Information systems security

Lorenzo Di Maio

October 2024

Contents

Chapter 1

Authentication techniques protocols, and architectures

Authentication refers to the process of verifying the identity of an entity (whether it's a human, software component, or hardware element) before granting access to resources in a system. Authentication can be applied to various type of "actors", such as:

- **Human being**
- **Software component**
- **Hardware element**

Authentication vs Authorization

- **Authentication (authC/authN)**: established the identity of an entity.
- **Authorization (authZ)**: determines where a authenticated entity has permission to access.

1.1 Authentication factors

Authentication can be based on 3 primary factors:

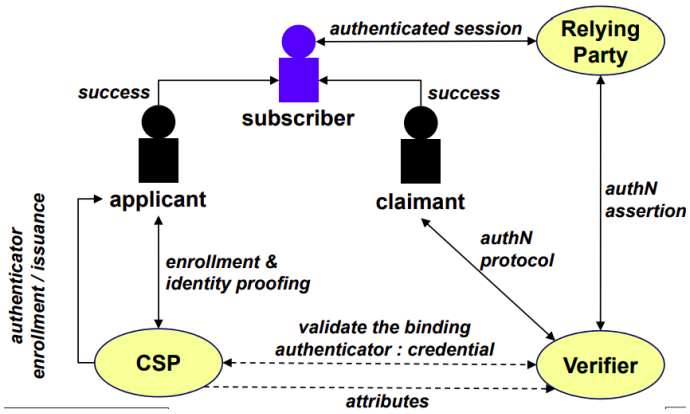
- **Knowledge**: Information that only the user knows and can provides as proof of their identity.
- **Ownership**: Physical object or device that only the user has access to.
- **Inherence**: This factor relies on unique biological traits of the user (e.g fingerprint).

N.B Authentication can be applied not just to human user, but also to processes and devices.

1.1.1 Risks

- **Knowledge:**
  - Storage → if passwords are stored improperly, they are vulnerable to thefts.
  - Demonstration → user might inadvertently reveal their password through social engineering.
  - Transmission → if passwords are sent over insecure channel, they can be intercepted by attackers.
- **Ownership:**
  - Authentication thieft
  - Cloning
  - Unathorized usage
- **Inherence:**
  - Counterfeiting → biometric data can be spoofed or replicated by attackers using sophisticated techniques.
  - Privacy → the use of biometric data raises the risk of biometric information being exposed.
  - Irreversibility → biometric traits cannot be replaced if compromised.

1.2 Digital Authentication model (NIST SP800.63B)



Entities:

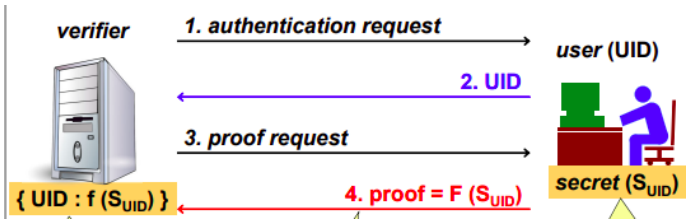
- **Subscriber**: applicant who has successfully completed identity proofing.
- **Applicant**: an individual applying to established a digital identity.
- **Claimant**: the user trying to prove their identity to access a system or service.
- **Relying Party**: entity (e.g service provider, website) that requests/receives an authN assertion from the verifier to asses user identity (and attributes).
- **Verifier**: validates the user's credential during each authentication event.
- **CSP**: an entity that issues, manages, and maintains **credentials** used by individuals to authenticate themselves.

Applicant vs Claimer

An applicant needs to enroll in the system for the first time to establish their identity. Instead, a claimant asserts their identity to gain access to the system after enrollment.

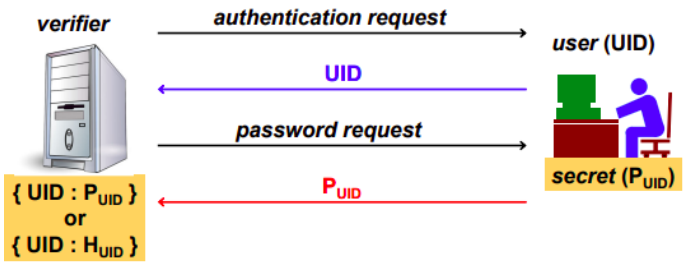
1.3 Generic authentication protocol

1. The user initiates an authentication request by sending their UID.
2. The user generates a proof based on their secret, using a secure function $F(S_{UID})$, and send this proof to the verifier.
3. The verifier checks if the received proof matches the stored representation of the secret.
4. If it matches, the user is successfully authenticated.



1.4 Password base authentication

1. The user sends their UID and P_{UID} (= Password) to the verifier.
2. The server verifies the proof:
   - If password are stored in cleartext, it directly compares the proof with the stored password.
   - If password are stored in hashes, it hashes the proof and compares it to the store hash H_{UID}.



Problems of reusable Passowrds

- **PWD Sniffing** (attackers intercept password during transmission)
- **PWD Database attack** (if DB contains plaintext or obfuscated PWD)
- **PWD Guessing** (very dangerous if it can be done offline, e.g against a list of PWD hashes)
- **PWD Enumeration** (PWD brute force attack)
  - If PWD is limited in length and/or character type.
  - If authN protocol does not block repeated failures.
- **PWD Duplications** (using the same PWD for one service against another one, due to user PWD reuse)
- **Cryptographic Aging** (as computing power grows, older cryptographic methods become vulnerable to new attacks)
- **PWD capture via server spoofing and phishing** (attackers deceive user into giving away their PWD by pretending to be legitimate service)

Password best practies

Suggestion to reduce password risks:

- Use alphabetical characters (upper case + lower case), digits and special characters
- Make passwords long (at least 8 character)
- Never use dictionary words
- Change password regularly, but not too frequently
- Do not reuse passwords across different services

Password storage

- **Server Side:**
  - Passwords should never be stored in cleartext.
  - Encrypted passwords aren't ideal since the server would need to know the encryption key.
  - Better to store a password digest (hashed password), though vulnerable to dictionary attacks.
  - Rainbow tables can speed up these attacks, so it's important to add a "salt" (random variation) to each password.
- **Client-side:**
  - Ideally, passwords are memorized by the user, but having many passwords makes this difficult.
  - People may resort to writing them down or using simple passwords, which is risky.
  - Using a password manager or encrypted file is a safer alternative.

1.5 The "dictionary" attack

- **Hypothesis:** The attacker knows the hash algorithm and the hashed password values.
- **Pre-computation:** For each word in a dictionary, compute and store its hash $store(DB, Word, hash(World))$
- **Attack process:**
  - Let HP (=hash password) to be the hash of an unknown password.
  - Lookup HP in the precomputed dictionary (DB) to find a matching password.
  - If found, output the password; if not, indicate it's "not in dictionary".

1.6 Rainbow Table attack

Rainbow Table is a **space-time trade-off technique** that reduces storage needs for exhaustive hash tables, making certain brute-force attacks feasible within limited space. It uses a reduction function $r : h \to p$ (which is NOT h^{-1}) to generate chains of hashes.
Example:

- For a 12-digit password, an exhaustive hash table would require $10^{12} rows(P_i : HP_i)$
- rainbow = 10^9 rows, each representing 1000 possible passwords.

Attack

```
for (k=HP, n=0; n<1000; n++)
 ■ p = r(k)
 ■ if lookup( DB, x, p ) then exit ( "chain found, rooted at x" )
 ■ k = h(p)
exit ( "HP is not in any chain of mine" )
```

1.7 Salting Passwords: A Defense Against Dictionary and Rainbow Table Attacks

Salting passwords is a security technique used to protect stored passwords from dictionary attacks and rainbow table attacks. A salt is a unique, random string added to each password before hashing. This ensures that even if two users have the same password, their hashes will be different due to the unique salt.

Steps for each user (UID):

- Generate or ask for the user's password.
- Create a unique, random slat for each user.
- Compute the salted hash: $SHP = hash(password \mid\mid salt)$
- Store the triplet $\{UID,\ SHP,\ salt\}$

Password Verification with Salt

- **Claimant:** Provides their user ID (UID) and password (PWD).
- **Verifier:**
  - Uses the UID to find the stored salted hash (SHP) and salt.
  - Computes $SHP' = hash(PWD \mid\mid salt)$.

The Linkedin attack

In 2012, LinkedIn was breached, exposing 6.5 million unsalted SHA-1 password hashes. The lack of salting allowed attackers to crack at least 236,578 passwords through crowdsources efforts before restrictions halted the exposure.

1.8 Strong authentication definitions

The concept of strong authentication (authN) is crucial in ensuring secure identity verification, but it has never been formally defined with a universal definition. Various definitions exist depending on the context, such as the European Central Bank (ECB) and PCI-DSS.

ECB definition

The ECB defines strong authentication as a process that involves at least two independent elements from **knowledge** (e.g. password), **ownership** (e.g. smartcard), and **inherence** (e.g. biometrics). The key requirement is that these elements must be mutually independent, so compromising one should not affect the others. Furthermore, at least one element should be **non-reusable** or **non-replicable** (except for inherence), with the entire process safeguarding the confidentiality of the authentication data.

PCI-DSS Definition

PCI-DSS mandates **multi-factor authentication (MFA)** for access to cardholder data, particularly for administrators and remote access from untrusted networks. Since version 3.2, MFA has become compulsory for remote access, and the use of the same factor twice (e.g., two passwords) does not qualify as MFA.

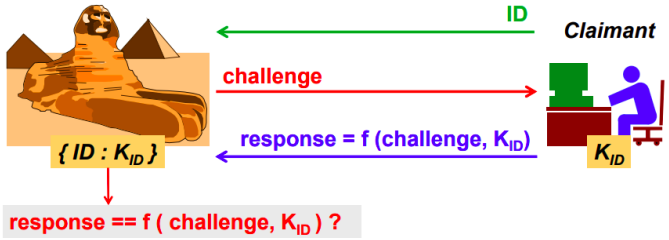
1.9 Challenge-Response Authentication (CRA)

Challenge-response authentication (CRA) is a widely used technique where a challenge is issued, and the claimant responds by solving it with a secret (shared or private). The challenge must be **non-repeatable** (usually a random nonce) to avoid replay attacks. The function used to compute the response must be **non-invertible**, otherwise, a listener can record the traffic and easily find the shared secret:

$$if(\exists f^{-1})\ then\ K_c\ =\ f^{-1}(response, challenge)$$

1.9.1 Symmetric CRA

In symmetric CRA, both the client and the server share a secret key that is used to verify the authenticity of a user or system This method is fast, often utilizing hash functions (e.g., SHA1, SHA2, SHA3).



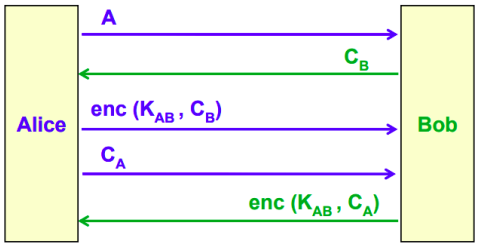
1.9.2 Mutual symmetric CRA

Mutual symmetric CRA requires both parties to authenticate each other. However, it's a old protocols so it has many vulnerabilities.

Version 1: Basic Exchange

In this case, the initiator explicitly provides its claimed identity (This version is considered outdated and insecure).
Process:

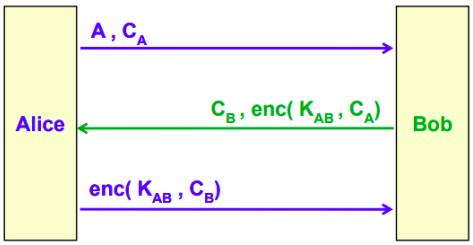
- Alice sends an encrypted challenge (C_B) to Bob using the shared key K_{AB}.
- Bob responds with an encrypted challenge (C_A) for Alice, also using K_{AB}.



Version 2: Improved Performance

Optimized by reducing the number of messages, which improves performance without compromising security.
Process:

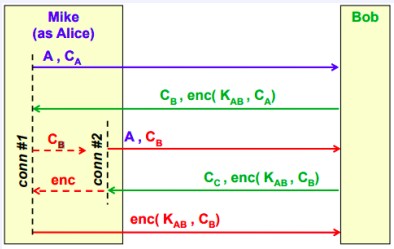
- Alice includes her identity (C_A) and sends an encrypted challenge (C_B) in the same message.
- Bob responds with his encrypted challenge C_A to complete the exchange.



Attack on Mutual Symmetric CRA

A potential attacker, "Mike" (posing as Alice), exploits the protocol by mimicking responses:

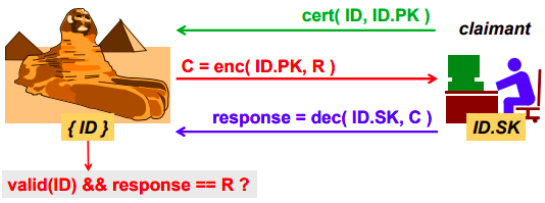
- The attacker intercepts Alice's identity (C_A) and Bob's challenge (C_B).
- The attacker uses the shared key K_{AB} to manipulate responses and mimic both parties.



1.9.3 Asymmetric CRA

Process:

- A **random nonce (R)** is generated by the Verifier.
- The verifier encrypts R using the user's public key ($ID.PK$) and sends it to the Claimant: $C = enc(ID.PK, R)$
- The Claimant decrypts C using their private key ($ID.SK$) and sends R back in cleartext: $response = dec(ID.SK, C)$
- The Verifier validates: $valid(ID)$ && $(response == R)$.



Applications

- Widely implemented in secure communication protocols like IPsec, SSH, and TLS.
- Fundamental in modern authentication frameworks such as FIDO.

Asymmetric CRA analysis

Security:

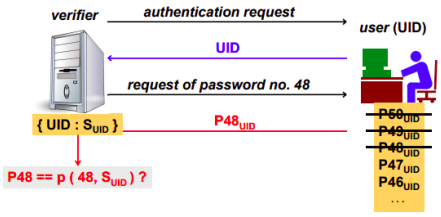
- It's the strongest mechanism.
- Does not require the Verifier to store any shared secret, reducing potential attack vectors.

Problems:

- It **slower** compared to symmetric methods.
- If designed inaccurately may lead to an involuntary signature by the Claimant.
- Trust issues managing root certificates, name constraint, and certificate revocation.

1.10 One-Time Password(OTP)

One-Time Passwords are temporary and valid for a single use in an authentication session. They mitigate risks like password reuse and passive sniffing but can still be vulnerable to man-in-the-middle (MITM) attacks. These passwords are often designed with random characters to prevent guessing, but this can make password insertion difficult for users.



1.10.1 S/KEY System

S/KEY System was the first OTP implementation by Bell Labs (1981). It pre-computes a sequence of passwords derived from a user's secret. Each password is validated and replaced with its predecessor, ensuring security without storing the secret:

$$Secret = S_{ID}$$

$$P_1 = h(S_{ID}), P_2 = h(P_1), ..., P_N = h(P_{N-1})$$

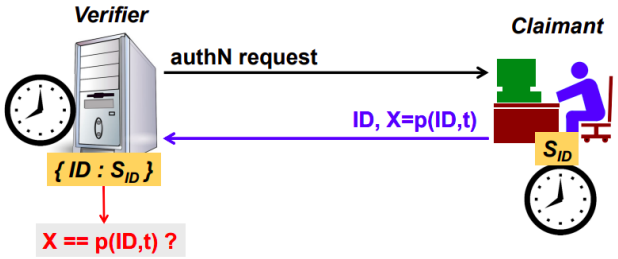
This approach minimizes verifier storage needs and offers robust protection, with users solely responsible for password retention.

One-time generation with S/KEY
In the S/KEY system, the user creates a secret passphrase (PP), which is combined with a server-provided seed to generate a 64-bit password. The passphrase is concatenated with the seed, and an MD4 hash is used to produce the password. The result is presented as six short words from a shared dictionary, making it easy to remember. This method allows secure password generation while using the same passphrase across multiple servers with different seeds. If the passphrase is compromised, security is at risk.

1.10.2 Time-based OTP (TOPT)

TOTP systems generate passwords based on the user's secret and the current time, requiring synchronization between the user and the verifier:

$$p(ID, t) = h(t, S_{ID})$$



Authentication Process:

- The claimant sends an authentication request with ID and the generated OTP x.
- The verifier checks if X matches the computed OTP for the corresponding ID and t.

Requirements:

- Local computation of OTPby the subscriber.
- Clock synchronization (or keeping track of time-shift for each subscribers).

Limitations:

- Only one authentication is allowed per time-slot, typically 30s or 60s.
- This time limit may not suit all services.

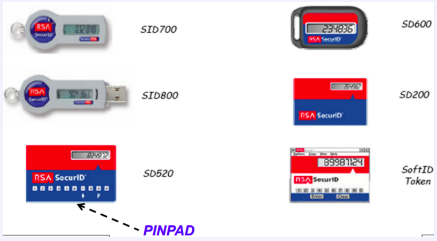
Vulnerabilities:

- Potential attacks on the subscriber and verifier:
  - Fake NTP servers or compromised mobile network femtocells.
  - Sensitive database storage at the verifier (e.g., the RSA SecurID attack).

Example: RSA SecurID

Authentication process:

- **The claimant sends to the verifier:**
  - Without PIN Pad: User ID, PIN, and Token Code (computed using seed and time).
  - With PIN Pad: User ID and a combined Token Code* (includes seed, time, and PIN).



1.10.3 Out-of-Band (OOB) OTP

Out-of-Band OTP requires a **secure channel** with server authentication to prevent MITM (Man-In-The-Middle) attacks. Traditionally, it uses text or SMS as the communication channel, but this method is increasingly vulnerable due to weaknesses in VoIP, mobile user identification, and the SS7 protocol. Nowadays, a push mechanism over a **TLS-secured channel** to a registered device is recommended for enhanced security.

1.10.4 Two-/Multi-Factor Authentication (2FA/MFA)

MFA enhances authentication (authN) by requiring multiple factors, such as a PIN, OTP, or biometrics, to verify identity. These factors can include something you know (like a PIN or password), something you have (like a token or phone), and something you are (like biometric data). MFA also protects the authenticator, for example, by using a PIN to safeguard it, but risks arise if the lock mechanism is weak or if there's no protection against multiple unlock attempts.

Importance of MFA: The iPhone Ransomware (2014)

In 2014, iCloud accounts with 1FA were hacked, allowing attackers to lock devices remotely. Victims were extorted for $100, but paying didn't help as the PayPal account was fake. This incident underscores the need for MFA to secure devices and prevent such attacks.

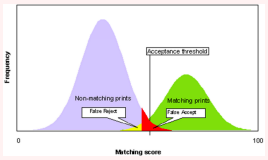
1.11 Authentication of human beings

To verify whether we're interacting with a human rather than a machine, there are two common approaches:

- **CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart):** A method where users must solve challenges like distorted characters in images to prove they are human.
- **Biometric Techniques:** These involve verifying human characteristics such as fingerprints, voice, retinal scans, iris scans, blood vein patterns in hands, heart rate, and hand geometry.

Problems of biometric systems

There are several issues with biometric systems:



- **FAR (False Acceptance Rate)** and **FRR (False Rejection Rate)**: These rates depend on the system's cost and can be adjusted, but biological factors like injuries or emotional changes can affect accuracy.
- **Psychological Acceptance:** Many people fear the "Big Brother" scenario—personal data collection and potential privacy invasions.
- **Irreplaceability:** Once compromised, biometric data cannot be changed, unlike a password or PIN. Thus, biometrics are primarily useful for local authentication but unsuitable for global identity systems.
- **Lack of Standardization:** High development costs and dependency on specific vendors are significant drawbacks in current biometric systems.

1.12 Kerberos Authentication System

Kerberos is a widely-used authentication protocol based on a Trusted Third Party (TTP) model. It's designed to ensure that user passwords are never transmitted over the network. Instead, the password is used locally for encryption.

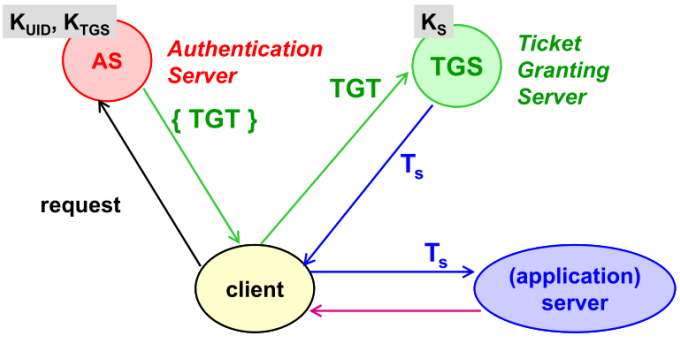
- **Realm:** Refers to a Kerberos domain, grouping together all systems that use Kerberos for authentication.
- **Credential:** A unique identifier for a user, typically in the format **user.instance@realm**.

1.12.1 Players

There are 4 key players:

- **Client (User/Application:)** the person or application trying to access a resource.
- **Authentication Server (AS):** Confirms who you are and issues a Ticket Granting Ticket (TGT).
- **Ticket Granting Server (TGS):** Issues tickets for specific services after seeing the TGT.
- **Service (Server):** The resource you want to access.

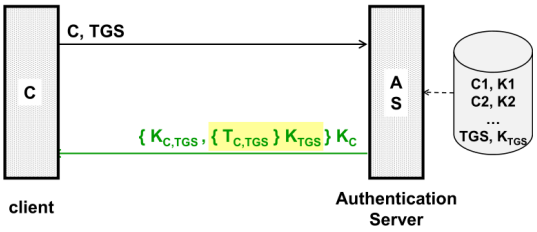
1.12.2 How it Works?



1. **TGT Request**: Client authenticates with the Authentication Server (AS) to obtain a Ticket Granting Ticket (TGT).

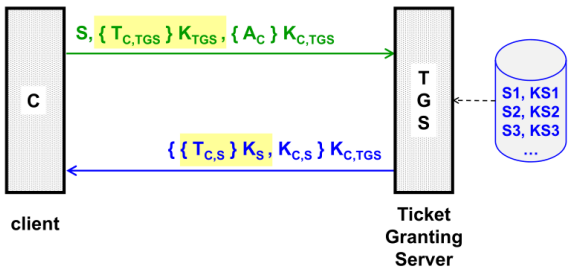
   The expression $KC, TGS, TC, TGS\ KTGSKC$ represents a **TGT**:

   - The enitre structure is encrypted using the client's secret key (KC) (This ensure that only the intended client can decrupt and use the TGT).
   - $TC, TGS\ KTGS$ is the **TGS** that contains the client's identity and other information. It's encrypted using the TGS's secret key ($KTGS$), ensuring only TGS can read and verify the ticket.



2. **Service Ticket Request**: TGT is sent to the Ticket Granting Server (TGS) to request a service-specific ticket.

   - The client sends: (S) $\rightarrow$ the indetifier of the target device, $TG, TGS\ \ KTGS$ $\rightarrow$ TGS ticket and ($AC\ KC, TGS$) $\rightarrow$ the authenticator.
   - TGS Verifies and Responds:
     - The TGS decrypts the TGS ticket using its secret key (KTGS).
     - It verifies the client's identity using the authenticator.
     - If successful, the TGS generates a Service Ticket ($TC, S\ KS$) encrypted with the secret key (KS), ensuring only the service can read it; and a new session key (KC, S), shared between the client and the service.

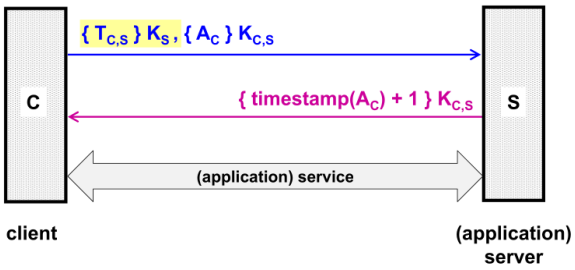


3. **Access Service**: Client uses the service ticket to authenticate and access the resource.

   The client send:

   - The service ticket encrypted using the service's secret key (KS), so only the service can decrypt and validate it.
   - The authenticator encrypted using the session key (KS, S) shared bewteen the client and the service.

   The service responds with a response in which confirms successfully authentication. It encrypts the message with the session key (KC, S) to ensure it's secure and readable only by the client.



1.12.3 Single Sign-On (SSO)

SSO allows users to authenticate once and access multiple services without repeated logins. Types of SSO:

- **Fictitious SSO:**
  - Relies on tools like password synchronization or management (e.g., password wallets).
  - Limited to specific applications.
- **Integral SSO:** Uses advanced multi-application methods like Asymmetric Challenge-Response Authentication (CRA) or Kerberos. Often requires application changes.
- **Multi-Domain SSO:** Expands SSO across domains using technologies like SAML tokens, which generalize Kerberos tickets.

N.B. Single Sign-On (SSO) is not exclusive to Kerberos, but Kerberos is one of the prominent technologies that implements SSO capabilities.

1.13 Authentication Interoperability

Authentication interoperability define methods, standard, and protocol for performing authentication securely and efficiently. Let's start to analyze some framework.

1.13.1 OATH

The Open Authentication (OATH) framework provides standards for one-time password (OTP) and symmetric key management.

- **HOTP:** Uses a shared secret key (K) and a counter (C) to generate an OTP.
  Function:
  $$HOTP(K, C) = sel(HMAC - h(K, C)) \ \& \ 0x7FFFFFFF$$
  The result is truncated and transformed into an N-digit code (e.g., 6 digits)[1].
- **TOTP:** Similar to HOTP but uses time intervals (TS) instead of counters.
  Function:
  $$C = (T - T0)/TS$$
  With default values: $T0$ -> Unix epoch, TS -> 30-second intervals, T -> unixtime (now).
- **OCRA** (OATH Challenge-Response Algorithm)
- **PSKC** (Portable Symmetric Key Container): XML-based format for symmetric key transport.
- **DSKPP** (Dynamic Symmetric Key Provisioning Protocol): A client-server protocol for securely provisioning symmetric keys.

1.13.2 Google Authenticator

Supports HOTP and TOTP with adjustments for usability:

- **K:** Base-32 encoded.
- **C:** 64-bit unsigned integer.
- **sel(X):** Uses the 4 least-significant bits of X to locate a portion of the result.
- **Defaults:** TS = 30 seconds, N = 6 digits (zero-padded if necessary).

[1] K-> shared key, C -> counter, sel -> function to select 4 bytes out of a byte string

1.13.3 FIDO (Fast Identity Online)

FIDO, developed by the FIDO Alliance, improves authentication by offering secure, password-less, and multi-factor methods. It uses biometric data locally to unlock cryptographic keys and employs asymmetric cryptography for signing challenges or transactions. FIDO prevents phishing by ensuring that authentication responses cannot be reused. Each response is a unique signature created over various data, including the Relying Party (RP) identity, making it specific to the service being accessed. Additionally, a new key pair is generated during each registration, which prevents the association of a user's identity across different services or accounts.

FIDO's frameworks include:

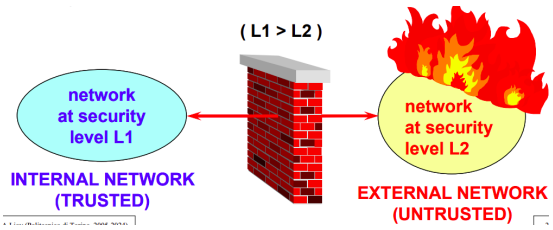
- **UAF** (The Universal Authentication Framework): for password-less login.
- **U2F** (Universal 2nd Factor) for device-based two-factor authentication.
- **FIDO2**: integrates U2F with WebAuthn for robust web authentication.

Chapter 2

Firewall and IDS/IPS

2.1 What is a Firewall?

A firewall acts as a protective barrier, much like a physical wall against fire. It is primarily a controlled connection point between networks with differing security levels. Its key functions include:



- **Boundary Protection:** Serving as a network filter between trusted (internal) and untrusted (external) networks.
- **Compartmentalization:** Dividing network zones based on security levels to enforce stricter control.

2.2 Ingress vs Egress firewall

Firewalls can be classified based on the direction of traffic they manage:

- An **ingress firewall** focuses on incoming connections, typically regulating access to public services provided by the network or supporting exchanges initiated by internal users.
- An **egress firewall** monitors outgoing connections. It's typically used to check the activity of internal personnel [Works well for channel-based services (e.g., TCP applications) but faces challenges with stateless, message-based services (e.g., ICMP, UDP)].

2.3 The three principles of the firewall

1. The firewall should be the only connection between the internal and external networks.
2. Only the "authorized" traffic is allowed to pass the firewall.
3. The firewall itself must be secure against potential vulnerabilities.

These principles were outlined by *D.Cheswick, S.Bellovin.*

2.4 Authorization policies

- **Permitlist/allowlist**: All that is not explicitly permitted, is forbidden.
  - It offers higher security but it's difficult to manage.
- **Blocklist/denylist**: All that is not explicitly forbidden, is permitted.
  - It's less secure but it's more easy to manage.

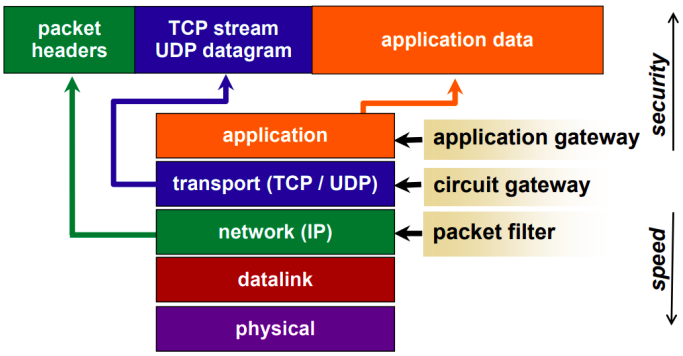
2.5 Basic components

Firewalls include several key components that work together to provide security:

- **Packet Filter/ Screening Router / Choke**: Filters traffic at the network level based on packet attributes such as IP headers and transport headers.
- **Bastion Host**: A secure system with auditing capabilities, often positioned to handle critical traffic.
- **Application Gateway (Proxy)**: Acts on behalf of an application, managing access control and providing detailed packet inspection at the application level.
- **Dual-Homed Gateway**: A system with two network interfaces and disabled routing, isolating internal and external networks. (In the way we can decide which packets are sent from a network to another).

2.6 A which level the controls are made?

Firewalls at higher levels provide more security but tend to be slower, while firewalls at lower levels offer less security but greater speed.



To undersand:

security is better up → speed is better down
security is better down → speed is better up

2.7 Network-Level Controls

Firewalls operate at various levels of the network stack applying different controls based on the type of firewall used:

- **Packet Filters**: Operate at the network level, inspecting packet headers.
- **Stateful Packet Filters**: Track connection states for more dynamic filtering.ù
- **Circuit-Level Gateways / Proxies**: Work at the transport level, ensuring secure connection establishment.
- **Application-Level Gateways / Proxies**: Provide inspection and control at the application layer, often with more granular rules.

2.7.1 Packet filter

A packet filter inspects packets based on headers, such as IP and transport headers. These filters are traditionally available on routers and now in most OS. They allow for rules like permitting incoming connections to specific services (e.g., a web server) or limiting DNS queries from specific internal servers.

- **Pro:**
  - Independent of applications, scalable, and cost-effective (available in many OS and routers).
  - Good performance and low cost.
- **Cons:**
  - Vulnerable to attacks like IP spoofing or fragmented packets.
  - Difficult to support services using dynamically allocated ports (e.g. FTP).
  - Complex configuration and hard to implement user authentication.

2.7.2 Circuit-Level Gateway

A circuit-level gateway acts as a transport-level proxy, creating secure communication channels between client and server without inspecting the payload. It protects against Layer 3/Layer 4 attacks like IP fragmentation or TCP handshake exploits.

- **Pro:**
  - Provides protection by isolating the server from attacks.
  - Offers client authentication and eliminates many low-level attack vectors.
- **Cons:**
  - Still shares many limitations of packet filters and requires modifications to the application for full functionality.

2.7.3 Application-Level Gateway

An application-level gateway (or proxy) operates at the application layer, inspecting the payload of packets. These proxies often require modifications to client applications and can enhance security by checking the semantics of application data (e.g., HTTP methods) or performing peer authentication.

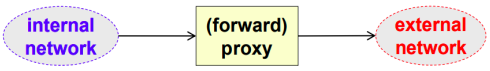
- **Pro:**
  - Strong security against application vulnerabilities (e.g., buffer overflow attacks).
  - Fine-grained access controls and the ability to mask internal IP addresses.
  - May provide protection against attacks like buffer overflows.
  - Not transparent to the client and may break the client-server model.
- **Cons:**
  - Requires specific proxies for each application and can introduce delays when supporting new applications.
  - High resource usage and lower performance due to user-mode operations.

Variants of application-level proxies include **transparent proxies**, which are less intrusive to clients, and **strong application proxies**, which focus on checking data semantics, not just syntax.

2.7.4 HTTP Proxies

An HTTP forward proxy is a server that acts as an intermediary or front-end for client requests. It receives requests from internal users and forwards them to the real external server (So it's a egress control). **Benefits:**

- **Shared Cache**: External web pages are cached, reducing load times for all internal users.
- **Authentication and Authorization**: Enforces user authentication and controls access for internal users.



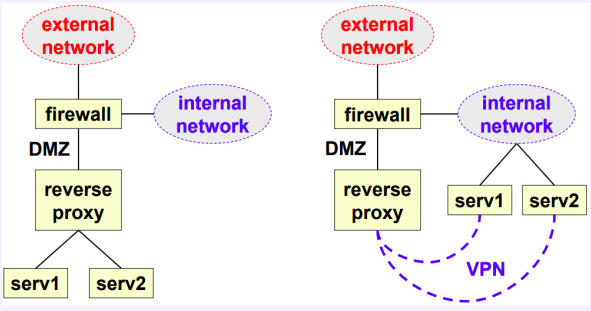
Type of Proxies

- **Forward Proxy (HTTP Proxy)**: This type of proxy controls outgoing traffic from the internal network to the external one, enforcing access controls and caching content. It provides features like user authentication and authorization, content filtering, and bandwidth control.
- **Reverse Proxy (HTTP Reverse Proxy)**: A reverse proxy sits in front of web servers, providing additional services like load balancing, content inspection, TLS acceleration, and caching. It can obfuscate the internal server structure, improving security, and support performance optimizations like dynamic page feeding based on client speed.

Reverse proxy: possibile configuration

Key components:

- **External Network (Red Cloud)**
- **Firewall**
- **DMZ (Demilitarized Zone)**: A buffer zone between external and internal networks where systems that interact with the external network are placed for added security.
- **Reverse Proxy**
- **Internal Network**: The secure area where your actual servers (e.g., serv1, serv2) reside.



How configurations work?

Left configuration: Direct reverse proxy setup

- **Flow:** External user → Reverse Proxy → Internal Servers (serv1, serv2).
- **Benefits:** Internal servers are hidden and protected, while performance and security are improved.

Right Configuration: Reverse Proxy with VPN

- **Flow**: External user → Reverse Proxy → Encrypted VPN connection → Internal Servers (serv1, serv2).
- **Benefit**: Adds an extra layer of security through encryption for communication between the reverse proxy and internal servers.

2.7.5 WAF (Web Application Firewall)

A WAF is a module installed at a proxy (forward and/or reverse) to filter the application traffic. Filters the followings types of traffic:

- HTTP commands.
- HTTP request/response headers.
- HTTP request/response content.

Popular WAF example: ModSecurity

ModSecurity is a widely-used WAF plugin for web servers like Apache and NGINX, offering protection through predefined rules like the OWASP ModSecurity Core Rule Set (CRS).

2.8 Firewall's Architectures

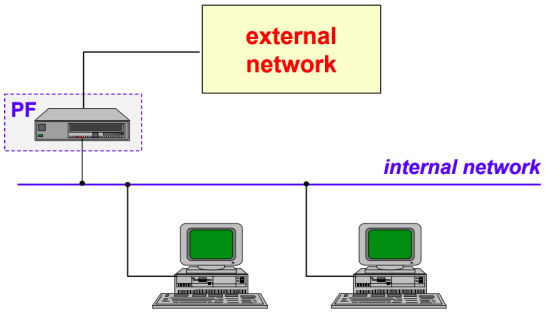
2.8.1 Packet Filter

What it does?

This architecture uses a simple packet filter to screen traffic at both the IP and higher protocol levels (like TCP)

Key Points:

- If implemented with a router then it's a "screening router" and there's **no need for extra dedicated hardware**.
- The packet filter element represents a **single point of failure**.
- **There's no need for proxies** or application modifications.



Beware

Simple, cost-effective, but insecure!

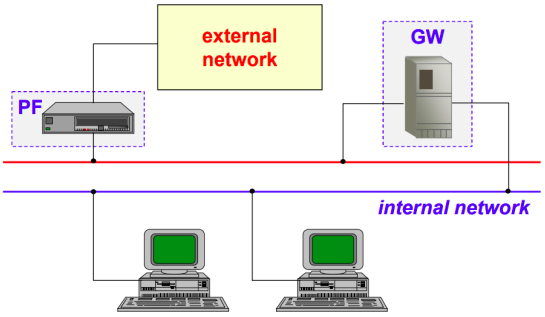
2.8.2 Dual-homed Gateway

What it does?

Uses a system with two network interfaces (hence "dual-homed") to provide a basic firewall between external and internal networks.

Key Points:

- Easy to implement with small hardware requirements.
- The internal network can be masquerade.



Beware

Inflexible, with higher management overhead and reduced flexibility in handling complex network setups.

2.8.3 "Screened host" architecture

What it does?

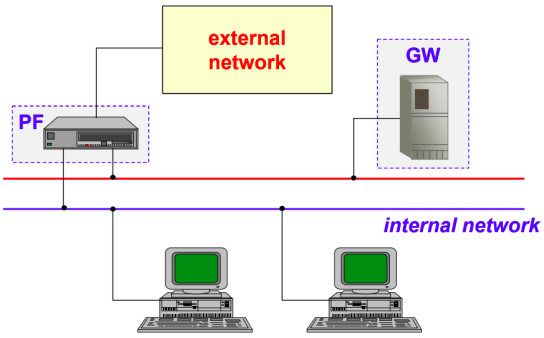
Uses two primary components:

- **Router**
- **Bastion host**

This setup aims to provide enhanced security by controlling the flow of traffic between the internal and external networks.

Key Points:

- Traffic from the internal network (INT) to external (EXT) is blocked unless it's from the bastion host. Similarly, external traffic is blocked unless directed to the bastion.
- More flexible (skip control over some services / hosts)



Beware

- **More Expensive and Complex** to manage: Requires two systems (router + bastion host) instead of just one.
- **Limited Masking**: Only the host and protocols that go through the bastion host can be masked for security (such as hiding internal IP address or data). However, if the packet filter (PF) uses NAT, it can mask additional traffic and hide internal network details.