Appunti Information System Security

Brendon Mendicino

November 28, 2022

Contents

1 Introduzione **3**

2 Some classes of attacks **4**

3 Foundations **7**
3.1 Public key criptograpy 10
3.2 Public key algorithm 10
3.3 Diffie-Hellman (DH) 11

4 Message Integrity **11**
4.1 SHA-3 12

5 Integrity and Secrecy **12**
5.1 Authenticated Encryption 13

6 Definitions of authentication **14**

7 Esercizi **17**

1 Introduzione

Security Principles:

- security in depth: there must be more layers in security;
- security by design: you design the system to be secure;
- security by default: the security feature are turned on even if you don't want to;
- least privilege: when someone operates in the system, that program must have the minimun permession to allow the basic work;
- need-to-know: which data can be accessed by a program or by a huan, a program must operate only on the sufficient data it needs;

Security propreties:

Peer authentication When a person tries to perform an operation on system it has to proof it's identity to the system, but even the system hhas to authenticate itself to the accessing user, this is called mutual authentication (when both peer have to authenticate).

Data authentication Non-repudiation: undeniable proof of the data creator; there needs to be several facetes:

- authentication;
- integrity of the data;
- identification;
- ...

Authorization (access control) Someone has to give you the Authorization to perfrom some kind of action.

Privacy (communication) Someone listening to a conversation means to vilate the privacy.

Privacy (data, actions, position) Keeping some data encrypted inside a disk to protect them from someone accessing them. By law the internet provider are required to keep track of all the visited websites up to some years in the past, those can be accessed in case of an investigation. The internet provider needs to keep track at any time of the position of the connected device.

Integrity (data modification) The manager of the network can in some way modify the data shared by the systems, it can also cancel some parts of the data or filtering specific ones.

Replay attack If the data is encrypted they can no longer be modified by an external actor. The data cannot be changed but it can be send more that once, a **replay attack**. This can be solved by giving an ID to every transaction.

Data protection:

- data in transit: data must be sucered while travelling between systems;
- data at rest: even whene the data is parked on the disk is not modificated in any way;
- data at work: data at some point is copyed in the RAM to perform some kind of action, and if an external program modify them inside the RAM it will be never kwnown;

Basic Problems Networks are insecure:

- most of the communication are in clear;
- LANs operate in broadcast;
- geographical connections are not made through end-to-end dedicated lines but through shared lines and third-party routers;
- weak user authentication;
- there is no server authentication;
- the software contains many bugs;

2 Some classes of attacks

IP spoofing : creating an IP packet that contains a different address, typically the level 3 and level 2 address. It's a tecnique to hide somenes identiy. The attacks are: data forging a packet in someone else name, or currupting them. The countermeasures are: to never use an address-based authentication.

Packet sniffing : the ability to read the packets addressed to another network node, this is very to do in broadcast area networks (all intranet networks are broadcast). This attack allows to read the payload of the packets sniffed, the countermeasures are: to don't use broadcast networks (not possible), to encrypt the payload of the packets.

Denial-of-Service (DoS) : keeping an host busy so that it cannot perform any other relevant action. Some examples are:

- mail or log saturation;
- ping flooding;
- SYN attack

The purpose of the attack is to block the use of a system. There are no countermeasures, the only possiblity is to monitor.

Distributed Dos The attacker is using many computers (deamons, zombie, ...), usally infected with a malware, those machines contrlled by another master are called **Botnet**. To control the machines the attacker may uses encrypted channels, using a CC structure (Command and Control).

Attack: some of the deamons are turned into masters contrlling all the deamons, the attacker unplugs from the network avoind being monitored, some time later the masters give the command to the deamons and they start attacking all the vectim.

Shadow/fake server There are two kind of tecnique to fake a server:

- being in the same network (sniffing packet), if I (the attacker) respond to the user faster than the server, now your connection is opened with me, now the victim exchanges informatios with the attacker;
- creating a fake DNS to my own versions of the servers, providing wrong services, capturing data;

The countermeasures are: server authentications.

Connection hijacking (MITM, Data Spoofing) The attacker takes phisycal control of some node, becoming a **Man In The Middle**, letting the comunications of two cominacotrs get through my node, reading all the traffic, being able to perform any action. The countermeasures: even if the channel is already opend and the peers are already identified, the attackers can still take control of the comunications, to prevent this kind of attaks there needs to be some kind of authentication inside of the packets, even then also serializations is needed, beacause the attacker can swap the order of the packets.

Trojan A trojan is program containing a malicious payload. The networks are becoming more protected but terminals are becoming more vulnerable (smartphone, smartTV, IOT, ...). The attacker could trick the user into download an extension, or pirated copies of games, this programs could contain things like keyloggers. This attacks are colled:

- MATE: Man At The End;
- MITB: Man In The Browser;

Zues Also kwown as Zbot, this malware is installed on millions on devices, it can be user to:

- perfom keylogging or form grabbing;
- to load other malware, like CryptoLocker ransomware;

It was very difficoult to discover.

Sotware Bugs Bugs in a sotware are explotable, thanks to them DoS can be performed.

Virus & Co. (malware)

- virus: damages the target and replicates itself thanks to humams, requires involuntary complicity;
- worm replicates itself sucking all the available resources and the propagating;
- trojan: a vector for malware;
- backdoor: entry point not known but the developers;
- rootkit: something isntalled in the computer that provides root access to the attacker, remaining unnoticed;
- PUA: Potentially Unwanted Applications, some applications not very dangerous but performing annoying operations;

Ransomware It's a kind of malware that is oriented on getting a ransom, typically performed by ecrypting the disk or changing passwords. The only way of unlocking the device is to reset it, or

Solcial Engineering :

- phishing:
- psychological pressure: asking for help, or impersonating a person of interest imposing to perform some action;

Fake Mail It can be possible to take an old email (send to the victim), copying it and changing the attachment with a malware.

Important classes of attacks:

- Stuxnet (2010): new king of attack, it was a worm + virus for Windows, it was the first time to attempt to damage a SCADA system. It contained in itself an attack based on a known vulnerability (patched), a known vulnerability (not patched), and 2 zero-day vulnerability. This malware was used to destroy a nuclear plant in Iran (in fact in Iran the spread was 51%), destroying most of the machines, the facility was isolated from internet, the only way it could have accessed it was via USB, the malware spread thanks to shared disks and old sotfware with bugs, the malware aslo used digital signature validate by Microsoft;
- Mirai: it was a cyberworm exploiting IOT vulnerabilities, trasforming the victim in botnets for a large scale attack. All this devices ware because they have little to none protection. The software was compiled with static libraires, cross compiled and also open source. It also abserves the victim before contactin the C&C, if it detects some secuirty protection it contacts a fake C&C.

The pillars of security

- Planning;
- Avoidance;
- Detection;
- Investigation;

3 Foundations

Cryptography It's a mathematical algorithm that consists in: a clear message gets encrypted with a key, send to the receiver, that will decrypt the message with a second key to read the original message.

- message in clear: called: **plain text** or **P**;
- ecrypted message: called **chipertext** or **C**;

Kerchoff's principle:

- the keys needs to be secret;
- the keys are managed by a trusted system;

- the keys needs to be of adequate lenght;

If this three priciples are met then the ecryption and decryption algorithms can be public.

Security through obscurity (STO)

Secret key/symmetric cryptography The key is shared

$$C = enc(K, P) \text{ or } C = PK$$

$$P = dec(K, C) = enc^{-1}(K, C)$$

The main problem of symmetric cryptography is how to share the shared key.

DES is an **obsolete** symmetric algorith with 64 bit block and 56 bit key. AES is a **state of the art** symmetric algorithm that uses 128 bit block and 128-192-256 bit key.

The XOR function If the input is random the output has the same probability (the 0:1 have 50% probability of outcome).

DES It used 64 bits as a key but only 56 bits were for the key, the other 8 bit were parity bits, this algorithm was created to run fast on hardware.

Triple DES (3DES) 3DES works with 2 keys, following the algorithm:

$$C' = enc(K_1, P)\ C'' = dec(K_2, C')\ C = enc(K_1, C'')$$

3DES with 3 keys:

$$C' = enc(K_1, P)\ C'' = dec(K_2, C')\ C = enc(K_3, C'')$$

The reason why it's not good to encrypt twice with the same algorithm, the reason is beacause of the attack **meet-in-the-middle**, which allows to decrypt the data with 2^{n+1} attempts if the keys are n-bit long, this is not a good enough improvemnt for the incresed volume of work added, or worse if the algorithm is part of a group there could exist a K_3:

$$enc(K_2, enc(K_1, P)) = enc(K_3, P)$$

Meet-in-the-middle attack Hypothesis:

- n-bit key;
- known P and C such that $C = enc(K_2, enc(K_1, P))$;

Note:

- $\exists M$ such that $M = enc(K_1, P)$ and $C = enc(K_2, M)$;

Actions:

- compute 2^n for $X_i = enc(K_i, P)$;
- compute 2^n for $Y_j = dec(K_j, C)$;
- if X_i and Y_j are matched then K_i and K_j are found;
- false positives can be discarded with other (P, C) couples;

Application of block algorithms How a block algorithm is applied to a data quantity different from the alogorithm's block size?

- ECB (Electronic Code Book): if we have to encrypt some data, we simply split the data in equal size blocks and encrypt every block with the same key, this is very insecure, for example an attcker could swap two ciphertext, also if there is a known plaintex the attacker could find the matching key and then decrypt all the ciphertext;
- CBC (Cipher Block Chaining): we also split the data, we take every blcock, and before ecrypting the the block we xor it with the previous encrypted block, for the initial block we use a IV (Initialization Vector, C_0), for this reason the first block is the most vulnerable. In decryption to recover the plain text, the decrypted block will need to be xor-ed with the previous block (proprety of the xor function);

Padding with explicit lenght

- Schneier: the last byte has the value of bytes of padding, the other bytes are null;
- SSL/TLS: the bytes of padding are all at the value as the lenght of the padding;

If the data is an exact multiple of the block lenght, then the padding will have to be added in anyhow, if the that is the case the last block will be entirely of padding.

Ciphertext stealing (CTS) CTS permits to use any block algorithm without requireing any padding to be added.

CTR (Counter mode)

AES = rinjdael It can have 3 possible key lenght (128/192/256) and a block size of 128 bit.

3.1 Public key criptograpy

It is an asymmetric algorithm, it means that the keys are different, one is used to encrypt and the other one is for dercrypt, that is why they are called **key-pair**. They take a lot of time to compute, this is why it is used to encrypt small quantities of data, usually used to share symmetric keys.

The keys are called **public key** (SK) and a **private key** (PK), one encrypts the other decrypts, both of them can be interchangable so we must chose which one of them must be public and which one must be kept secret.

Digital signature The idea of digital signature: the asymmetric keys provides a way of signing messages, if I encrypt my data with my private key, anybody else can decrypt that data the public key, this works because this only works with key-pair, and because the key used to encrypt is private nobady else know it, and the derciption only with the respective public key.

Confidentility without shared secrets To share secret messages with someone: I can encrypt the date with the public key, sending the message to the receiver I am sure that nobady else can read the message, and when it reaches the receiver it can decrypt the message with his private key.

3.2 Public key algorithm

RSA It is based an the factorization of the product of two prime numbers, it can be used for both digital signature and confidentiality.

DSA (Digital Signature Algorithm) It is based on taking power, logarithm of the result, it can only be used for the digital signature.

Both DSA and RSA are going deprecated.

Certificate How to bind a public key to the identiy of the person? To avoid this kind of problem a **public-key certificate** has been created.

Secret key exchenge One way of sharing keys between two users is to encrypt the symmetric key with the public key of the receivers, the receiver can then decrypt the symmetric key with his private key and then both of them can start sharing messages with symmetric keys. This is the base of secret comunications nowadays. This is still not secure beacuse the sender can choose the password and share it with somebady else.

3.3 Diffie-Hellman (DH)

There are some common parameters know. Sender $A = g^x \mod p$; Receiver $B = g^y \mod p$. Then A and B are shared,

This algorithm is not a good algorithm in the case of a man-in-the-middle attack, this can be rendered more secure with a certification.

DH, RSA and quantum computing Quantum computer pose a threat to those kind of algorithn, there is a quantum algorithm called **Shor's algorithm** that can foctorize a number in $O(N^3)$ time.

Elliptic Curve Criptography Isntead of using modular arithmetic but we take computation on a plane that represent the graph of an elliptic curve, this kind of algorithm is more secure than RSA or DSA, in fact the keys or shorter.

- **ECDSA**: digital signature;
- **ECDH**: for key agreement;
- **ECMQV**: authenticated key agreement;

4 Message Integrity

Even if something has been enrypted it does not mean it cannot be damaged. The normal way of computing data inegrity is to create a **digest**. A digest is always of a fixed lenght, and it is created with a mathematical algorithm, in cybersecurity a digest is always created with a **cryptographic hash function**. The structure of a hash function is always the same: it splits the data in blocks and in computing a new block it reuses the previous block. The families of the hash functions are:

- sha-2: sha-256, sha-512, ..., good enough, but they are based on the algorithm of sha-1 which has already been attacked;
- sha-3: most secure;

An important part of the hash function is create less **collisions** possible, a collision accurs when two differnt messages create two identical digests.

4.1 SHA-3

The selected sha-3 algorithm was **Keccak**, it is has simple implementation and it has very good performance on a veriety of very differnt hardware.

KDF The hash functions are also used for **deriving passwords** (**KDF** Key Derivation Function). The KDF is composed of:

- P = password;
- S = salt;
- I = number of iteration;

The passwrod is derived in folling way: (da ricontrollare)

$$hash_{I\text{-times}}(P||S)$$

MAC,MIC,MID ...

keyed-digest HMAC A hmac is a function to compute the digest, with HMAC the Hash function needs to be specified like: HMAC-SHA256.

CBC-MAC CBC can also be used to create the digest of some data, you discard all the block exept for the last one, in this way all the previous bloks are used.

5 Integrity and Secrecy

We use:

- K1: symmetric encryption;
- K2: key-digest

How do you use the two keys?

1. authenticate and ecrypt:
2. authenticate then encrypt:
3.

Combining authentication and integirty is not good. The solution is **Authenticated Encryption (AE)**.

5.1 Authenticated Encryption

da finire

- peer authentication:
- data oridin authentication:
- confidentiality:
- integrity:
- authorization:
- non-repudiation:
- availability:
- accountability:

6 Definitions of authentication

Usally an "actor" is needs to be handled authentication, called authN (or authC), the an authorization is give authZ.

knowledge: something the user knows; ownershiop: something the user possesses (like a bank card chip); inherence: something the user is; This can also be applied to hardware and sotfware.

Risks for those: The risk of kwoledge (e.g. passwords): how to protect known passwords. ownership (e.g. smartphone): for example owning a smarthone with a malware that can read some data, or it could be stolen, ... inherence (e.g. biometrics): counterfeiting (the fingerprint can be reproduced) and privacy, this is way inherence must only be used locally;

Nowasdays systems are getting more and more complex, digital authentication model (NIST SP800.63B) an appllicant when trying to enroll and proving his identiyt, the CPS (credential sevice provider) gives yuo authenticator enrollement/issuance, so it becomes a subscriber, a relying party (like SPID) takes an authneticated session from a subscriber.

Or a claimant can rely on a relying party (like SPID), a that comunicates with a verifier (like isp.polito.it), when the relying party instates ..... da finire .....

Authentication protocal, is protocal that runs between a verifier and a client, the user has a UserID and a secret associated with the UID, the verifier containe a UID with a the result of a function that take the SUID, { UID : f(SUID) }, 1. user: authentication request; 2. user: UID 3. verifer: proof request; 4. user: proff = F(SUID)

1. Reusable password as SUID: user: has a PIUD, verifier has (UID: H(PIUD)) the client creates and trnsmits the password, the server needs to verify the proof: 1. 1. password in clear text 2. proof = password ?

2. 1. server kwows the digest of a password 2. f(proof) = H(PUID) ?

The only advanteage is that is simple to handle for a user, the cons are: some password are easy to discover, the user has to remember lots of password.

password best practices: 1. alpha char (up + lower) + number + spectial char; 2. they need to be long; 3. never use dictionary words 4. frequently changed 5. don't use passwords

server side: 1. never store the password in clear 2. sotore a digest of a password 3. dictionary attack: to protect from this attack a salt needs to be inserted 4. using a protected store accessible by one password

Dictionary attack: Hhyp:the attacker knows the hash algo and all the digests the attacker uses a large dictionary of words, and stores the digest associated with each word. After the precomputation the attacker, HP = hash of unkown password w = lookup (DB, HP) if we the attcker find a match than he found the original password.

To defeat this attack a salt needs to be used for every UID, the salt is a random long number generated by the verifier the verifier stores HP = hash (pwd —— salt

), { UID, HP(UID), salt(UID) }. The reason why this is effective is because the dictionary attack is based on precomputation, if the data gets stolen it will take a lot of time to recompute all the hashes, and by the time the attacker found a match the password may have been changed, aldo every UID has a different salt.

Originally linux stored passwords in /etc/password with a DES hash function, since that file needs to be word-redalbe (UID, GID, home shell, ..) they have been moved to /etc/shadow (it's not readable by normal users). (seee crypt(5)) they are stored as idsalt$hashedpwd

example mysql stores username and password in the "user" table, it stores the passwd = p with sha1(sha1(p)) (very bad). sha1("Superman!!!") = sha1(sha1("Superman!!!"))

Strong (peer) authN we consider strong authentication for ECB (european central bank), stated that password must not be used for money transaction operations. Authenticaiton shold be based on two or ways, the elements should not be related, non-reusable, strong authentication procedure.

PICIDSS (v3.2): requires multifactor authenticatoin, for accessing into the card-holder data environment (one exeption is via using a console),

an example of multifactor is when picking money from a atm (card chip + password).

Challange-repsonse authentication (CRA) The most strong kind of authentication, the claimant has a key Kc, the verifier contain a Kv associated with an ID. 1. cla: ID 2. ver: challange 3. cla: response = f(challange, Kc) 4. ver: response == g(challange, Kv) ? the key is not transmitted (protected form sniffing), the cahllange must not be repeatable, usally is a random nonce (number used olny once), the function f must not be invertable (someone sniffing could invert the function from the nonce and find the key),

symmetric CRA, uses a hash function instead of encryption, but Kc must be known in cleartext to the verifier (there are attacks against $ID : K$ table at the verifier), SCRAM (Salted CRA Mechanism) to use CRA to it's full potential we can combine it with aysmetric encryption the claimant has a ID.SK 1. claimant: cert(ID¡ ID.PK) 2. ver: C = enc(ID.PK, R) 3. cla: response = dec(ID.SK, C) 4. ver: valid(ID) && repsonse == R ?

R is random number Asymmetric CRA is the strongest machanism known, it's implemented for peer authentication (client and sever). The problems are: slow, PKI (trused root, name constrain, revacation), avoi, the table does not contain any information (it requires integrity).

One-time passwords (OTP) the user has got a list of passwords, each password is usable only one time, when the user tries to authnticate sends a password to the server, the sever contains a SUID that is associated to the password,

this is immune to authentication subject .... finish ....

if the user needs to have all the paswords already precomputed (banks give passwords card). On intelligent and trusted hardware there are applications that can generate passwords ad-hoc.

Time-based OTP the user has a passwords with a count down timer associated with it, if the time expires the password is no more valid, 1. ver: authN request 2. cla: ID, X = p(....

the passwords cannot be precomputed, and claimant and verifier need to be synchronized (need to have the same time), the server checks that the time slot is correct X == p(ID, t) —— X == p(ID, t-1) —— X == p(ID,t+1), usally hours and minutes are used, In order to av oid replay attacks the authentication is one per timeslot. It is based on time, so it can be attacked with a time attack (servers get time via NTP servers (Network Time Protocal)) causing DoS, also the device can get the time from mobile network vulnerable to a femtocell attack, also ottaining DoS, but if a future time slot is used by the attacker he can sniff the network and the use the password himselft in that time slot.

Event-Based OTP Time-base OTP is not enough when trying to make more request in the same time slot, this is based on a counter that is advanced every time an operation is performed. p(ID,C) = h(C, SID) requires local computation, it need OTP pre-computation also by an attacker who temporarely have access to

Out-of-Band OTP It allows to avoid local computation of the OTP, cla: UID, PUID when the cla tries to authenticate he gives to the OTP a UID and a PUID, than the verifier sends on out of band (outside of the current channel like a SMS) a OTP, the number has to be set before.

the channel must be secure OOB channel is text/SMS message, this are not very secure because they are based on SS7 (insecure protocol). other times the message is sent via a PUSH message

Multi authN

Biometric Biometrics always take insto consideration two factors: * FAR = false acceptance rate * FRR = false rejection rate they are tightly coupled with the cost of the device

HOTP the HMAC is used. K: shared secret key C: counter (monotonic positive integer) h: has function (sha1 by default) sel: funciton to select 4 byte string HOPT(K,C) = sel(HMAC-h(K,C)) & 0x7FFFFFFF The mask is used to sed the MSB=0 to set the numbero to be positive

Used for Eventbased OPT

TOTP as HTOP but the counter is the number of intevals TS elapsed since a fixed origin T0 C = (T - T0) / TS RFC-6238 T0 = unix epoch T = unixtime(now) Ts = 30 seconds C = floor(unixtime(now) / 30)

Google authenticator supports HOTP and TOTP the key is provied in base 32 C is provided as uint_64 sel(X) offset = 4 LSB of X return X[offset ... offset+3] TS = 30 seconds N = 6 if the generated code is contains lassa than 6 bytes then its left padded with zeros

FIDO Fiast IDentity Online Defined: biometric auth = passwordledd user experience 2-factor authN = 2nd factor user experience

based on personal devices caplable of asymmetric crypto

UAF U2F ASM
used on many websites to authenticate like google, twitter, aws, ...

7 Esercizi

- Symmetric:

  alg-keylen-

- Hash functions:

  - B = data blob;
  - D = digest;

  D = h-alg(B)

- Keyed digest:

  D = kd(K, B) or D = mac(K, B) D = keyed-digest-alg(K, B) $\implies$ hmac-sha-1(K, B)

- Asymmetric encryption:

  C = asymm-enc(PK, P) $\implies$ rsa-2048-enc(Alice.PK, P)

- Basic operation:

  - random();
  - save();
  - send();
  - receive();
  - "——" or "+" concatenate;

- Extract valuem from certificates (PKC = public key certificate):

  get-hashalg(PKC), get-pubkey(PKC)

1.

```
IV = random()
C = aes-128-cbc-enc(K, P, IV)
Alice.send(IV, C)
Bob.receive(IV, C)
P = aes-128-cbc-dec(K, C, IV)
```

2. Alice wants to write data P protected on disk, size of data must not be increased, key must be generated:

```
Alice passwd
salt = random()
IV = random()
K = h-sha-512(passwd || salt)
D = hmac-sha-512(passwd || salt)
C = aes-256-cbc-cts-enc(K, P, IV)
writes(C, D, IV, salt)
```

3. alice wants to send bob a message P protected for integrity and authentication, they sahre symm key K

```
alice
h = hmac-sha-256(K, P || "hmac-sha-256")
Alice.send("hmac-sha-256", h, P)
Bob.receive("hmac-sha-256", h, P)
h' = hmac-sha-256(K, P || "hmac-sha-256")
if h != h' FAIL
```

4. Alice wants to send bob a digitally signed message P, alice has Key Pair, alice send OOB PK

```
S = rsa-2048-enc(Alice.SK, h-sha512(P))
Alice.send(P, S, "rsa-2048", "h-sha512(P))
Bob.receive(P, S, "rsa-2048", "h-sha512(P))
digest = rsa-2048-dec(Alice.PK, S)
if digest == h-sha512(P) then SIGNATURE VERIFIED
```