisions are evaluated and issued
  - Policy Enforcement Point (PEP)
    - Where access requests to a resource are intercepted and confronted with PDP's decisions
  - Policy Information Point (PIP)
    - Provides external information to a PDP

XACML: Access control with PEP and PDP

- A subject sends a request
  - Which is intercepted by the Policy Enforcement Point (PEP)
- The PEP sends an the authorization request to the Policy Decision Point (PDP)
- The PDP evaluates the authorization request against its policies and reaches a decision
  - Which is returned to the PEP
  - Policies are retrieved from a Policy Retrieval Point (PRP)
  - Useful attributes are fetched from Policy Information Points (PIP)
  - Policies are managed by the Policy Administration Point (PAP)

XACML big picture



From https://en.wikipedia.org/wiki/XACML

Break-the-glass access control model

- In some scenarios it may be required to overcome the established access limitations
  - e.g. in a life threatening situation
- In those cases the subject may be presented with a break-the-glass decision upon a deny
  - Can overcome the deny at their own responsibility
  - Logging is fundamental to prevent abuses

Separation of duties

R.A. Botha, J.H.P. Eloff, "Separation of duties for access control enforcement in workflow environments", IBM Systems Journal, 2001

- Fundamental security requirement for fraud and error prevention
  - Dissemination of tasks and associated privileges for a specific business process among multiple subjects
  - Often implemented with RBAC
- Damage control
  - Segregation of duties helps reducing the potential damage from the actions of one person
  - Some duties should not be combined into one position

Segregation of duties:

ISACA (Inf. Systems Audit and Control Ass.) Matrix guideline

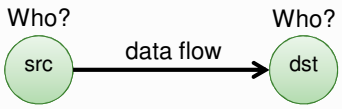
Exhibit 2.9—Segregation of Duties Control Matrix

	Control Group	Systems Analyst	Application Programmer	Help Desk and Support Manager	End User	Data Entry	Computer Operator	Database Administrator	Network Administrator	Systems Administrator	Security Administrator	Systems Programmer	Quality Assurance
Control Group		X	X	X		X	X	X	X	X		X	
Systems Analyst	X			X	X		X				X		X
Application Programmer	X			X	X	X	X	X	X	X	X	X	X
Help Desk and Support Manager	X	X	X		X	X		X	X	X		X	
End User		X	X	X			X	X	X			X	X
Data Entry	X		X	X			X	X	X	X	X	X	
Computer Operator	X	X	X		X	X		X	X	X	X	X	
Database Administrator	X		X	X	X	X	X		X	X		X	
Network Administrator	X		X	X	X	X	X	X					
System Administrator	X		X	X		X	X	X				X	
Security Administrator		X	X			X	X					X	
Systems Programmer	X		X	X	X	X	X	X		X	X		X
Quality Assurance		X	X		X							X	

X—Combination of these functions may create a potential control weakness.

Information flow models

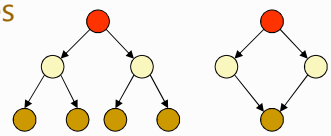
- Authorization is applied to data flows
  - Considering the data flow source and destination
  - Goal: avoid unwanted/dangerous information flows



- Src and Dst security-level attributes
  - Information flows should occur only between entities with given security-level attributes
  - Authorization is given based on the SL attributes

Multilevel security

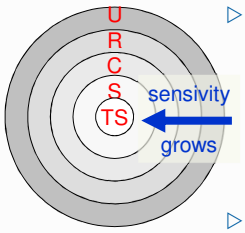
- Subjects (or roles) act on different security levels
  - Levels do not intersect themselves
  - Levels have some partial order
    - Hierarchy
    - Lattice



- Levels are used as attributes of subjects and objects
  - Subjects: security level clearance
  - Objects: security classification
- Information flows & security levels
  - Same security level → authorized
  - Different security levels → controlled
    - Authorized or denied on a "need to now" basis

Multilevel security levels: Military / Inteligence organizations

- Typical levels
  - Top secret
  - Secret
  - Confidential
  - Restricted
  - Unclassified



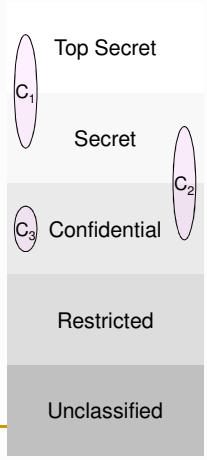
- Portugal (NTE01, NTE04)
  - Muito Secreto
  - Secreto
  - Confidencial
  - Reservado
- EU example
  - EU TOP SECRET
  - EU SECRET
  - EU CONFIDENTIAL
  - EU RESTRICTED
  - EU COUNCIL / COMMISSION
- NATO example:
  - COSMIC TOP SECRET (CTS)
  - NATO SECRET (NS)
  - NATO CONFIDENTIAL (NC)
  - NATO RESTRICTED (NR)

Multilevel security levels: Civil organizations

- Typical levels
  - Restricted
  - Proprietary
  - Sensitive
  - Public

Security categories (or compartments)

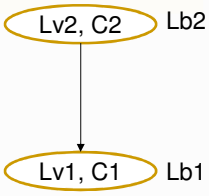
- Self-contained information environments
  - May span several security levels
- Military environments
  - Military branches, military units
- Civil environments
  - Departments, organizational units
- An object can have belong to different compartments and have a different security classification in each of them
  - (top-secret, crypto), (secret, weapon)



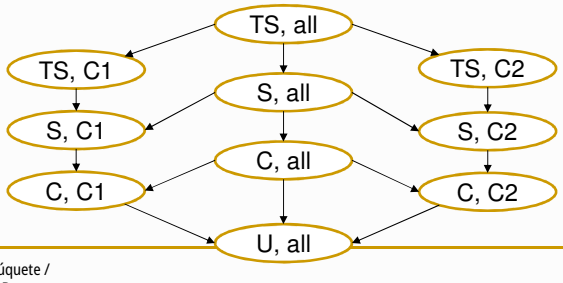
Security labels

- Label = Category + Level
- Relative order between labels

  $Lb1 \leq Lb2 \Rightarrow C1 \subseteq C2 \wedge Lv1 \leq Lv2$



- Labels form a lattice



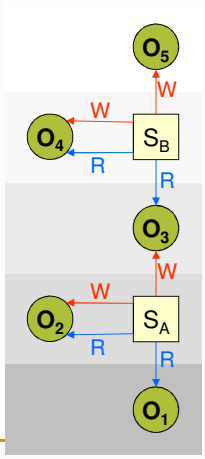
Bell-La Padula MLS Model

D. Elliott Bell, Leonard J. La Padula, "Secure Computer Systems: Mathematical Foundations", MITRE Technical Report 2547, Volume I, 1973

- Access control policy for controlling information flows
  - Addresses data confidentiality and access to classified information
  - Addresses disclosure of classified information
    - Object access control is not enough
    - One needs to restrict the flow of information from a source to authorized destinations
- Uses a state-transition model
  - In each state there are subjects, objects, an access matrix and the current access information
  - State transition rules
  - Security levels and clearances
    - Objects have a security labels
    - Subjects have security clearances
    - Both refer to security levels (e.g. CONFIDENTIAL)

Bell-La Padula MLS Model: Secure state-transition model

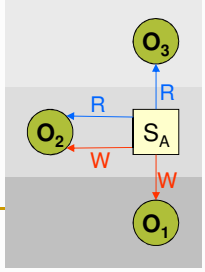
- Simple security condition (no read up)
  - S can read O iff L(S) ≥ L(O)
- *-property (no write down)
  - S can write O iff L(S) ≤ L(O)
  - aka confinement property
- Discretionary Security Property
  - DAC-based access control



Biba Integrity Model

K. J. Biba, "Integrity Considerations for Secure Computer Systems", MITRE Technical Report 3153, The Mitre Corporation, April 1977

- Access control policy for controlling information flows
  - For enforcing data integrity control
  - Uses integrity levels, not security levels
- Similar to Bell-La Padula, with inverse rules
  - Simple Integrity Property (no read down)
    - S can read O iff I(S) ≤ I(O)
  - Integrity *-Property (no write up)
    - S can write O iff I(S) ≥ I(O)



Clark-Wilson Integrity Model

D. D. Clark, D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies", IEEE Symposium on Security and Privacy, 1987

- Addresses information integrity control
  - Uses the notion of transactional data transformations
  - Separation of duty: transaction certifiers ≠ implementers
- Terminology
  - Data items
    - Constrained Data Item (**CDI**)
      - Can only be manipulated by TPs
    - Unconstrained Data Item (**UDI**)
  - Integrity policy procedures
    - Integrity Verification Procedure (**IVP**)
      - Ensures that all CDIs conform with the integrity specification
    - Transformation Procedure (**TP**)
      - Well-formed transaction
        - Take as input a CDI or a UDI and produce a CDI
      - Must guarantee (via certification) that transforms all possible UDI values to "safe" CDI values

Clark-Wilson Integrity Model: Certification & Enforcement

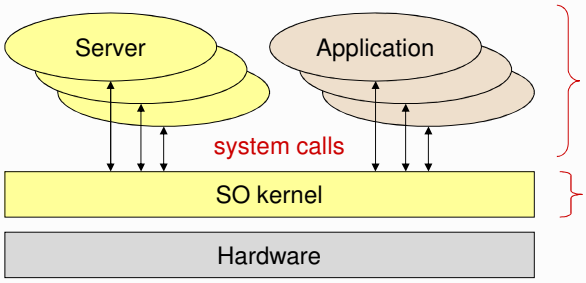
- Integrity assurance
  - Certification
    - Relatively to the integrity policy
  - Enforcement
- Two sets of rules
  - Certification Rules (C)
  - Enforcement Rules (E)

Clark-Wilson Integrity Model: Certification & Enforcement rules

- Basic rules:
  - **C1**: when an IVP is executed, it must ensure that all CDIs are valid
  - **C2**: for some associated set of CDIs, a TP must transform those CDIs from one valid state to another
  - **E1**: the system must maintain a list of certified relations and ensure only TPs certified to run on a CDI change that CDI
- Separation of duty (external consistency)
  - **E2**: the system must associate a user with each TP and set of CDIs. The TP may access CDIs on behalf of the user if authorized
  - **C3**: allowed user-TP-CDI relations must meet "separation of duty" requirements
- Identification gathering
  - **E3**: the system must authenticate every user attempting a TP (on each attempt)
- Audit trail
  - **C4:** all TPs must append to a log enough information to reconstruct operations
- UDI processing
  - **C5**: a TP taking a UDI as input may only perform valid transactions for all possible values of the UDI. The TP will either accept (convert to CDI) or reject the UDI
- Certification constraints
  - **E4**: only the certifier of a TP may change the associated list of entities

Security in Operating Systems

Operating system

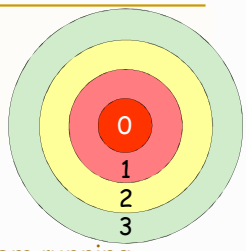


user-mode:
Execute in normal CPU mode,
no access to privileged instructions

supervisor mode:
Execute in privileged CPU mode,
has access to privileged instructions

- Kernel mission
  - Virtualize the hardware
    - Computational model
  - Enforce protection policies and provide protection mechanisms
    - Against involuntary mistakes
    - Against non-authorized activities

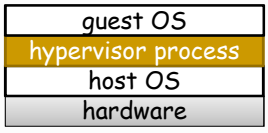
Execution rings



- Different levels of privilege
  - Forming a set of concentric rings
  - Used by CPU's to prevent non-privileged code from running privileged opcodes
    - e.g. IN/OUT, TLB manipulation
- Nowadays processors have 4 rings
  - But OS's usually use only 2
    - 0 (supervisor/kernel mode) and 3 (user-mode)
- Transfer of control between rings requires special gates
  - The ones that are used by syscalls

Virtual machines and hypervisors

- Emulation of a particular (virtual) hardware with the existing one (real)



- Hosted virtualization
  - The hypervisor is a process of a given OS (host)
  - The VM runs inside the virtualizer (guest OS)

- Bare-metal virtualization
  - The hypervisor runs on top of the host hardware

Execution of virtual machines

- Common approach for hosted virtualization
  - Software-based virtualization
  - Direct execution of guest user-mode code
  - Binary, on-the-fly translation of privileged code (full virtualization)
    - Guest OS kernels remain unchanged
    - No direct access to the host hardware
- Hardware-assisted virtualization (bare-metal)
  - Full virtualization
  - There is a ring -1 below ring 0
    - Hypervisor (or Virtual Machine Monitor, VMM)
  - It can virtualize hardware for many ring 0 kernels
    - No need of binary translation
    - Guest OS's run faster

Computational model

- Set of entities (objects) managed by the OS kernel
  - User identifiers
  - Processes
  - Virtual memory
  - Files and file systems
  - Communication channels
  - Physical devices
    - Storage
      - Magnetic disks, optical disks, silicon disks, tapes
    - Network interfaces
      - Wired, wireless
    - Human-computer interfaces
      - Keyboards, graphical screens, text consoles, mice
    - Serial/parallel I/O interfaces
      - USB, serial ports, parallel ports, infrared, bluetooth

Computational model: User identifiers

- For the OS kernel a user is a number
  - Established during a login operation
  - User ID (UID)
- All activities are executed on a computer on behalf of a UID
  - The UID allows the kernel to assert what is allowed/denied to processes
  - Linux: UID 0 is omnipotent (root)
    - Administration activities are usually executed with UID 0
  - Windows: concept of privileges
    - For administration, system configuration, etc.
    - There is no unique, well-known identifier for and administrator
    - Administration privileges can be bound to several UIDs
      - Usually through administration groups
      - Administrators, Power Users, Backup Operators

Computational model: Group identifiers

- Groups also have an identifier
  - A group is a set of users
  - A group can be defined by including other groups
  - Group ID (GID)
- A user can belong to several groups
  - Rights = UID rights + rights of his groups
- In Linux all activities are executed on behalf of a set of groups
  - Primary group
    - Typically used for setting file protection
  - Secondary groups

Computational model: Processes

- A process defines the context of an activity
  - For taking security-related decisions
  - For other purposes (e.g. scheduling)
- Security-related context
  - Identity (UID and GIDs)
    - Fundamental for enforcing access control
  - Resources being used
    - Open files
      - Including communication channels
    - Reserved virtual memory areas
    - CPU time used

Access control

- The OS kernel is an access control monitor
  - Controls all interactions with the hardware
  - Controls all interactions between entities of the computational model
- Subjects
  - Usually local processes
    - Through the system call API
    - A system call (or syscall) is not an ordinary function call
  - But also messages from other hosts

Mandatory access controls

- OS kernels have plenty mandatory access control policies
  - They are part of the computational model logic
  - They cannot be overruled not even by administrators
    - Unless they change the OS kernel behavior
- Examples:
  - Kernel runs in CPU privileged modes, user applications run in non-privileged modes
  - Separation of virtual memory areas
  - Inter-process signaling
  - Interpretation of files' ACLs

Protection with ACLs

- Each object has an ACL
  - It says which subjects can do what
- An ACL can be discretionary or mandatory
  - When mandatory it cannot be modified
  - When discretionary it can be tailored
- An ACL is checked when an activity, on behalf of a subject, wants to manipulate the object
  - Ifs the manipulation request is not authorized by the ACL, the access is denied
  - The SO kernel is the responsible for enforcing ACL-based protection
    - It acts as a security monitor

Protection with capabilities

- Less common in normal OS kernels
  - Though there are some good examples
- Example: open file descriptors
  - Applications' processes indirectly manipulate open file descriptors through the OS kernel
    - Using integer indexes (also called file descriptors …)
    - The OS kernel has full control over the contents of open file descriptors
  - Open file descriptors can only be granted to other processes through the OS kernel
    - Not really a usual operation, but possible!
  - Changes in the protection of files does not impact existing open file descriptors
    - The access rights are evaluated and memorized when the file is open

Unix file protection ACLs: Fixed-structure, discretionary ACL

- Each file system object has an ACL
  - Binding 3 rights to 3 subjects
  - Only the owner can update the ACL
- Rights: **R W X**
  - Read right / Listing right
  - Write right / create or remove files or subdirectories
  - Execution right / use as process' current working directory
- Subjects:
  - An UID (owner)
  - A GID
  - Others

Windows NTFS file protection: Variable-size, discretionary ACLs

- Each file system object has an ACL and a owner
  - **The ACL grants 14 types of access rights to a variable-size list of subjects**
  - **Owner can be an UID or a GID**
  - **Owner has no special rights over the ACL**
- Subjects:
  - **Users (UIDs)**
  - **Groups (GIDs)**
    - The group "Everyone" stands for anybody
- Rights:
  - **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**

Privilege elevation: Set-UID mechanism

- It is used to change the UID of a process running a program stored on a Set-UID file
  - If the program file is owned by UID X and the set-UID ACL bit is set, then it will be executed in a process with UID X, independently of the UID of the subject that executed the program
- It is used to provide privileged programs for running administration task invoked by normal, untrusted users
  - Change the user's password (passwd)
  - Change to super-user mode (su, sudo)
  - Mount devices (mount)

Privilege elevation: Set-UID mechanism (cont.)

- Effective UID / Real UID
  - Real UID is the UID of the process creator
    - App launcher
  - Effective UID is the UID of the process
    - The one that really matters for defining the rights of the process
- UID change
  - Ordinary application
    - eUID = rUID = UID of process that executed **exec**
    - eUID cannot be changed (unless = 0)
  - Set-UID application
    - eUID = UID of **exec**'d application file, rUID = initial process UID
    - eUID can revert to rUID
  - rUID cannot change

Privilege elevation: Set-UID/Set-GID decision flowchart

- exec (path, ...)
  - File referred by path has Set-UID?
  - Yes
    - ID = path owner
    - Change the process effective UID to ID
  - No
    - Do nothing
  - File referred by path has Set-GID?
  - Yes
    - ID = path GID
    - Change the process GIDs to ID only
  - No
    - Do nothing

Privilege elevation: sudo mechanism

- Administration by root is not advised
  - One "identity", many people
  - Who did what?
- Preferable approach
  - Administration role (uid = 0), many users assume it
    - Sudoers
    - Defined by a configuration file used by sudo
- sudo is a Set-UID application with UID = 0
  - Appropriate logging can take place on each command run with sudo

Privilege reduction: chroot mechanism (or jail)

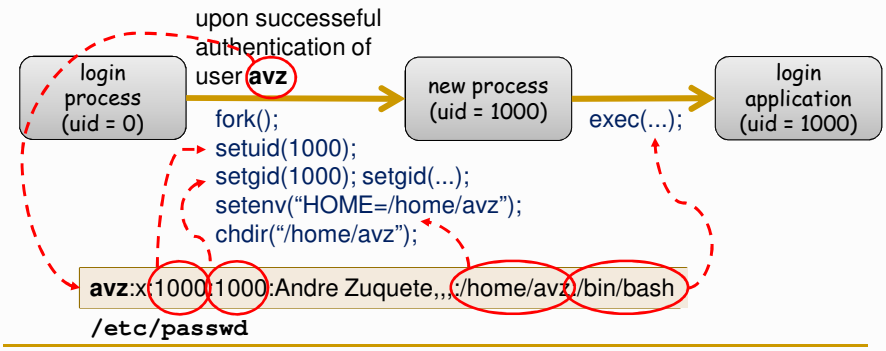
- Used to reduce the visibility of a file system
  - Each process descriptor has a root i-node number
    - From which absolute pathname resolution takes place
  - chroot changes it to an arbitrary directory
    - The process' file system view gets reduced
- Used to protect the file system from potentially problematic applications
  - e.g. public servers, downloaded applications
  - But it is not bullet proof!

Linux login: Not an OS kernel operation

- A privileged login application presents a login interface for getting users' credentials
  - A username/password pair
  - Biometric data
  - Smartcard and activation PIN
- The login application validates the credentials and fetches the appropriate UID and GIDs for the user
  - And starts an initial user application on a process with those identifiers
    - In a Linux console this application is a shell
  - When this process ends the login application reappears
- Thereafter all processes created by the user have its identifiers
  - Inherited through forks

Linux: from login to session processes

- The login process must be a privileged process
  - Has to create processes with arbitrary UID and GIDs
    - The ones of the entity logging in



Login in Linux: Password validation process

- Username is used to fetch a UID/GID pair from /etc/passwd
  - And a set of additional GIDs in the /etc/group file
- Supplied password is transformed using a digest function
  - Currently configurable, for creating a new user (/etc/login.defs)
  - Its identification is stored along with the transformed password
- The result is checked against a value stored in /etc/shadow
  - Indexed again by the username
  - If they match, the user was correctly authenticated
- File protections
  - /etc/passwd and /etc/group can be read by anyone
    - This is fundamental, for instance, for listing directories (why?)
  - /etc/shadow can only be read by root
    - Protection against dictionary attacks

Secure data storage

Problems (1/3)

- The classical file system protection is limited
  - Physical protection assumptions
    - Physical confinement of storage devices
  - Logical protection assumptions
    - Access control performed by systems managing the devices
      - e.g. operating systems
    - Proper use of ACLs or other authorization mechanisms

Problems (2/3)

- There are numerous scenarios where this protection is useless
  - Direct/physical access to storage devices
    - Mobile computational units
      - Laptops, PDAs, smartphones
    - Removable storage devices
      - Tapes, diskettes, CDs, DVDs, memory cards
  - Bypassing of logical access control mechanisms
    - Unethical access by powerful users (e.g. administrators)
    - Personification of users

Problems (3/3)

- Distributed access raises security issues
  - Trust in (unknown) administration teams
  - Secure communications
    - Confidentiality, integrity
  - Remote authentication of users
    - Security level provided
      - i.e. how hard it is to impersonate someone
    - Integration among clients and servers
      - Applications, operating system
    - Interaction model
      - Sessions vs. requests
    - Entities
      - People vs. machines/systems

Solution:
File encryption

- Encryption/decryption of files' contents
  - Can safely circulate along dangerous networks
  - Can safely be stored in insecure storage devices
    - Either mobile or administrated by others
- Problems
  - Data retrieval
    - End-users cannot loose encryption/decryption keys
    - Illegitimate end-user encryption
      - Corporate data
  - File sharing
    - It implies some sort of (group) key sharing
  - Interference with regular storage administration procedures
    - e.g. backups

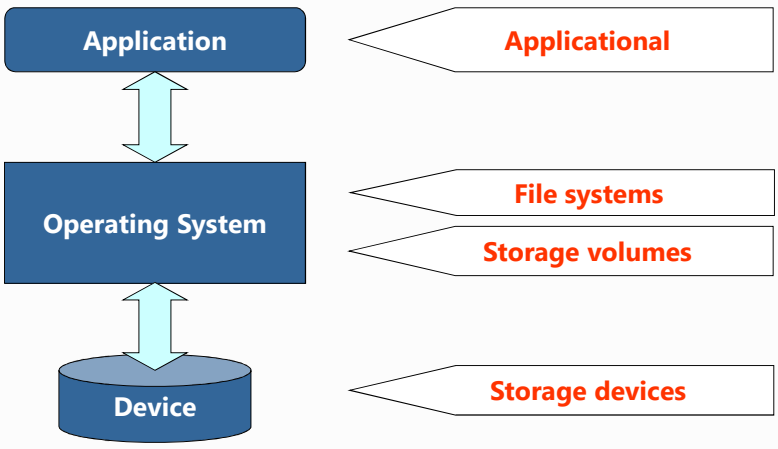
Ideal architecture (1/2)

- Cipher/decipher transparency
  - At the application level
  - At the level of OS file caches
    - But tacking into consideration authorization issues
- Visibility of securely stored data
  - Visual awareness
    - Of what is protected and not protected
  - Automatic setting of encryption attributes
    - With customization options

Ideal architecture (2/2)

- Easy sharing of encrypted data
  - By groups of users
- Decryption capacity under special circumstances by authorized people
  - Legal enforcement
  - Protection against the loss of decipher keys

Current approaches



Applicational

- Data transformed by autonomous applications
  - Little or no integration with other applications
  - Usually it is clear what is secure or not
    - e.g. using specific file extensions
- Data can be transformed with different algorithms
  - Adds flexibility, increases security
  - Complicates recovery procedures
- There are vulnerability windows
  - Cleartext resulting files used by other applications
- Hard to share data without sharing keys
  - Secret keys or public keys
- Examples:
  - PGP, AxCrypt, etc.

Storage volumes / devices

- Cipher/decipher operations at the volume / device level
  - Total transparency for applications and possibly to the OS
  - The visibility of protected data has volume / device granularity
  - Not required to handle file systems issues
    - Protection of meta-information and file data
    - Users and access rights
- Cannot differentiate accesses by different users
  - More suitable for personal storage devices
- Cannot solve issues raised by distributed file systems
  - Decipher occurs when data is fetched from devices to server caches
- Examples:
  - PGPdisk, LUKS (Linux Unified Key Setup)
  - Self-Encrypting Drives

Secure file systems: Approaches

- Data is transformed in the path between storage devices and the memory of applications
  - Storage device ⇔ file cache
    - No protection for remote accesses (server deciphers)
    - The access to caches gets more complex
      - Coordination with ACLs
      - Knowledge of cipher/decipher keys by the OS
  - File cache ⇔ memory of applications
    - Protection for remote accesses (clients decipher)
    - Can take place outside the OS (e.g. STDIO in UNIX)
- Examples:
  - CFS (Cryptographic File System), encfs
  - EFS (Encrypted File System)

Secure file systems: Limitations (1/2)

- File system integrity must be preserved
  - Some file attributes cannot be hidden
    - For keeping the regular file system operation
    - Because of other administration tools (e.g. backup tools)
- Attributes that can easily be hidden
  - Arbitrary file/directory names
    - Encrypted versions must conform FS naming rules
  - File contents
    - Preferably without changing file's size

Secure file systems: Limitations (2/2)

- Attributes that **cannot** (should not) be hidden/changed
  - Object types
    - They define the structure of the file system
  - Contents of directories
  - Some well-defined names
    - e.g. **"."** and **".."** in UNIX
  - Dates
    - For managing backups
  - Dimension
    - For knowing the real occupation of storage devices
  - Ownership
    - For managing storage quotas
  - Access protection
    - For keeping the normal access control policies

Secure file systems: Practical encryption issues

- Uniform random access to encrypted data
  - Ciphers with feedback are not suitable
- Confidentiality
  - Not advised to use the same key for different files
    - Similar patterns could reveal similar files
  - Not advised to use the same key for an entire file
    - Similar patterns along a file could reveal its semantics
  - Stream ciphers are not advised w/ the same key for different files
    - Known-plaintext attacks could reveal contents of other files

CFS (Cryptographic File System)

- NFS extension
  - OS ⇔ local CFS server ⇔ local or remote NFS server
  - The NFS interface is kept
  - The MOUNT interface changes
    - Includes a password
- Encryption / decryption operations
  - Performed by the local CFS server
    - Files circulate encrypted in the network
    - Decrypted file contents are maintained in the client OS file cache
      - All local users with READ access to the file can read the decrypted contents
  - Cipher/decipher keys supplied per each mount point
    - Communicated to the local CFS server by a modified mount command
    - This command uses the new MOUNT interface

CFS

- Encrypts file names and file contents
  - Using two keys (K1 and K2) derived from a password
- Name
  - Concatenated with and integrity control value
  - Encrypted with ECB
- File contents
  - Stream with OFB ⊕ block ECB
    - OFB with K1
    - ECB com K2 (disk blocks are not increased)
  - OFB mask computed with K1 per mount point
  - Random IV per file
    - Applied between XOR with OFB mask and ECB
    - Stored in the i-node GID
    - CFS provides the directory GID instead of the file GID

EFS (Encrypted File System)

- Windows NTFS extension
  - First appeared in Windows 2000
  - Provides encryption facilities to NTFS 5
- Functionality
  - Each user is bound to an asymmetric key pair
    - Stored and managed by the OS
  - Each file is encrypted with a unique symmetric key
    - FEK (File Encryption Key)
  - An encrypted file can be accessed by many users
    - For each file EFS stores copy of FEK encrypted with the public key of each authorized user
    - Encrypted FEKs are stored in a STREAM associated to the file
      - NTFS files are formed by sets of STREAMS
  - Each encrypted file is clearly visible
    - Using the Explorer file navigator

EFS cryptographic technology

- Algorithms
  - Asymmetric encryption of FEKs: RSA
  - Symmetric encryption with FEKs: DESX

    DESX ≡ DES with whitening

    FEK = (K1, K2, K3)

    C = K1 ⊕ DES(K2, P ⊕ K3)
- Problems
  - Asymmetric key pairs are stored in disk
    - Loss risk
    - Illegitimate access by administrators
  - Files are decrypted by servers
    - No network protection for files stored remotely

Database security

Advantages of using databases

- Shared access
  - Many users, one common, centralized data set
- Minimal redundancy
  - Individual users do not have to collect and maintain their own sets of data
- Data consistency
  - A change to a data value affects all users of that data value
- Data integrity
  - Data values can be protected against accidental or malicious undesirable changes
- Controlled access
  - Only authorized users are allowed to view or to modify data values

Security requirements (1/2)

- Physical integrity
  - Immunity to physical problems
    - e.g. power failures
  - Ability to reconstruct the database if destroyed in a catastrophe
- Logical integrity
  - Data structure (schema) is preserved
- Element integrity
  - Data in each element is accurate
- Auditability
  - It is possible to track who or what has accessed (or modified) which elements in the database

Security requirements (2/2)

- Access control
  - A user/role is allowed to access only authorized data/queries
  - Different users/roles can be restricted to different modes
    - e.g. read or write records
- User authentication
  - Every user/role is positively identified
  - Fundamental for audit trails and for permissions to access data
- Availability
  - Users/roles can access the database in general and all the data for which they are authorized

Two-phase updates

- Problem
  - Failures during updates may render databases incoherent
    - Logical integrity problem
  - But DBMS require ACID properties
    - Atomicity
      - Entire transaction happens or not
    - Consistency
      - The DB state must be consistent after transactions
    - Isolation
      - Concurrent transactions do not interfere with each other
    - Durability
      - Changes occur even in the presence of failures
- Solution: two-phase updates

Two-phase update

- 1st phase: intent phase
  - The DBMS gathers resources it needs to perform the update
  - It does everything to prepare for the update, but makes no changes to the database
  - Committing: writes a commit flag to the database
    - Point-of-no-return
    - After, the DBMS begins making permanent changes
- 2nd phase: commit phase
  - Makes the permanent changes in the database
    - Idempotent changes
  - It lasts until finishing all changes prepared in the first phase
  - When it finishes, the database changed to a new, stable and coherent state

Redundancy / internal consistency

- Error detection and correction codes
  - Parity bits, Hamming codes, cyclic redundancy checks
  - Can be applied to different data elements
    - Fields, records, entire database
  - More space
    - To store error detection/correction information
- Shadow fields
  - Duplication of fields or records
  - Requires substantial storage space

Concurrency / consistency

- Accesses by two users of the same DBMS must be constrained so that neither interferes with each other
  - Simple locking: multiple readers, one writer
  - But simple locking may not be enough on query-update cycles
- Solution
  - Treat every query-update cycle as a single atomic operation (a transaction)
    - e.g. flight booking
  - Synchronization should be applied to transactions
    - Two concurrent transactions cannot write (and sometimes read) the same field/record

Monitors

- DBMS unit responsible for the DB structural integrity
  - Checks entered values to ensure their consistency with field, record or database consistency constraints
- Types of monitors
  - Range comparisons
    - Tests if values belong to an acceptable range
  - State constraints
    - Describe the condition of the entire database
      - e.g. the commit flag
    - Impose integrity restriction rules
      - e.g. to detect duplicate records
  - Transition constraints
    - Describe required conditions before changing the database

Database activity monitoring

- DBMS usage supervision
  - To detect abuses
  - To detect unusual/suspect activity or operations
- DBMS independent
  - Not part of the DBMS
  - External observation of DBMS activity
- Monitoring sensors
  - Network activity
  - Local SQL commands performed
  - Log analysis

Sensitive data

- Data that cannot be publicly disclosed
  - With restricted visibility & use
- Risks
  - Privacy and welfare of individuals
  - Business activities
  - Security-related activities

Sensitive data

- Some databases contain sensitive data
  - Data that should not be made public
    - e.g. clinical records of patients
- Sensitivity depends on: BD purpose + DB data
  - Some record fields, entire records/tables, entire database
    - e.g. personal health record (HER) with all detected pathologies, treatments and interventions
    - e.g. clinical records of an AIDS table
    - e.g. defense-related databases
- Complexity
  - Simple cases: all or nothing
    - Everything sensitive, nothing sensitive
  - Complicated cases: part of the DB elements are sensitive
    - In some cases, sensitivity is extended to the simple existence of a field data or record

Sensitive data:
Factors that make data sensitive

- Inherently sensitive
  - The value itself may be so revealing that it is sensitive
- From a sensitive source
  - The value may reveal the identity of its source
- Declared sensitive
  - The value was explicitly declared sensitive
- Belongs to a sensitive record
  - Value of a record explicitly marked as sensitive
- Sensitive given previously disclosed information
  - By itself, the data is not sensitive, but together with other data, the whole can be sensitive

Sensitive data:
General Data Protection Regulation (GDPR)

- Personal data
  - Data that can be unequivocally linked to a (living) individual
  - Links can be provided by unique identifiers or sets of quasi-identifiers
- Specially sensitive personal data
  - Those that can threat fundamental rights
    - Ethnic/racial origins
    - Political opinions
    - Philosophical or religious beliefs
    - Syndicate memberships
    - Sexual life and orientations
    - Health status and history
    - Related with genetics or biometrics

Laws for the protection of personal data

- Each country has its how set of laws
  - There is not a global consensus
- In Portugal this is supervised by CNPD
  - Comissão Nacional de Proteção de Dados
  - All data processing involving personal data gathered from individuals needs to be submitted to CNPD for authorization
- General Data Protection Regulation (GDPR)
  - Started on May 25, 2018

Types of disclosures (of sensitive data)

- Exact data
  - The exact value of a sensitive datum
  - The most serious disclosure
- Bounds
  - Sensitive data item is > lower bound or < upper bound
  - Sometimes bounds are used to protect (hide) sensitive data
    - By providing bounds to elements instead of their exact value
- Negative result
  - By getting a negative result for a query on a sensitive value, a user can conclude that the value has a particular set of values
    - e.g. from a list of effective voters we can conclude who didn't vote

Types of disclosures (of sensitive data)

- Existence
  - The existence of a sensitive field in a record can be, by itself, sensitive information
    - Because it may reveal a hidden data gathering & processing activity
- Probable value
  - By crossing the results of several queries we can infer a probability for an element value

Inference

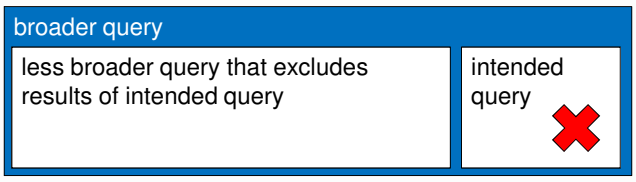
- Definition
  - A way to extract, or derive, sensitive information from non-sensitive information
  - We are assuming that there is no free access to the entire data repository
    - Conclusions need to be taken from authorized queries that, by themselves, alone, do not:
      - Leak any sensitive information
      - Allow an exclusive use of sensitive fields to select information

Inference attacks

- Direct attack
  - Uses queries with a blend of selection rules that use sensitive fields and non-sensitive fields
  - The DBMS can be deceived by the selection rules with non-sensitive fields, which are not intended to select particular records
- Indirect attack
  - Inference of particular values from statistical values computed over several records
    - Counts, sums, averages

Inference attacks

- Tracker attack
  - The database may conceal data when a small number of records make up the large proportion of the data revealed
  - A tracker attack can fool the DBMS by using different queries that reveal data and, by combining the results, the attacker can get the desired information



K-anonymity

L. Sweeney, "K-anonymity: A Model for Protecting Privacy", Int. Journal on Uncertainty, Fuzziness and Knowledge-based Systems. 2002

- Definition
  - No query can deliver an anonymity set with less than k entries
  - The anonymity set is the set of all possible subjects
- Privacy-critical attributes
  - (Unique) identifiers
  - Quasi-identifiers
    - When combined can produce unique tuples
  - Sensitive attributes
    - Potentially unique per subject
    - Disease, salary, crime committed, etc.

Multilevel security: Goal

- Tag information items with security classifications
  - e.g. unclassified, confidential, secret, top secret
- Tag queries with security levels
  - The security level of the entity responsible for the query
- Prevent queries from observing values of fields with a different security classification
  - Or from observing meaningful values

Multilevel security: Confidentiality with poli-instanciation

- A record with a particular key field may be duplicated in different security levels
  - Possibly with different values
- This reduces the precision of the database information
  - The correctness of the information depends on the entity performing the query
  - Duplicates can legitimately occur

Multilevel security: Separation strategies (1)

- Partitioning
  - Different security levels, different databases
  - Queries are directed to the appropriate DB
- Advantages
  - Easy to implement
- Disadvantages
  - Redundancy of information
  - Problems in the access to records with fields with different security levels

Multilevel security: Separation strategies (2)

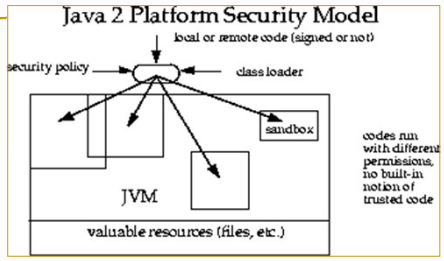
- Encryption
  - Fields are encrypted with a security-level key
- Advantages
  - Single database, same database structure
- Disadvantages
  - Decryption on each query with the adequate security level key
  - Randomized encryption: equal fields should not produce the same cryptogram
    - Otherwise statistics and known-plaintext attacks disclose values
    - Solution: different keys per record or different IVs per record
  - No encrypted values should be updated by providing another encrypted value

Multilevel security: Separation strategies (3)

- Integrity lock
  - Each data item is formed by three parts:
    - Data item, sensitivity label, checksum
  - The sensitivity label should be
    - Unforgeable (cannot be changed)
    - Unique (cannot be copied to another data item)
    - Concealed (cannot be observed)
- Advantages
  - Can use a regular DBMS
  - Trusted stored procedures are enough to implement them
- Disadvantages
  - Space for storing sensitivity labels and checksums

Java Virtual Machine Security

Java 2 Security Model

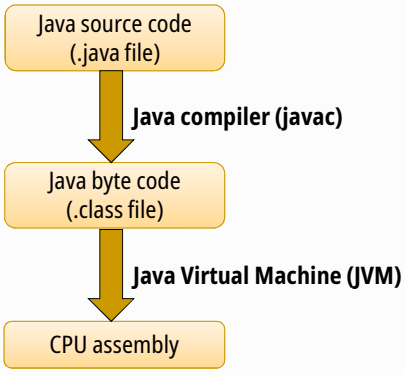


- Java Virtual Machine (JVM)
  - Java programs are implemented by a set of Java classes
    - From different sources
    - Not necessarily trusted
  - Secure sandbox for executing Java programs
- Security capabilities
  - Easily configurable security policy
  - Easily extensible access control structure
  - Extension of security checks to all Java programs

JVM sandbox model

- Creates a barrier around a Java execution environment
  - Applications are executed within a sandbox bounds
  - Cannot affect resources outside the sandbox
    - i.e. can only access resources available to the sandbox
- Basic rules of sandbox
  - Remote resource protection
    - Enforced by remote system
  - Local resource protection
    - Enforced by local security manager
  - JVM code and data protection
    - Enforced by static and dynamic checking

Java byte code

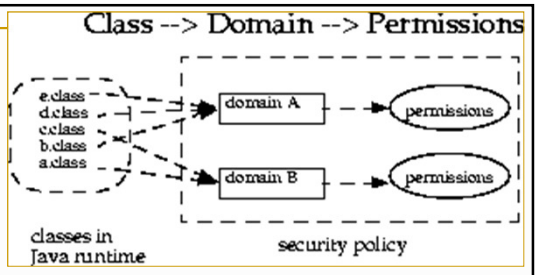


- One cannot assume that byte codes are produced by a correct compiler!
  - Byte codes can be produced by malware
- The JVM must tackle wrong or malicious byte codes

Java Run-time Environment (JRE): Security-related features

- Loads required classes
  - Usually upon a class method invocation
- Verifies the correctness of loaded classes
  - Checks consistency and integrity
- Compiles bytecodes
  - Only for invoked methods
    - Just-in-time
  - Keeps original bytecodes
    - For enforcing run-time validations
- Correct memory management
  - Memory allocation when needed
  - Automatic garbage collection
- Checks the correct execution of classes' code
  - Run-time integrity validations
    - Null pointer (ref)
    - Type checking
    - Dynamic (down)casting
    - Array bounds, etc.
  - Run-time security validations
    - Access control
      - Public, Package, Protected and Private access levels
      - Other permissions for Protection Domains
- Confinement
  - Isolation of Protection Domains

Protection domain



- A set of classes whose instances are granted the same set of permissions
  - Determined by the policy currently in effect
- JRE maintains a mapping from code (classes and instances) to their protection domains
- Instantiation of Protection Domains
  - ProtectionDomain (CodeSource, PermissionCollection);
  - ProtectionDomain (CodeSource, PermissionCollection, ClassLoader, Principal[]);
  - CodeSource (URL, Certificate[]);

Permissions

- Definitions of what is allowed or denied
  - Subclasses of interface java.security.Permission
- Examples
  - BasicPermission
    - Hierarchical name and arbitrary (or boolean) action
    - RuntimePermission, AWTPermission, ManagementPermission, NetPermission, PropertyPermission, etc.
  - FilePermission
    - Pathname & action (read, write, execute, delete)
  - SocketPermission
    - Host + port + action (accept, connect, listen, resolve)

Security policies

- Each JRE maintains an installed security policy
  - It determines the set of granted/denied authorizations
  - Subclass of java.security.policy
- Installed policy
  - There is always a policy installed (Policy Policy.getPolicy())
    - JRE includes a default policy reference implementation
      - Policy specified within one or more configuration files
      - [java_home]/lib/security/default.policy
    - Can be referenced by caller with getPolicy permission
  - Can be overwritten (void Policy.setPolicy(Policy))
    - Requires a setPolicy permission
    - The source location for the policy information utilized by the Policy object is up to the Policy implementation

Security manager

- At most one per JVM
  - Enforces a security policy for an application
    - What is allowed and denied
  - It helps to check whether an action is allowed before requesting it
    - In the context of the calling thread
- Class SecurityManager of java.lang
  - Default run-time security manager
  - Can be redefined
    - but requires runtime permission setSecurityManager
    - This prevents malicious classes to overrun an installed security manager
  - Many checkXXX methods
    - For checking authorization for specific actions
      void checkRead(String file)
    - Uses the AccessController class and the method checkPermissions

AccessController

- An abstract class used for:
  - Decide whether an access to a critical system resource is to be allowed or denied
    - According to the security policy currently in effect

```
FilePermission p = new FilePermission( "/temp/testFile", "read" );
AccessController.checkPermission( p );
```

  - Mark code as being *privileged*
    - Affecting subsequent access determinations
  - Obtaining a *snapshot* of the current calling context so access-control decisions from a different context can be made with respect to the saved context

Dynamic class loading: Class loaders

- Primordial class loader
  - Critical part of VM
    - Trusted VM component, defined in JVM specification
  - Prevents name spoofing of java.* library classes
- Additional class loaders
  - Defined by users/applications
    - They can help application to locate and download classes' contents
    - But the bytecodes of classes are installed by the VM class loader
  - Each one defines separate namespace environment
    - Each class is tagged with class loader that loaded it
    - Classes in one namespace cannot interact with classes in other namespaces
  - Allows different versions of same class name to co-exist
    - Typically associated with code from different origins

Dynamic class loading: Overview (1/2)

- Class loading security policies
  - No class loading of packages java.* other than from the canonical local repository
    - To avoid the replacement of the basic Java classes
    - Primordial class loader ensures this
  - Classes from different network servers do not interact
    - Different domains
    - No interference between "programs" of different sources

Dynamic class loading: Overview (2/2)

- Class loading steps
  - Locate the requested binary class
    - .class file
  - Parse/translate into internal data structures for emulation
  - Enforce the naming conventions
    - Domain, package, classes, fields/methods
    - Accessibility levels: public, private, package
  - Check correctness of binary class
    - File integrity check
    - Class integrity check
    - Bytecode integrity check
    - Runtime integrity check
  - Perform any translation of code and metadata
    - Make the method ready to be run
  - Initialize memory and pass control to emulation engine

Dynamic class loading: Class loader checks

- File integrity check
  - Magic number, proper formats used
  - Component declared and actual sizes
- Class integrity check
  - Has superclass and is not final
  - No override of final superclass method
  - Methods and fields have legal names and signatures
- Bytecode integrity check
  - Data-flow analysis
  - Stack checking
  - Static type checking for method arguments and bytecode operands
- Runtime integrity checks
  - Verifications on method calls

Subjects, Principals and Credentials

- Subjects (javax.security.auth.Subject) aggregate info related with a single (authenticated) entity
  - Identities
  - Credentials (public and private)
- Principals (java.security.Principal) encapsulate identities
  - Bind names to Subjects
- Credentials can be any kind of object