L03 - 03/10/2024

[...]

Complete mediation

[...]

Open design

- do not rely on secret designs, attacker ignorance or security by obscurity. Invite and encourage open review and analysis.
- For example, the Advanced Encryption Standard was selected from a set of public canditates by open review; undisclosed cryptographic algorithms are now widely discouraged.

Least privilege

- allocate the fewest privileges needed for a task, and for the shortest duration encessary
- for example, retain superuser privileges (chapter 5) only for actions requiring them; drop and reacquire privileges if needed later
- do not use a unix root account for tasks where regular user privileges suffice. This reduces exposure.

Modular design

- avoid monolithic designs that embed full priileges into large single components; favor object-oriented and finder-grained designs that segrefate privileges (including addres spaces) across smaller units or processes.
- NOTE: a related financial accounting principle is separation of duties, whereby codependent tasks are assigned to independed parties so that an insider attack requires collusion

Psychological acceptability

- design mechanisms, and their user interfaces, to behave as users expect
- [...]

Logical security: the basic security strategy

Gold standard rules (Au):

- authenticating principals
- authorizing access
- auditing the decisions of the guard

[...]

Reference monitor

In the model, a system first indentifies all subjects and objects. For each object, the types of access (access attributes) are determined, each corresponding to an access permission or privilege. Then for each subject-object pair, the system predefines the authorized acces permissions of that subject to that object.

[...] last part of the lecture is missing

L05 - 10/10/2024

Designing Secure Systems (II)

Terminology

- subjects: active entities that perform actions
- objects: are the entities on which the subjects perform those actions

Access control matrix

Buona in via teorica, non implementabile in pratica. Si utilizzano due metodi per rappresentarla:

1. Capability lists (C-lists)
2. ACL (access control list) and ACE (access control entry)

Issues:

- How to represent subjects in a computer
- How to represent objects in a computer
- ...

Subjects: users

All processes that will be generated by the user, from the moment he is recognized by the system as valid, will inherit his UID.

Superuser

The superuser is a special priviliged principal with UID 0 and usually username `root`

- all security checks are turned off for superuser
- the superuser can become any other user
- the superuser can change the system clock

Superuser cannot write to a read-only file system but can remount it as writable, and it also cannot decrypt passwords but can reset them.

How is a UID assigned to a process

- users launch programs/commands which are executed by the operating system
- any program give rise to one or more processes
- a process consists of a program (sequence of instructions)
- ....

Creating a process

Process control block

- the context of a process is stored in a special kernel data structure known as a PCB
- address space of a process in the central memory (.text, .data, heap and stack)

Process creation via syscall

- the system call `fork()` creates a copy of the calling process. `fork()` returns:
  - -1 on error
  - 0 to the child process
  - Child's PID to the parent

```c
#include <stdio.h>

void main() {
    int i;

    for (i = 0; i < 3; i++) {
        fork();
```

```
8
9           // this printf statement is for debugging purposes
10          // getppid(): gets the parent process-id
11          // getpid(): gets child process-id
12          printf("[%d] [%d] i=%d\n", getppid(), getpid(), i)
13      }
14
15      printf("[%d] [%d] hi\n", getppid(), getpid())
16  }
```

Shell process

∗ ⟶ Read command ⟶ interpret command ⟶ execute the command ⟶ display prompt ⟶ ∗

Shell pseudocode

Objects in Unix

All computers ressources: files, directories, memory devices, I/O are treated in UNIX as files

- this way the general access control in UNIX is reduced to the file system access control
- they are the objects of access control
- resources organized in a tree-structured file system
- each file entry in a directory is a pointer to a data structure called inode

i-node:

- user and group owner IDs

Windows objects

[...]

Ugo

The ugo permission model assigns privileges based on three categories of principals: user, group, others.

- the user category refers to the principal that is the file owner
- the group category enables sharing of resources among small to medium sized sets of users (like project groups) with relatively simple permissions management.

This provides a compact and efficient way to handle an object for which many users should be given the same privileges. This ugo model allows fixed-size filesystem emta-data entries, and saves storage and processing time.

In unix we can represent an access control list with just three 3 bits groups: drwxrwxrwx (octal numbers form).

Permissions: order of checking

access control uses the effective UID/GID:

- if the subject's UID owns the file, the permission bits for owner decide whether access is granted
- if the subject's UID does not own the file byt its GID, does, the permission bits for group decide whether access is granted
- if the subject's UID and GID do not own the file, the permission bits for other (also called world) decide whether access is granted.

Privilege escalation

a feature which allows authorized users to temporarily elevate their privileges to perform specific tasks that require highr permissions.

- f.e. a system administrator might need to escalate their privileges to install updates or troubleshoot issues

there is a key distinction between controlled and authorized privilege escalation versus unauthorized and malicious privilege escalation.

In unix like systems, authorized privilege escalation is done via SET-UID

Set-UID concept

allow user to run a program with the program owner's privilege

- allow users to run programs with temporary elevated privileges

example: the `passwd` program (ls -l /usr/bin/passwd)

- it only lets you to access your own username's password

Set-UID implementation

every process has 3 User IDs:

1. Real UID (RUID)
2. Effective UID (EUID)
3. Saved UID (SUID)

when a normal program is executed, RUID = EUID, they both are equal to the ID of the user whor uns the program

when a Set-UID program is executed, RUID != EUID. RUID is still equal to the user's ID, but EUID equals to the program owner's ID

Example of turning a program into Set-UID

Chmod to change the access control list

L06 - 14/10/2024

Exploits and privilege escalation (cap 6 CSaTIT)

TOCTOU Race attack (cap 6.1)

Race conditions

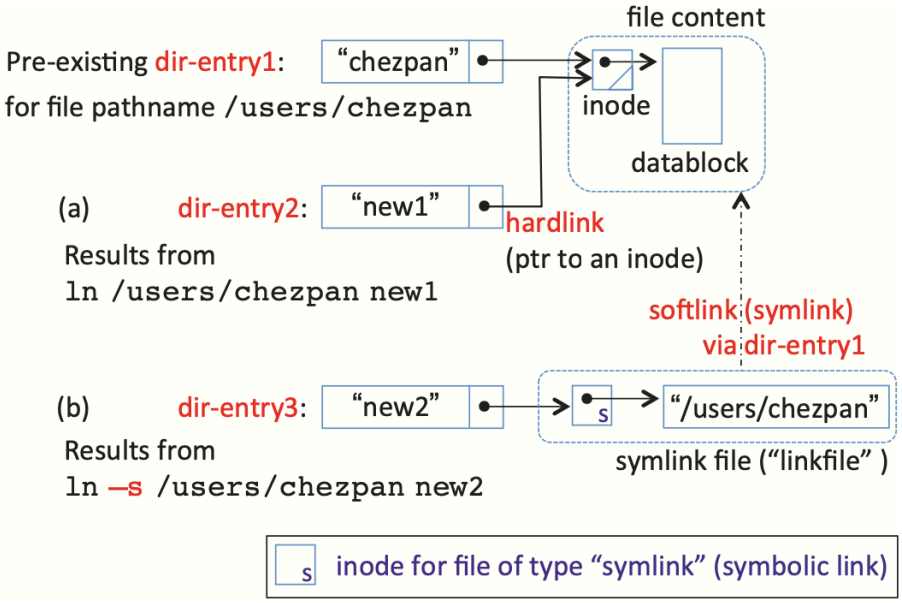
Race conditions in software occur when two or more concurrent threads happen at the same time on the same resouce, and they are managed with some locks.

Preliminary (cap 5.6)

In Unix, the same non directory file can appear in multiple directories, optionally with different names. This is done by linking using the `ln` command.

- A link can be either a **symlink** (soft link or indirect alias) or a **hard link** (direct alias)
- Hard links are done by using inodes tables and filesystem file tables. They are heavily restricted to the structure of the filesystem on the specific OS (they are not *transferable*).
- Soft links have a different effect: they generate a new inode which points to a file, which content is the path of the first file (the file that is being linked). Those links can be shared across different filesystems.

Considering a file with name `existing`, then the command `ln existing new1` creates a logcal copy of existing whose name is `new1`.



Privilege escalation via TOCTOU race

- the `access()` system call invoked by a process P checks wheter the real user ID or group ID of P has permissions to access a resource, and returns 0 if it does.
  - this system call is usually used by set-UID program [...]
- the `open()` system call also conducts access control, it checks wheter the effective user ID or group ID has permissions to access a file.

Race conditions: the vuln program

The following code snippet has been found inside a root owned setuid program; thus, it is executed with RUID = *some user UID*, and EUID = 0

```
if (!access("/tmp/XYZ", W_OK)) {
    // the real user ID does have access right
    f = open("/tmp/XYZ", O_WRITE);
    write_to_file(f);
} else{
    // the real user ID does not have access right
    fprintf(stderr, "Permission denied\n")
}
```

We want to use such a program for modifying the `/etc/passwd` file, eve if:

- we do not have root privileges
- the program is written for accessing the file `/tmp/XYZ`

But we know about the existence of the `/dev/null` file

The attack

```
#include <unistd.h>

int main() {
    while(1) {
        unlink("/tmp/XYZ");
        symlink("/dev/null", "/tmp/XYZ");
        unsleep(1000);

        unlink("/tmp/XYZ");
        symlink("/etc/passwd", "/tmp/XYZ");
        usleep(1000);
    }
```

```
13
14      return 0;
15  }
```

[... gpt notes

1. **Unlink the file** /tmp/XYZ**:**
   The first unlink() call removes the existing file /tmp/XYZ. This is necessary so that we can create a symbolic link in its place.
2. **Create a symbolic link to** /dev/null**:**
   The symlink() call creates a symbolic link from /tmp/XYZ to /dev/null. At this moment, if the vulnerable program checks the file /tmp/XYZ, it will think that the user has access to it because /dev/null is universally writable.
3. **Sleep for a short time:**
   The usleep(1000) call adds a small delay, hoping that the vulnerable program will perform the access() check during this time. This is key to exploiting the race condition.
4. **Unlink** /tmp/XYZ **again:**
   The next unlink() removes the symbolic link to /dev/null, freeing up the name /tmp/XYZ.
5. **Create a new symbolic link to** /etc/passwd**:**
   The symlink("/etc/passwd", "/tmp/XYZ") creates a new symbolic link from /tmp/XYZ to /etc/passwd. Now, if the vulnerable program calls open("/tmp/XYZ"), it will actually open /etc/passwd instead of /tmp/XYZ.
6. **Sleep again:**
   Another small delay is introduced to increase the chances that the program will execute the open() system call after the symbolic link has been changed.

The Exploit Outcome

- During the race, if the vulnerable program checks the access rights on /tmp/XYZ while it is linked to /dev/null, the check will pass because /dev/null is writable by everyone.
- Then, after the access check but before the file is opened, the attacker switches the symbolic link to point to /etc/passwd.
- When the vulnerable program calls open("/tmp/XYZ"), it will end up opening /etc/passwd instead, with root privileges (EUID = 0), allowing the attacker to write to the system password file.

Summary

- *Race condition:** The attacker exploits the time between the access() and open() system calls.
- **Symbolic links:** The attacker uses symbolic links to manipulate the file that the program opens after the access check.
- **Root privileges:** The program runs with EUID = 0 (root), allowing the attacker to perform privileged operations (like modifying /etc/passwd).

This type of race condition is a common vulnerability in programs that check access rights before opening files, and it demonstrates why TOCTOU (Time-of-Check-to-Time-of-Use) issues are dangerous.

...]

We execute concurrently the vuln program and our attack program, if we are lucky enough to get the following interleaving of instructions we succeed:

```
1  unlink("/tmp/XYZ");
2  symlink("/dev/null", "/tmp/XYZ");
3  ----------------------------------------------
4  1: if (!access("/tmp/XYZ", W_OK)) {
5  2: // the real user ID does have access right
6  ----------------------------------------------
7  unlink("/tmp/XYZ");
8  symlink("/etc/passwd", "/tmp/XYZ");
9  ----------------------------------------------
10 3: f = open("/tmp/XYZ", O_WRITE);
11 4: write_to_file(f);
```

Usually there is a short interval window between `access()` and `open()`, that is the windows between the checking and the using (time-of-check time-of-use: TOCTOU)

Buffer overflow attack (cap 6.3)

buffer overflow attack via smashing the stack technique

Overwriting in the writing process space with arbitrary data

The holy grail of exploitation is to take control of the instruction pointer of a process by exploiting security bug contained in an executable

- In a two-stage process, first a safed instruction pointer is overwritten and then the program executes a legitimate instruction that transfers control to the attacker-supplied address.
- The smashing the stack technique can be used for obtaining such a result.

Memory layout of a Unix process

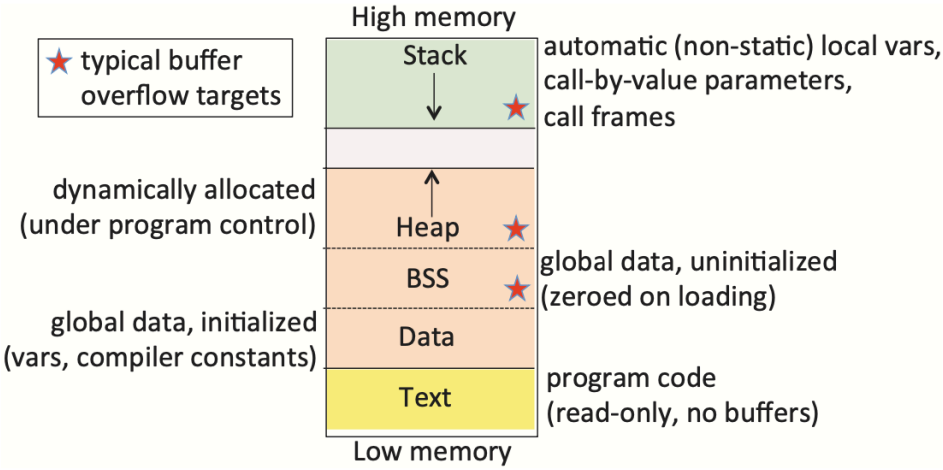


Figure 6.3: Common memory layout (user-space processes).

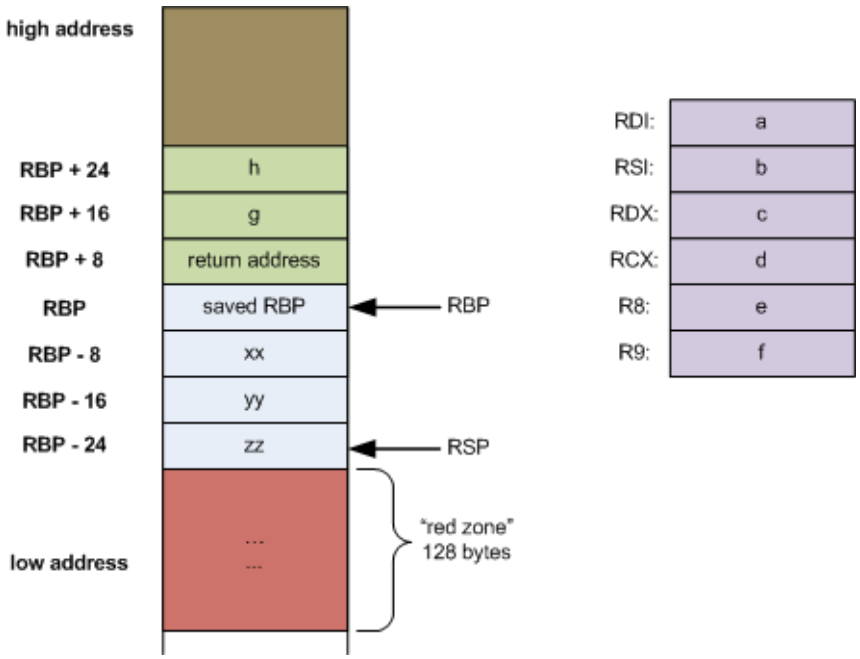
Stack

- A stack is a LIFO memory zone, all CPUs have internal instructions to access this memory zone (PUSH & POP)
- Any process has its own stack which is generally managed by the compiler which uses it as a transient memory
- The PUSH statement adds data to the stack and the POP removes it

Stack frame

Whenever there is a function call in our program a stack frame is allocated which contains all the information required for managing the function execution

- The memory allocated for a function call in the stack lives only while the function is executing
- Each stack frame maintains the Stack Pointer (RSP), and **[... RBP]**



Information stored in a stack frame

1. Calling and returning

Two instructions that use the stack are used to make calling subprograms quick and easy:

- The `CALL` instruction makes an unconditional jump to a subprogram and **pushes** the address of the next instruction on the stack.
- The `RET` instruction **pops off** an address and jumps to that address

Push

The push instruction inserts a double word on the stack by subtracting 4 from ESP and then stores the double word at [ESP]

```
pushl src   ->  subl $4, %esp
                movl src, (%esp)
```

The stack pointer containing the address is decremented and the data is loaded in there

Pop

```
popl dest    ->     movl (%esp), dest
                    addl $4, %esp
```

Il PC è un registro che contiene l'indirizzo di istruzioni di codice, e lo stack ne contiene delle copie.

Exercises about the stack

Exercise 1

```c
#include <stdio.h>

int main() {
    int cookie;
    char buf[80];

    printf("buf: %08x cookie: %08x\n", &buf, &cookie);
    gets(buf);

    if (cookie == 0x41424344)
        printf("you win!/n");
}
```

L'input per ottenere *"you win!"* è di 80 caratteri spazzatura con alla fine "DCBA"

- 0x41424344 sarebbe ABCD codificato in ASCII, ma essendo l'architettura Intel in Little Endian bisogna invertire la stringa.

Questa è la prima forma di buffer overflow: il nostro buffer è di 80 ma noi ne abbiamo scritte 84.

[guardare la foto 1 sul telefono per la struttura dello stack]

How to modify a program control flow

```c
void function() {
    char buffer1[4];
    int *ret;
    ret = buffer1 + 8;
    (*ret) += 8;
}

void main() {
    int x = 0;
    function();
    x = 1;
    printf("%d\n", x);
}
```

All'interno di un programma C è possibile quindi modificare il return address (si possono quindi governare degli instruction pointers)

[guardare la foto 2 sul telefono per la struttura dello stack]

Smashing the stack

Buffer overflow

```c
void function(char *str) {
    char buffer[8];
    strcpy(buffer, str);
}

void main() {
    char large_string[256];
    int i;
    for (i = 0; i < 255; i++)
        large_string[i] = 'A';
    function(large_String);
}
```

Questo programma restituisce un segmentation fault poiché `strcpy` inizia a copiare i 256 bit di str (large_string) in buffer, che però occupa nello stack solo 8 bit. strcpy però copia finché non arriva al carattere di terminazione di str: salendo sullo stack, inizia a sovrascrivere tutto: return address, la variabile i e forse anche large_string stesso. Viene eseguita la RET, ma essendo stato il return address nello stack sovrascritto da "AAAA" (414141) il processo viene bloccato -> segmentation fault.

- questo è interessante per noi perché se la stringa non fosse predefinita dal programmatore (in questo caso AAAA....) ma fosse data in input dall'utente, noi possiamo arrivare idealmente al return address.
- possiamo sfruttare ciò per sovrascrivere il return address con l'indirizzo di un nostro programma, però come facciamo con l'input a far entrare del codice malevolo in un'altra macchina?
    - bisogna inserire il codice dell'attacco e l'indirizzo di questo codice

[guardare la foto 3 sul telefono per la struttura dello stack]

Consequences of buffer overflow

[...]

```c
#include <stdio.h>
#include <stdlib.h>

void main () {
    char *name[2];
    name[0] =
}
```

[foto telefono]

Il programma eseguito viene sostituito da una shell

Il programma in C viene tradotto in shellcode, ossia in linguaggio macchina che può essere direttamente eseguito dalla macchina.

Noi quindi riusciamo a introdurre il codice malevolo sullo stack, ora però bisogna dire al program counter l'indirizzo dove si è messo il codice.

- Si mettono i Nop per aumentare le probabilità di indovinare l'indirizzo del codice maligno
- questi tipi di attacchi sono efficaci su linguaggi come il C e i suoi derivati, che sono flessibili sulla gestione dei dati e non è tipato (non controlla i tipi dei dati coinvolti nelle istruzioni)
    - (la vulnerabilità deriva quindi dal linguaggio di programmazione)

The exploit is the string of bytes which given as input to a vulnerable program enables the execution of malicious code, usually inserted in the payload of the exploit.

| Nop | Shellcode | RA | RA | RA |
^
start of buffer

Si prendono dei codici scritti in C che chiedono dei dati in input. Si mette in input una stringa enorme, e se si ottiene uno segmentatio fault allora la stringa è finita sullo stack e non ci sono controlli.

L09 - 24/10/2024

Integer overflow [pag 179 pdf CSatI]

Since an integer is a fixed size variable, usually 32 bits, there is a fixed maximum value it can store. When an attempt is made to store a value greater than this maximum value, it's being done an integer overflow.

Integer overflows cannot be detected after they've happened, so there is no way for an application to tell if a result it has calculated previously is in fact correct. This can get dangerous if the calculation has to do with the size of a buffer.

The ISO C99 says that a computation involving **unsigned operands** can never overflow.

Integer representation

Negative numbers have the MSD equal to 1, on the other hand positive numbers have a 0 on the MSD (rappresentati mediante **complemento a 2**)

So numbers are represented through the use of a modular arithmetic (check computer architecture I)

Unsigned 32bit:
0000 0000 0000 0000 0000 0000 0000 0000 lowest number 0
1111 1111 1111 1111 1111 1111 1111 1111 highest number $2^{32} - 1$

Signed 32bit
0000 0000 0000 0000 0000 0000 0000 0000 lowest positive number 0
0111 1111 1111 1111 1111 1111 1111 1111 highest positive number $2^{31} - 1$
1111 1111 1111 1111 1111 1111 1111 1111 highest negative number -1
1000 1000 1000 1000 1000 1000 1000 1000 lowest negative number -2^{31}

Example

system-call handler may check string lenghts to defend against buffer overruns

```
char buf[128];

combine(char *s1, size_t len1, char *s2, size_t len2) {
    if (len1 + len2 + 1 <= sizeof(buf)) {
        strncpy(buf, s1, len1);
        strncat(buf, s2, len2);
    }
}
```

Consider the following scenario:

- size_t is a typedef of `uint16`: max number is 65.535
- len1 = 65550
- len2 = 100

In the above example `len1 + len2` = 64 < 128 (due to overflow), thus in `strncpy` there is room for a BOVF (buffer overflow)

Checking for integer overflow / underflow

Unsigned overflow checks need to use the complementary operation to the one being checked.

- subtraction to check for addition ovf `if (UINT32_MAX - a < b)` (so `UINT32_MAX - a` has still room for `b`)
- division to check for multiplication ovf `if ((a != 0) && (UINT32_MAX / a >= b))`

Signedness bug

```
int copy_something(char *buf, int len) {
    char kbuf[800];
    if (len > sizeof(kbuf)) {           // [1]
        return -1
    }
    return memcpy(kbuf, buf, len);      // [2]
}
```

Set `len = -16`, since it is a signed integer its binary representation is 1111 1111 1111 0000, which becomes 65520 if it is considered an unsigned int.

- That's because `memcpy` only accepts unsigned integers, so `len` is converted from a signed integer to, inside memcpy, an unsigned integer (implicit cast).

Integer type conversion

Integer type conversions are yet another common source of security vulnerabilities

Whenever a value is changed from one type to another, they can be truncated, zero extended or one extended (whether you have positive or negative numbers)

Truncation in C occurs when a value is converted to a data type with a smaller range, causing the ecess bits to be discarted. [...]

```
struct s {
    unsigned short len;
    char buf[];
};

void foo(struct s *p) {
    char buffer[100];
    if (p -> len < sizeof buffer)
        strcpy(buffer, p -> buf);
    // Use buffer
}

int main(int argc, char *argv[]) {
    size t len = strlen(argv[0]);
    struct s *p = malloc(len + 3);
    p -> len = len; // qui avviene un troncamento
    strcpy(p -> buf, argv[0]);

    foo(p);
    return 0;
}
```

- A single line of code, due to a integer overflow in conversion of 64 bit float to 16 bit int, brought down a half-bilion euro rocket launch: the first Ariane V launch.

Memory error exploits: countermeasures [pag 190-191 pdf CSatI]

3 momenti nella cybersecurity:

- **prevention**: prevenire, ossia creare le condizioni affinchè un certo tipo di attacco si possa verificare.
- **detection**: rilevare, ci si rende conto che non si può prevenire un certo tipo di attacco, ma si fa in modo di assorbirlo tramite certe contromisure appena questo accade
- **response**: fallite la prevention e la detection, si hanno i sistemi compromessi e bisogna fare in modo di ripristinarli e ripartire il prima possibile.

Developer approaches (prevention): use of safer functions like strncpy(), strncat(), etc, and safer link libraries that check the length of the data before copying
Hardware approaches (prevention): non-executable stack
Compiler approaches (detection): stack-guard
Os Approaches: ...

Prevention

- Don't use C or C++ (use type-safe language)
- Better programmer awareness & training
- Use safer libraries

1. Non-Executable Stack (NX Stack)

Usually we store in the stack data, so there is no need to leave the stack executable.

- If someone does shellcode injection in the stack, it won't be executed anymore as the stack is now a portion of memory that cannot be executed.

Code injected onto the stack will not run anymore (so smash in the stack is no longer usable) (it's enabled by default in most linux distributions, mac os and windows).

- This can be bypassed by executing the code elsewhere and not anymore on the stack, as the return address can still be over-written:
  - **ret2libc attack** (la macchina viene attaccata con i suoi stessi programmi: bisogna trovare però in libreria le funzioni che mi servono)
  - **ret2strcpy attack**: based on ret2libc, you place the shellcode on the stack and then use strcpy() to copy NOP sled + shellcode to a writable and executable memory address.
  - **ret2gets attack**: needs only one argument: a writable and executable memory address. Reads NOP sled and shellcode from stdin. It's often controlled by attacker.

So the NX Stack still allows return address to be abused to divert execution flow, and does not prevent the execution of: code already present in a process' memory, and of code in other data areas.

NX stack is done by the hardware

2. W^X (writable or executable memory)

Memory is either writable but not executable or non-writable and executable. So injected code won't run, no matter where it goes.

In the Windows world is called Data Execution Prevention, and it was available starting with Win XP SP2

Data Execution Prevention

DEP prevents code from being run from data pages such as the default heap, stacks and memory pools.
Some system-calls, if executed from a root user, can disable DEP on certain memory pages (or also all memory)

- It's still possible do to ret2strcpy and ret2gets (*return-to-libc* exploits), even though it's toughier

3. ASLR (Address space layout randomization)

return-to-libc exploits require address of malicious code in memory and addresses of routines to be called.

To withstand against *return-to-libc* attacks, the idea is to introduce artificial diversity: do not load the code at a fixed address.

Attackers use absolute memory addresses during the attacks, so ASLR nullifies the attacker's assumption by making the memory locations of program objects unpredictable and by forcing attackers to guess memory location with low probability of succcess

Benefit: protection against known and unknown memory corruption attakcs

- Loading code in memory at different addresses so that it is harder to locate it
- Works best if all code is compiled with the Position Indipendent Execution (PIE) flag
  - apps and libraries
  - many third party apps are not compiled for PIE

ASLR enables randomisation of binary executable, heap, stack, libraries and dynamic linker (true only for strong ASLR with PIE)

- without PIE the ASLR randomises only stack and heap, so it is still possible to do *return-to-libc* attacks

ASLR is done by the operative system.

4. Stack guard

Stack guard is done by the compiler

```
void foo (char *str) {
    int guard; // also know as Canary
    guard = secret;

    char buffer[12];
    strcpy(buffer, str);

    if (guard == secret)
        return;
    else // se la guardia è stata sovrascritta (sta sopra il return address) allora c'è stato un overflow
        exit(1);
}
```

Oggi giorno la guardia, che al tempo era un numero randomico, è stata sostituita da 0x000A (00 è il terminatore di stringa, così come 0A)

- Questo perchè se si aggiungono 0x000A alla nostra shellcode injection, leggendo il terminatore di stringa la copia delle stringhe finisce.

Execution with StackGuard
[...]

Sono 6 istruzioni canarino (3 di canary set e 3 di canary check), e vengono aggiunte a ogni chiamata di procedura.

Questa tecnica rende il buffer overflow estremamente più difficile da fare

5. Static Analysis (source code analysis)

Automated analysis at compile time to find pontential bugs.

1. simple syntactic checks for unsafe instructions
2. type checking
3. more advanced analyses taking semantics into account

False positives and false negatives: false positives are worse, as they kill usability

Those systems warn about unused variables, about dead/unreachable code, about missing initialisation.

Static because analyses the code (on the other hand the dynamic analyses check the execution of the code)

Limits: some variables are initialised at run-time.

L11 - 31/10/2024

Cryptography (cap 2 CSatI)

Cryptography is fundamental to hide data from unauthorized access, and it's an important mean for providing data confidentiality, especially in distributed communications systems.

Encryption transforms data (**plaintext**) into an unintelligible form (**ciphertext**). The process is reversible: a decryption key allows recovery of plaintext, using a corresponding decryption algorithm.

- Access to the decryption key controls access to the plaintext; thus (only) authorized parties are given access to this key. It is generally assumed that the algorithms are known, but that only authorized parties have the secret key.

Sensitive information should be encrypted before submission to avoid any kind of intruded to be able to eavesdrop and understand the information: if this is encrypted, he'll just have a bunch of randomic and meaningless text.

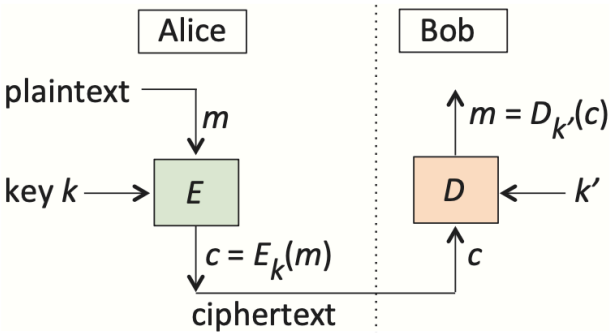


Figure 2.1: Generic encryption (E) and decryption (D). For symmetric encryption, E and D use the same shared (symmetric) key $k = k'$, and are thus inverses under that parameter; one is sometimes called the "forward" algorithm, the other the "inverse". The original Internet threat model (Chapter 1) and conventional cryptographic model assume that an adversary has no access to endpoints. This is false if malware infects user machines.

Let m denote a plaintext message, c the ciphertext and $E_k, D_{k'}$ the encryption, decryption algorithms parameterized by simmetric keys k, k' respectively. We describe the encryption and decryption with equations:

$$c = E_k(m); \quad m = D_{k'}(c)$$

Mathematically, we can describe and encryption-decryption system (**cryptosystem**) to consist of:

- a set $\mathcal{P}$ of possible plaintexts
- set $\mathcal{C}$ of possible ciphertexts
- set $\mathcal{K}$ of keys
- an encryption injective function $E : (\mathcal{P} \times \mathcal{K}) \to \mathcal{C}$
- a corresponding decryption function $D : (\mathcal{C} \times \mathcal{K}) \to \mathcal{P}$

Exhaustive key search

We rely on the fact that the algorithms E and D are good and have no algorithmic weakness. This way, it should be infeasible to recover m from c without knowing the key k'. So an intruder can only, after intercepting a ciphertext c, go trhough all keys k from the whole key space $\mathcal{K}$, parameterizing D with each k sequentially, computing each $D_k(c)$ and looking for some meaningful result $\to$ *this process is called* ***exhaustive key search***

Cipher attack models

- **cyphertext-only**: an attacker tries to recover plaintext or the key given access to the ciphertext alone.
- **know-plaintext**: given access to some ciphertext and its corresponding plaintext, attackerstry to recover the unknown plaintext (or the key) from further ciphertext.
- **chosen-plaintext**: attackers can choose some amount of plaintext and see the resulting ciphertext. This kind of control may allow advanced analysis that defeats weaker algorithms
- **chosen-ciphertext**: for a fixed key, attackers can provide ciphertext of their choosing, and receive back the corresponding plaintext

Those methos are very important to test the security and the robustness of cryptographic systems: an ideal encryption algorithm resists all these attack models, ruling out algorithmic “shortcuts”, leaving only exhaustive search.

Cryptographic protocols

There are two categories of algorithms:

- **symmetric key** (secret key): the encryption and decryption keys are the same ($k = k'$)
  - *that means that the key is the same for both the sender and receiver*
- **asymmetric key** (public key)

Types of symmetric ciphers

Stream ciphers generate a keystream simply XOR'd onto plaintext bits, decryption involves XORing the ciphertext with the same keystream. They encrypt the plaintext one bit or one character at a time.

Block ciphers process the plaintext in fixed-length chunks or blocks. Each block is encrypted with a fixed transformation dependent on the key.

Vernam cipher

The vermam cipher is an example of a stram cipher. It needs a key as long as the plaintext, and the algorithm is a bitwise XOR between plaintex and key:

- to encrypt a t-bit message $m_1 m_2 \cdots m_t$ using key $k = k_1 k_2 \cdots k_t$ we have $c_i = m_i \oplus k_i$ where $c = c_1 c_2 \cdots c_t$ is the ciphertext

If the key k is randomly chosen and never re-used, the Vernam stream cipher is called a **one-time pad**. One-time pads are know to provide a theoretically unbreakable encryption system.

[...] vedi appunti di critto

DES

Vedi appunti di critto -> quaderno

L12 - 04/10/2024

Public Key Crypto and PKI

Digital signatures (cap 2.4 CSatI)

Digital signatures, typically computed using public-key algorithms, are tags (bitstrings) that accompany messages. Each tag is a mathematical function of a message (its exact bitstring) and unique-per-sender private key.

A corresponding public key, uniquely associated with the sender, allows aytomated verification that the message originated from that individual, since only that individual knows the private key needed to create the tag.

Digital signature properties

- **Data origin authentication**: assurance of who originated (signed) a message or file
- **Data integrity**: assurance that received content is the same as that originally signed
- **Non-repudiation**: strong evidence of unique origination, making it hard for a party to digitally sign data and later successfully deny having done so. This follows from signature verification not requiring the signer's private key, verifiers use the signer's public key.

Digital signature (done with the right instruments and using the right certificators) has the same legal validity as handwritten signature

Public-key methods can be used to implement digital signatures by a process similar to encryption-decryption, but with subtle differences.

In place of encryption public keys, decryption private keys, and algorithms E, D (encrypt, decrypt), we know have signing private keys for signature generation, verification public keys to validate signatures, and algorithms S, V

Public ket cryptography protocols

The most commonly used public key cryptographic protocols are:

- RSA (based on the difficulty of factorization problem)
- El Gamal (based on the difficulty of the discrete logarithm problem)
- DSS (boh cerca su internet)
- ECC (crittografia con curve ellittiche)

The first two use two open problems from number theory (they are not computationally solvable at present)

Cryptographic hash functions (cap 2.5 CSatI)

Cryptographic hash functions help solve many problems in security. They take as input any binary string and produce a fixed length output called a **hash value**, hash, message digest or digital fingerprint.

They typically mapo longer into shorter strings, as do other (non-crypto) hash functions in Computer Science, but have special properties

For a good hash function, changing a single binary digit (bit) of input results in entirely unpredictable output changes (50% of output bits change on avarage)

- (per renderlo indistinguibile dal rumore, ossia stringhe casuali)

Properties

Hash functions, given any input, have to have a relatively small computational cost. Three hash function security properties are often needed in practice:

1. **one-way property** (or preimage resistance): for essentially all possible hash values h, given h it should be infeasible to find any m such that $H(m) = h$
2. **second-preimage resistance**: given any first input $m1$, it should be infeasible to find any distinct second input $m2$ such that $H(m1) = H(m2)$.
    - *Note: there is free choice pf m2 but m1 is fixed. H(m1) is the target image to match, m1 is its preimage*

3. **collision resistance**: it should be infeasible to find any pair of distinct inputs $m1, m2$ such that $H(m1) = H(m2)$
    - *Note: here there is free choice of both m1 and m2. When two distinct inputs hash to the same output value, we call it a collision*

The **first property** implies that given a hash value, an input that produces that hash cannot be easily found, even though many, many inputs do indeep map to each output.

- To see this, restrict your attention to only those inputs of exactly 512 bits, and suppose that the hash function output has bitlength 128. Then H maps each of these 2^{512} input strings to one of 2^{128} possible outputs strings, so on avarage $\frac{2^{512}}{2^{128}} = 2^{384}$ inputs produce the same 128-bit output.
    - *Every output value is produced by 2^{384} different input values*

The term infeasible used in the three properties means computationally infeasible in practice

Hash and password (H1 & H2)

One-way hash functions H are often used in password authentication as follow:

- a userid and password p entered on a client device are sent (hopefully over an encrypted link) to a server
- the server hashed the p received to $H(p)$ and uses the userid to index a data record containing the (known-correct) password hash. If the values match, login succeeds/
- this avoids storing, the server, plaintext passwords, which might be directly available to disgruntled administrators, anyone with access to backup storage, or via server database breakins

Data integrity

Consider an executable file corresponding to program P with binary representation p, faithfully representing legirima source code at the time P is installed in the filesystem

- at that time, using a hash function H with properties H1-H3, the operating system computes $h = H(p)$. This *trusted-good* hash of the program is stored in memory that is safe from manipulation by attackers.
- later, before invoking program P, the operating system recomputes the hash of the executable file to be run,, and compares the result to stored value h. If the values match, there is strong evidence that the file has not been manipulated or substituted by an attack.

Hash and digital signature (pag 44-45)

Most digital signature schemes are implemented using mathematical primitives that operate on fixed-size input blocks. Breaking a message into blocks of this size, and signing individual pieces, is inefficient

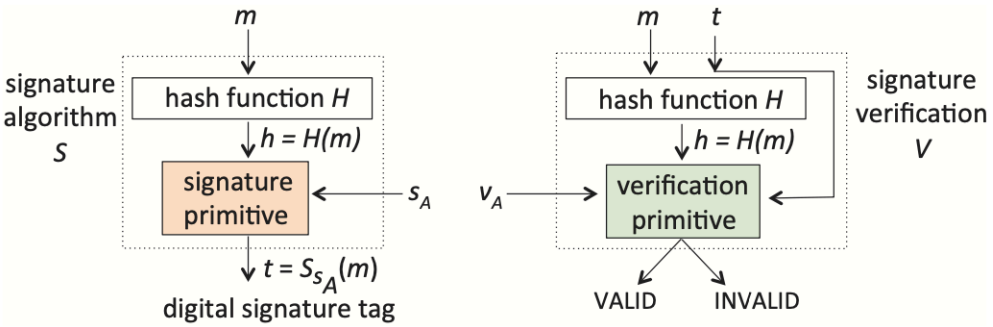


Figure 2.11: Signature algorithm with hashing details. The process first hashes message m to $H(m)$, and then applies the core signing algorithm to the fixed-length hash, not m itself. Signature verification requires the entire message m as input, likewise hashes it to $H(m)$, and then checks whether an alleged signature tag t for that m is VALID or INVALID (e.g., returning boolean values TRUE or FALSE). s_A and v_A are Alice's signature private key and signature verification public key, respectively. Compare to Figure 2.9.

MAC (message authentication) (cap 2.6)

Message authentication is the service of assuring the integrity of data (i.e., that it has not been altered) and the identity of the party that originated the data, i.e., data origin authentication.

This is done by sending a special data value, or tag, called a message authentication code (MAC), along with a message. The algorithm computing the tag, i.e., the MAC function, is a special type of hash function whose output depends not only on the input message but also on a secret number (secret key).

- È una funzione hash che non dipende solo dall'input ma anche da una chiave segreta condivisa tra mittente e destinatario.
- Il fatto che la chiave sia condivisa tra due fa sì che il MAC non soddisfi la non repudiabilità (uno potrebbe usare la chiave per mandarsi un messaggio falso da solo)

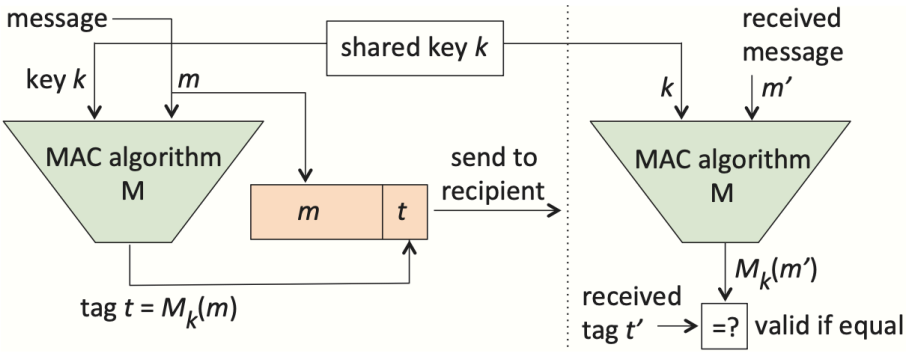


Figure 2.12: Message authentication code (MAC) generation and verification. As opposed to unkeyed hash functions, MAC algorithms take as input a secret (symmetric key) k, as well as an arbitrary-length message m. By design, with high probability, an adversary not knowing k will be unable to forge a correct tag t for a given message m; and be unable to generate any new pair (m, t) of matching message and tag (for an unknown k in use).

If Alice sends a message and matching MAC tag to Bob (with whom she shares the MAC key), then he can verify the MAC tag to confirm integrity and data origin. Since the key is shared, the tag could also have been created by Bob. Between Alice and Bob, they know who originated the message, but if Alice denies being the originator, a third party may be unable to sort out the truth. Thus, MACs lack the property of non-repudiation, i.e., they do not produce evidence countering repudiation (false denial of previous actions). Public-key signatures provide both data origin authentication and non-repudiation.

EC (pag 50)

Public-key systems are most easily taught using implementations over number systems that students are already somewhat familiar with, e.g., arithmetic modulo n = pq (for RSA), and modulo a large prime p for Diffie-Hellman (DH) later in Chapter 4. By their underlying mathematical structures, RSA and DH are respectively classified as integer factorization cryptography (IFC) and finite field cryptography (FFC). Public-key functionality—encryption, digital signatures, and key agreement—can analogously be implemented using operations over sets of elements defined by points on an elliptic curve. Such elliptic curve cryptography (ECC) implementations offer as a main advantage computational and storage efficiencies due to smaller key sizes

Certification Authorities (cap 8)

Fundamental problem: when someone (person 1) receives a public key from someone else (person 2), person 1 has no way to tell whether the public key he has received effectively belongs to person 2 or not. How can he trust the information received?

Solution: person 2 has to present his public key PUK together with a declaration which states that PUK is precisely his public key. To be accepted by person 1 such a declaration should be signed either:

- by a well known authorithy
- one or more persons who person 1 trusts

The choice on the most appropriate tool for attesting public key identity gave rise to two approaches solving the trust problem in communication:

- PKI and X.509 (fa riferimento a entità centralizzate radicate che rilasciano certificati)

- Pretty good privacy - GPG (struttura decentralizzata in cui è la rete che si autosostiene e attraverso essa la gente guadagna o perde fiducia)

X509 and PKI

- Find a trusted party to verify the identity
- Bind an identity to a public key in a certificate i.e. a document which establish the correspondence between a public key and its owner
- [...]

Certification Authority (cap 8.1 pag 214-215)

Certification Authority (CA): a trusted party, responsible for verifying the identity of users, and then bind them to a public key.

- The CA's role is critical for trustworthy certificates. Before signing a certificate, the CA is expected to carry out appropriate due diligence to confirm the identity of the named subject, and their association with the public key
- For example, to obtain evidence of control of the corresponding private key, the CA may send the subjcet a challange message whose correct response requires use of that private key (without disclosing it); the CA uses the purportedly corresponding public key in creating the challenge, or verifying the response.

Public-key certificate

A **public-key certificate** is used to associate a public key with an owner (i.e., the entity having the matching private key, and ideally the only such entity). The certificate is a data structure that binds a public key to a named Subject, by means of a digital signature generated by a trusted third party called a Certification Authority (CA).

Any party that relies on the certificate — i.e., any relying party — places their trust in the issuing CA, and requires the corresponding valid public key of that CA in order to verify this signature. Verifying the correctness of this signature is one of several steps that the relying party's system must carry out as part of checking the overall validity of the target public-key certificate.

- **Digital certificates allow relying parties to gain trust in the public keys of many other parties, through pre-existing trust in the public key of a signing CA. Trust in one key thus translates into trust in many.**

Field name	Contents or description
`Version`	X.509v3 or other versions
`Serial-Number`	uniquely identifies certificate, e.g., for revocation
`Issuer`	issuing CA's name
`Validity-Period`	specifies dates (`Not-Before`, `Not-After`)
`Subject`	owner's name
`Public-Key` info	specifies (`Public-Key-Algorithm`, `Key-Value`)
extension fields (optional)	`Subject-Alternate-Name/SAN-list`, `Basic-Constraints`, `Key-Usage`, `CRL-Distribution-Points` (and others)
`Signature-Algorithm`	`(algorithmID, parameters)`
`Digital-Signature`	signature of `Issuer`

Table 8.1: X.509v3 public-key certificate fields.

Certificate Revocation (cap 8.3)

Certificates have a predefined expiration date, from their validity field. A typical period is 1–2 years. Analogous to conventional credit cards, the validity period may be terminated ahead of time, i.e., the certificate can be **revoked**.

In centralized systems where certificates are signed by CAs, it is expected that the CA issuing a certificate is responsible for making information about revoked certificates available to relying.

For public-key certificates, the most serious **revocation reason** is the compromise, or suspected compromise, of the Subject's private key. Other reasons may include that the key has been superseded by another key prior to the planned expiry; the key owner is discontinuing use of the key; or the Subject (owner) changed job titles or affiliation and requires a new key for the new role.

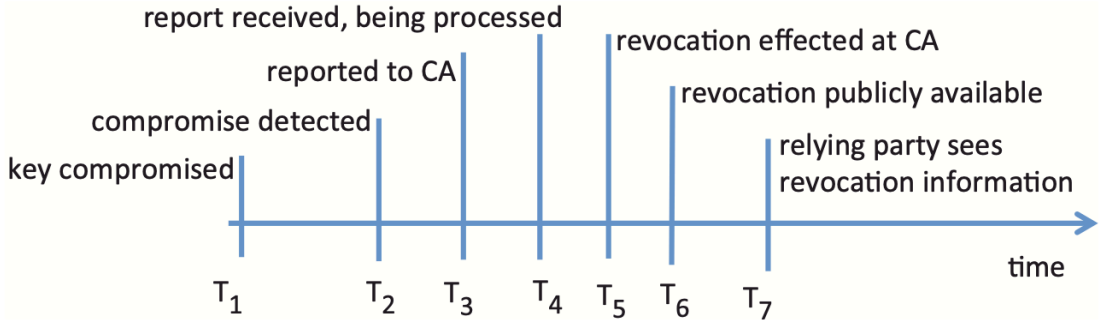


Figure 8.4: Certificate revocation timeline: from compromise to visibility. Possible delays at T4 are mechanism-dependent, e.g., CRLs are typically issued at periodic intervals. A benefit of OCSP mechanisms (over CRLs) is to remove the T6-to-T7 delay.

PKI (pag 216)

A public-key infrastructure (PKI) is a collection of technologies and processes for managing public keys, their corresponding private keys, and their use by applications. Its primary end-goal is to facilitate use of long-term keys used to authenticate entities, and to enable establishment of authenticated session keys.

PKI involves:

1. data structures related to key management (e.g., certificates, formatted keys);
2. use of cryptographic toolkits and related methods for automating key management;
3. architectural components such as CAs and (public-key) certificate directories; and
4. procedures and protocols for approving, acquiring and updating keys and certificates.

Certificate Chain (cap 8.2 pag 217)

CAs issue certificates for end-entities, e.g., human users in the case of email certificates, web servers in the case of TLS certificates.

A CA may also issue a certificate for the public key of another CA, sometimes called an **intermediate CA**, e.g., when the first CA is a trust anchor or atop a hierarchy

This results in the concept of a certificate chain. Before the public key in a certificate is relied on for some intended purpose, the relying party should validate the certificate, i.e., check to ensure that "everything is in order". The steps for validating a certificate include checking that the target certificate:

1. has not expired, and the current date is in the range [Not-Before, Not-After];
2. has not been revoked;
3. has a signature that verifies (mathematically), using the signing CA's public key;
4. is signed by a CA whose public key is (available and) itself trusted;
5. has a Subject or Subject-Alternate-Name matching the semantics of use. For example, if it is supposedly a TLS certificate from a browsed-to domain, the domain name in the certificate should match the URL domain the browser is visiting. If the certificate is for encrypting email to party B, the email address that the mail client believes corresponds to B should match that given in the certificate.2
   use is consistent with all constraints specified in certificate extension fields or policies (e.g., path length, key usage restrictions, name constraints—explained on page 220).
6. if not directly signed by a trust anchor CA, then a valid chain of certificates from a trust anchor to the target certificate must be available, with all the above steps checked for every certificate in the chain. Trust anchors are defined on page 219.

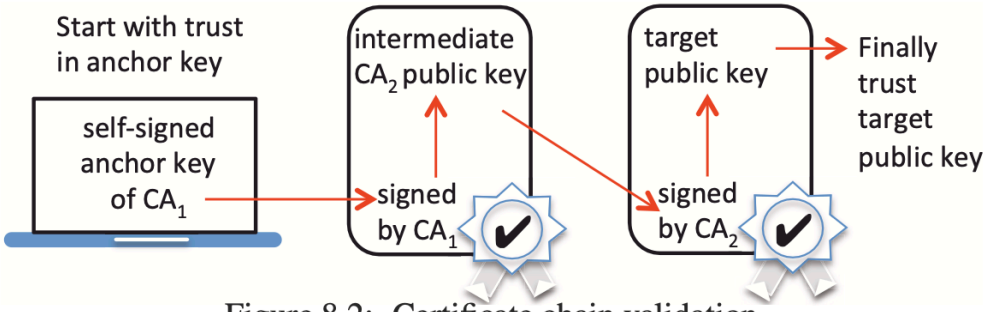


Figure 8.2: Certificate chain validation.

Self signed certificates (pag 219)

A public-key certificate is commonly validated using an in-hand trusted public key to verify the certificate's signature. In contrast, **self-signed** certificates are signed by the private key corresponding to the certificate's own public key. This does not allow deriving trust in one public key from another, but serves as a convenient structure for packaging a public key and related attributes.

Trust in a self-signed certificate should be established by a reliable out-of-band channel. Some browsers have a **trusted certificate store** of self-signed certificates of a large number of established CAs, vetted by the browser vendor; other browsers rely on a similar store maintained by the host operating system, and either option may use a small set stored locally, with a larger set hosted online by the vendor to dynamically augment the local set.

This dictates which TLS (server) certificates the browser, on behalf of the user, will recognize as valid. Such CA public keys relied on as pre-trusted starting points for certificate chains are called **trust anchors**.

(Tutte queste servono a far funzionare la crittografia a chiave pubblica)

[... solarwinds attack ...]

Applications: Credit Card Chip

Past: cards store card information in magnetic stripe (easy to clone)
With chip:

- chips can conducts computations and store data (not disclosed to outside)
- EMV standard

We will cover how public key technologies are used for

- card authentication
- transaction authentication

Credit card chip authentication

Card contains a unique pubic and private key pair

- private key is protected and wil never be disclosed to the outside
- public key is digitally signed by the issuer, so its athenticity can be verified by readers

Credit card transaction authentication

Issuer needs to know whether the transaction is authentic.
Transaction needs to be signed by the card using its private key.

Verified signature:

- to issuers: card owner has approved the transaction
- to honest vendor: enables the vendor to save the transactions and submit them later

Qui subentra la non ripudiabilità: non si può negare di aver effettuato gli acquisti (a meno di denunce per furto)

L13 - 07/10/2024

Bitcoin (cap 13, found only in 2nd edition)

Bitcoin is a communication protocol and peer-based system supporting transfer of virtual currency units denominated bitcoin (BTC). It uses hash functions and digital signatures to implement money.

Money is moved between parties by transactions, its ownership dictated by transaction records, public keys and control of matching private keys distinct from a line of earlier proposals, it does not rely on central trusted authorities.

HISTORY: 2008: the bitcoin white paper; 2009: reference implementation; 2011: Silk Road launches bitcoin as it's currency, as bitcoin can be pseudo-anonymous

Behind the Bitcoin there is no central bank that it can rely on; it is a totally virtual currency (a string of bits). The value is basically given by those who use it.

Nodes

Devices may connect to the Bitocin network as nodes. There are full nodes and light nodes.

- **Full nodes** validate and forward individual transactions and blocks, and mantain UTXO pools. They are the foundation of the peer network, and **miners** rely on access to their functionality (including to manage transactions in memory pools), for inclusion in new blocks.
  - The core mining activity (building new block) is distinct from peer sharing of data, and not all full nodes are also miners. One might thus expect fewer miners than full nodes.
- **Light nodes** or thin clients have a reduced data footprint, aside from using block headers (smaller by a factor of about 1000 than full blocks), they collect and keep transaction details only for transactions of specific interest, e.g. payments for a given user's addresses (keys)

Any node is required to have at least a pair of signing and verification keys related to some digital signature schema (ECDSA is adopted in Bitcoin -> based on elliptic curves)

Addresses

User identities are called **addresses** and are derived from public keys. The basic scheme for deriving an address from an ECDSA public key is as follow; "::" denotes concatenation:

- mainPart = the public key is hashed to create the main part
- checksum = a checksum is typically added to verify the integrity of the address and avoid errors when inputting addresses
- address = the final address is obtained by combining the main part and the checksum.

Addresses are 26-35 bytes (depending on version and format) after base58 encoding ((next)), and are made available selectively to other parties, e.g. by email, text message or 2D barcode.

A consequence of treating public keys as identities is that you can make a new identity whenever you want (you simply create a new fresh key pair, s_k and p_k, via the generateKeys operation in our digital signature scheme)

- if you prefer to be somewhat anonymous for a while, you can just create a new identity and use it

Single-use addresses

Bitcoin addresses hide user identities, providing anonymity. However, as they are in essence **pseudonyms**, reusing one address across many transactions allows user actions to be linked; recall, all transactions are in a public ledger.

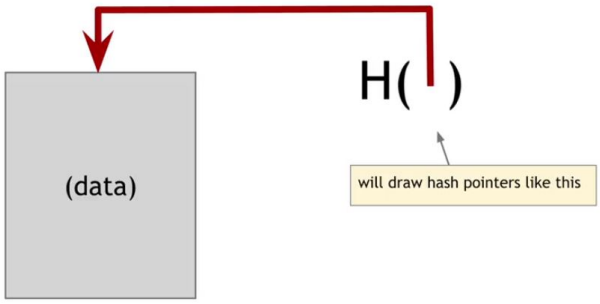
Such **linkability** does not directly expose identities, but when combined with information beyond formal transaction details, anonymity might nonetheless be compromised.

In practice, this motivates per-transaction addresses, e.g. users are encouraged to use software that creates a new Bitcoin address (as receiver) for each incoming transaction, implying a new key pair.

Hash pointer

A hash pointer is a data structure composed by a pointer to where some information is stored together with a cryptographic hash of the information.

Whereas a regular pointer gives you a way to retrieve the information, a hash pointer also gives you a way to verify that the information hasn't changed.



Digital money

A digital money is just a string of bit, as such it can be easily duplicated. How can we prevent the owner from using the same bit string over and over, thus mantaining an infinite supply of money? How can we also prevent that someone else uses our money?

Key challenges:

1. no stealing: only the owner can move his money **(cryptographic signature)****
2. minting: fair money creation **(mint for proof of work)**
3. no double-spending: the same money cannot be used more than once **(global ledger, il libro mastro)**

Cryptocurrency

In order to satisfy requirement 1, Goofy generates a unique coin ID *uniqueCoinID* that he's never generated before and construct the string "CreateCoin [uniqueCoinID]"

He then computes the digital signature of this string with his secret signing key. The string, together with Goofy-s signature, is a coin

Anyone can verify that the coin contains Goofy's valid signature of a CreateCoin statement, and is therefore a valid coin.

[...]

Doublespending

There's a fundamental security

[...]

Global Ledger [...]

Transaction

A transaction is a record which transfers bitcoin from a sender (who own the inout, prior to the transaction) to a receiver (who whill receive the input, as it turns into output). An input is itself the output of a prior transaction.

Transaction fields: table 13.1

[...]

Block subsidy (reward)

A special type of transaction is the **coinbase transaction**. Rather than consume output from an earlier transaction, it creates (mints) units of currency as part of a **block reward**

The reward value is conveyed using a btc-value field. The reward is for solving a puzzle task that requires a huge number of computational steps, called a **proof of work** within block mining.

The block subsidy started at 50 bitcoin in 2009. It is halved every 210,000 blocks (this works out to be about every four years); it became 6.25 bitcoin in May 2020. This is part of a ruleset limiting total currency units to 21 million bitcoin; at the current target average of one block every 10 minutes, the last bitcoin units will be minted in the year 2140, having value 1 satoshi = 10^{-8} BTC= 0.00000001 BTC (the smallest bitcoin unit).

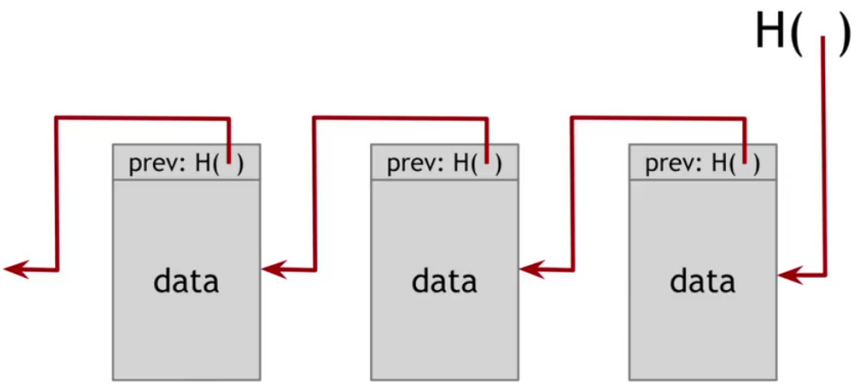
Bitcoin global ledger (pag 377)

Bitcoin uses a special type of **linked list**, called a **blockchain**. Each item or block in the list represents a set of transactions, and also points to its predecessor.

The blockchain and supporting data structures provide a continually updated log of transactions that is:

- **publicly verifiable** (replicated and open for public view; no encryption is used; transactions can be verified to be unaltered and valid according to system-defined rules);
- **append-only** (no parts can be erased; all past transactions remain intact);
- a **ledger** (with offsetting debits and credits, or inputs and outputs in Bitcoin terms).

Linked list



A **linked list** is a basic data structure used to traverse a sequence of items. Each item contains data plus a link to a next item. The link is implemented using either a pointer (memory address) or an index into a table. The end of the list is denoted by a reserved value (e.g., NULL); the start is accessed by a head link (HEAD).

Merkle tree (cap 13.4)

[...]

Block validation

Once a miner has completed a block by collecting enough transactions the block has to be validated and shared with all peers of the bitcoin network

However, different miner may have computed different blocks with different transaction, which of these blocks has to be chosen by the bitcoin network

Here we need a **distributed consensus protocol**

Distributed consensus

There are n nodes that each have an input value. Some of these nodes are faulty or malicious. A distributed consensus protocol has the following two properties:

1.

[...]

Bitcoin consensus algorithm simplified

1. new transactions are broadcast to all nodes
2. each node collects new transactions into a block
3. in each round a random node gets to broadcast its block
4. other nodes accept the block only if all transactions in it are valid (unspent, valid signatures)
5. nodes express their acceptance of the block by including its hash in the next ...

If we choose a random node to broadcast the definitive block it may happen that such a node is not honest, corrupting the blockchain. Can we so give nodes an incentive for behaving honestly?

Two incentives: block reward and transaction fees, **one disincentive**: to finalize a block you have to spend money (proof of work)

Proof of work (pag 389-390 + ...)

[...]

(Se SHA256 venisse bucata cadrebbe la block chain)

Block header

[...]

L14 - 11/11/2024

Laboratorio con approfondimento della Blockchain

L15 - 14/11/2024

Malware [cap 7.1]

Definition

We define malicious software (malware) as software intentionally designed or deployed to have effects contrary to the best interests of one or more users (or system owners or administrators), including potential damage related to resources, devices, or other systems.

1. **How does malware get onto computer devices? [cap 7.1]**

   phishing

   Downloaded executables that users intentionally seek may be repackaged to include bundled malware; users may be tricked to install executables that are either pure malware, or contain hidden functionality

   Computer worms, computer viruses
2. **What makes malware hard to detect?**
3. **How can installation of malware be prevented?**

   social engineering, code-signing, antivirus/malware tools and intrusion detection systems

Viruses and worms [cap 7.2]

We define a coputer virus as the virtual correspective of the biological virus: it's a program capable of infecting other program or files by modifying them to include a possibly evolved copy of itself.

- Viruses need the action of the user in order to spread (he has to run the infected program)

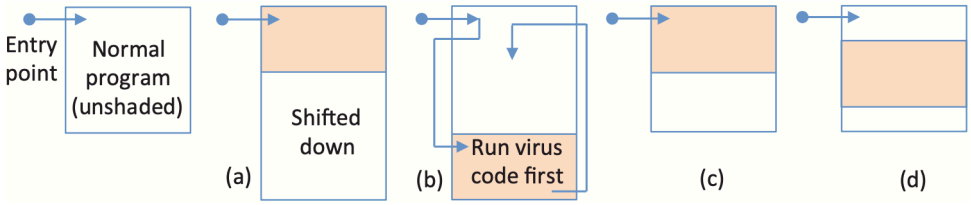
Worms are diffent from viruses:

- they need no human action to spread, as they can do it themselves with no interaction
- they can spread across machines over the network
- they exploit software vulnerabilities, like buffer overflow, while viruses tend to abuse software features.

email-based malware [...]
data file viruses and related malware [...] (macros) -> it is possible to put malevolous code inside of a document

- it's the approach used to infect mobile devices

program file viruses: Most viruses infect executable program files [...] + figure 7.1



Brain virus

Cited as the first PC virus, it's a boot sector virus: when we turn on our device. the boot section is of course the first to be loaded. Well, if you infect this program you'll have a virus that gets loaded every time you turn on your machine.

CIH Chernobyl Virus

Differetnly from the Brain Virus, CIH was very destructive and costly

Impossibility of malware detection

Cap 7.2 sezione **VIRUS DETECTION: UNDECIDABLE PROBLEM**

- proof by contradiction -> comes from the **Halting problem** thinked by Alan Turing (problema dell'arresto)

VIRUS DETECTION IN PRACTICE

malware signatures (deny-list), detection by allow-list (e.g. Tripwire), behavioral signatures which aim to identify malware by detecting sequences of actions pre-identified as suspicious

Virus anti-detection and worm-spreading techniques [cap 7.3]

Encrypted virus: they are weak though 'cause the encryption routine doesn't change

Polymorphic virus: they are based on encrypted virus, but they also have a mechanism to change the encryption routine -> it is not possible anymore to detect that part of the virus (every time the virus infects its body changes)

- they changed some instructions by creating some sintactical equivalent instructions
- we can detect it by running the virus, activating the encryption process that decrypts the virus body. Now we will check the signatures by checking them not in the program, but checking it in memory in the decrypted virus.

Metamorphic virus: they do not use encryption and decryption, but instead on a per-infection basis, the virus rewrites its own code, mutating both its body (infection and payload functionality) and the mutation engine itself

- Elaborate metamorphic viruses have carried source code and enlisted compiler tools on host machines to aid their task.
- Antiviruses execute viruses in a virtual machine in order to find them, but eventually the virus can also acknowledge that it's being executed in a VM and so they behave correclty.

Reverse engineering [...]

Stealth viruses [cap 7.4]

Trojan: that deliver malicious functionality instead of, or in addition to, purported functionality (with the malicious part possibly staying hidden) is called a trojan software.

Backdoor: they allow ongoing stealthy remote access to a machine

Rootkits (worst type of viruses): page 195 book, you cannot see the rootkit through the process list (does not appear in the `ps` command)

- they are viruses that go deep down in the machine components, changing them in order to not let known its presence
- **user mode rootkit**, **kernel mode rootkit** that run in kernel space, with access to all of its resources and memory
- they are the so called **spy-software** (they are used for surveillance purposes)

Rootkit detail [cap 7.5]

hyjacking system calls: Applications acces kernel resources by system calls [...]

hooking + figure 7.4

Kernel mode rootkit changes the system call table, and it is an universal and irreversible change. User mode

Drive-by downloads and droppers [cap 7.6]

Ransomware, botnest and other beasts [cap 7.7]

asymmetric file locking: one criticity: while he's encrypting the disk, the key k is present in memory. We dumb the memory, and if we can find the key before the process of encryption of the disk is finished, we can revert the encryption.

- if we encrypt every file with a different key, we solve this problem but after the payment we have to give back every key for every file to the user.

wannacry: [...]

- [**eternal blue**, un baco scoperto dalla NSA su windows 7, tenuto segreto ma scoperto da un attacco da parte di russi. 2 giorni dopo il leak di questo baco esce wannacry]

L16 - 18/11/2024

Web and Browser Security

La system-security prevede di arrivare in fondo ai protocolli e di capire i meccanismi interni dei sistemi operativi, mentre gli attacchi web possiedono un livello di astrazione molto più elevato, e sono generalmente fatti in javascript.

- I due attacchi richiedono un livello di conoscenze profondamente diverso

We'll see the **XSS, CSRF and SQLi** attacks

Web review [cap 9.1]

The Domain Name System defines a scheme of hierarchical domain names, supported by an operational infrastructure. Relying on this, the Uniform Resource Locators (URLs), such as those commonly disolayed in the *address bar* of browsers, specify the source locations of files and web pages.

Domains and subdomains

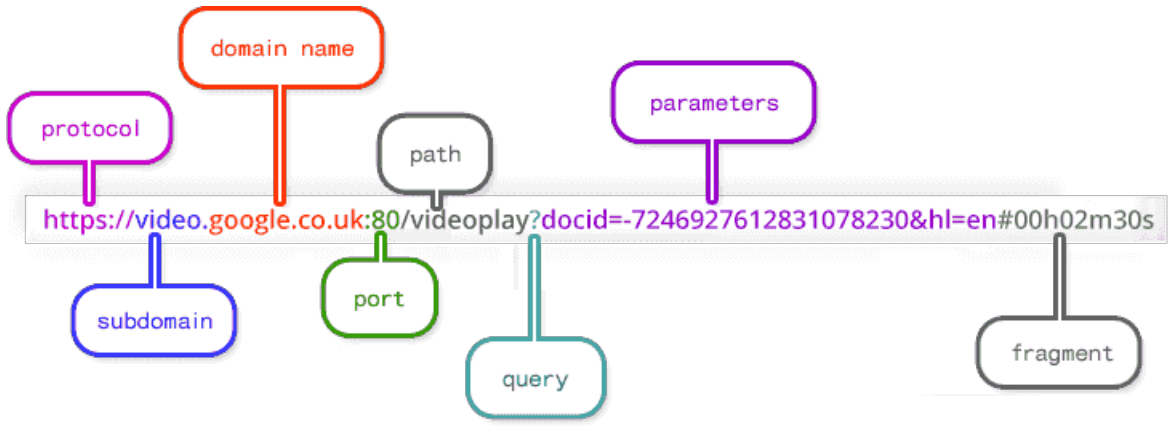
Root contains the addresses of the machines of the various countries (like .it).

Domain namespace is organized in a hierchical tree-like structure. Each node is called a domain, or subdomain.
The root of the domani is called `ROOT` and is denoted with `.`
Below root we have Top-Leve Domain

There are currently 13 organizations maintaining root servers; all name servers configured with the IP addresses of these root servers. All of those DNS are controlled by the USA.
[...]

URL

protocol - subdomain - domain name - port - path - ? query = parameters # fragment



Look at page 247 of the book

HTML

Hypertext Markup Language (HTML) is a system for annotating content in text-based documents.

+ Javascript and HTTP

$$\frac{\text{HTTP}|\quad\quad|}{\text{TCP}\,|\,\text{UDP}}$$
$$\text{IP}$$

- Il protocollo HTTP usa una serie di funzioni che gli vengono fornite dal protocollo TCP.
- Differenza tra TPC ed UDP: UDP non garantisce l'affidabilità della comunicazione, ma è di conseguenza più veloce di TCP, che invece instaura un canale di comunicazione tra due end point affidabile.

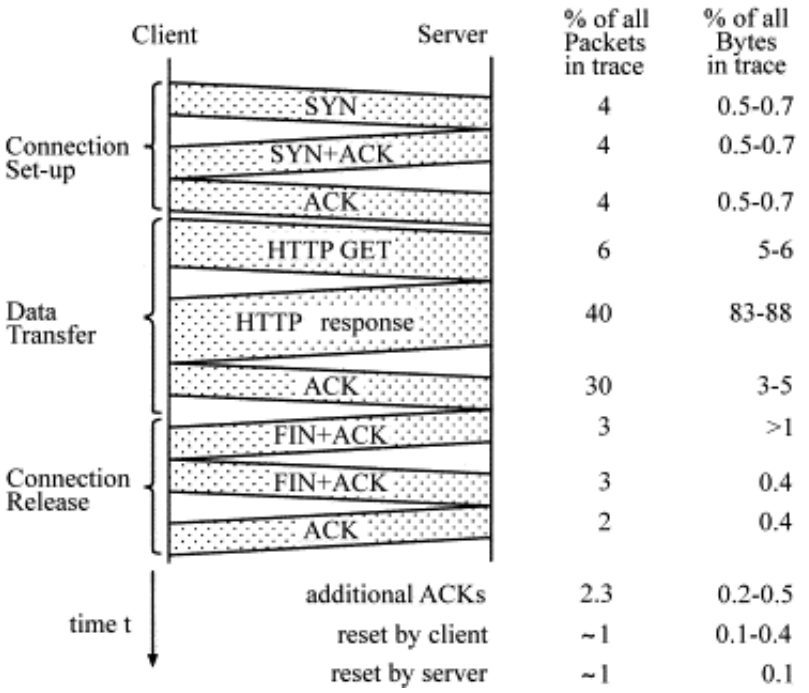
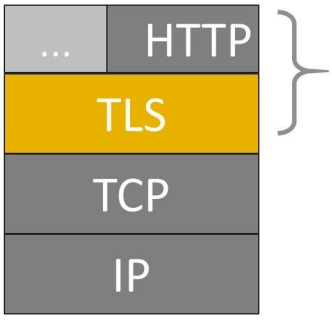


Figure 9.2 page 249

HTTP Proxies

[...]

HTTPS [cap 9.2]



TCP è affidabile, ma ciò non significa che sia sicuro: questa proprietà gli viene fornita da TLS tramite funzioni crittografiche. HTTPS si basa su TLS.

1. Handshake layer
2. Record layer

Si utilizza una crittografia ibridia: viene scambiata tra le due parti una chiave tramite l'utilizzo di una criptazione a chiave pubblica (asimmetrica), e dopodiché si utilizza questa chiave segreta per comunicare tramite l'utilizzo di una criptazione simmetrica.

Key exchange [pag 9]

1. Diffie-Hellman ephemeral -> guardare appunti di crittografia **lezione 14**. Utilizzato per scambiarsi la chiave.
   - attaccabile tramite man-in-the-middle
2. ecc

The client and the server use a Pseudo-Random function (PRF) to calculate the master secret key. PRF is a determinsitic algorithm.

[...]

DOM objects and HTTP cookies

[...]

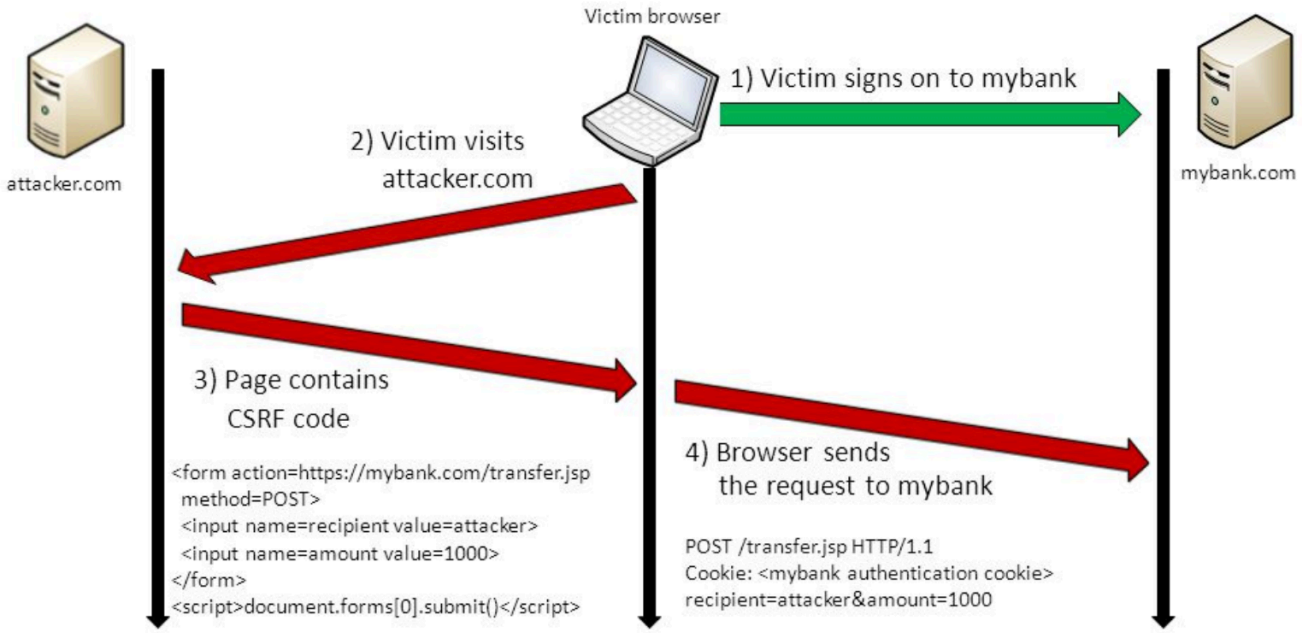
Highlights in the book

[...]

CSRF

There are two main parts to executing a CSRF attack

- **Tricking the Victim**: The attacker lures the victim into clicking a link or interacting with malicious content (e.g., a webpage or email)
- **Forging a Request**: The malicious action sends a request to a website where the victim is authenticated, exploiting their session credentials to execute unauthorized actions (e.g., transferring funds).



L17 - 21/11/2024

XSS

[...]

SQLi

[...]

Network Security

Send information from one computer to another.

- Endpoints are called **hosts**. Could be computers, phones, ecc
- THe plumping is called **link**. We don't care if it is ethernet, wireless, cellular...

Routers/switches: moves bits between links

- circuit switching
- packet switching

Components

- computers are hosts nodes, they send and receive messages
- routers are communication nodes, they pass on messages
- channels: the media which delivers messages
- **LAN**: private network of physically close computers
- **WAN**

Hubs and **switches** connect devices on a LAN:

- ethernet hubs forward all frames to all attached devices. Lots of extra traffic.
- ethernet switches work initially like hubs. Over time, learns the addresses of attached devices. Eventually, only forwards a frame to the destination device. Fewer collisions.

Network Interfaces Cards (NIC)

Computers are connected to a network via a network interface card:

- ethernet card
- wifi adapter

A computer may have multiple network interfaces.

LAN

[...]

WAN

[...]

Ethernet and TCP/IP

Hosts comminucate to each other following predifined protocols.

The internet Protocol (IP) is the main protocol in the internet, TCP/IP protocol suite for packet-switched networks.

While the standard protocol used for communicating on local network is Ethernet.

Circuit and packet switching

Circuit switching:

- legacy phone network
- single route through sequence of hardware devices established when two nodes start communication
- data sent along route
- route maintained until communication ends

Packet switching:

- internet
- data split into packets
- packets transported independently through network
- each packet handled on a best efforts basis
- packets may follow different routes

On internet any host is identified by two addresses:

- address of network card (MAC address)
  - globally unique and unchangeable address stored on the network card
  - ethernet header contains the mac address of the source and the destination computer
- IP address
  - each computer on a network must have a unique IP address to communicate
  - virtual and assigned by software

Protocols

- UDP (connectionless protocol)
- TCP (connection-oriented protocol)

Protocols are developed using a layered approach

- higher layers protocols use the services of lower layers
- a layer can be implemented in hardware or software
- the bottommost layer must be in hardware

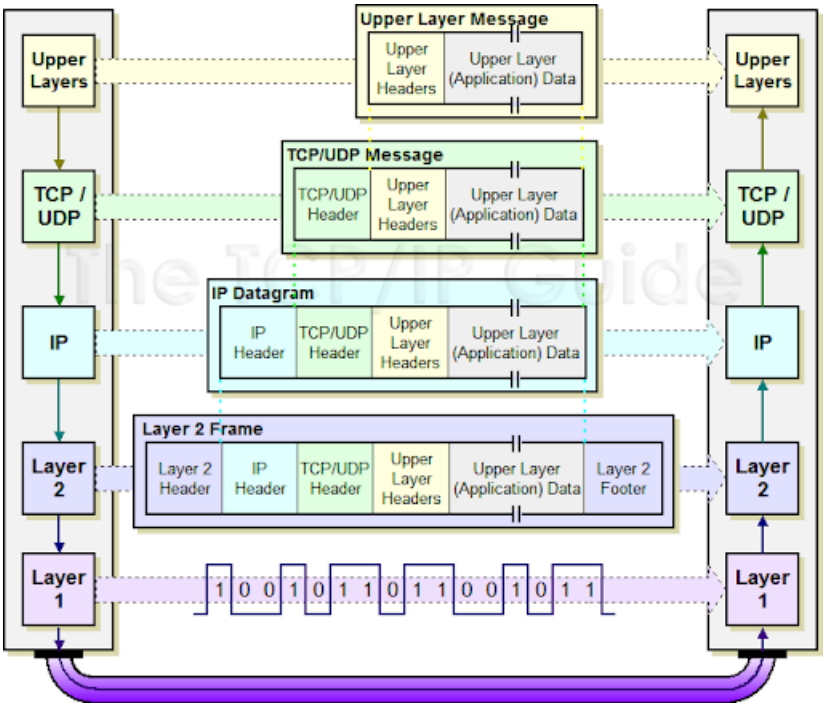
A network device may implement several layers

A communication channel between two nodes is established for each layer

- actual channel at the bottom layer
- virtual channel at higher layers

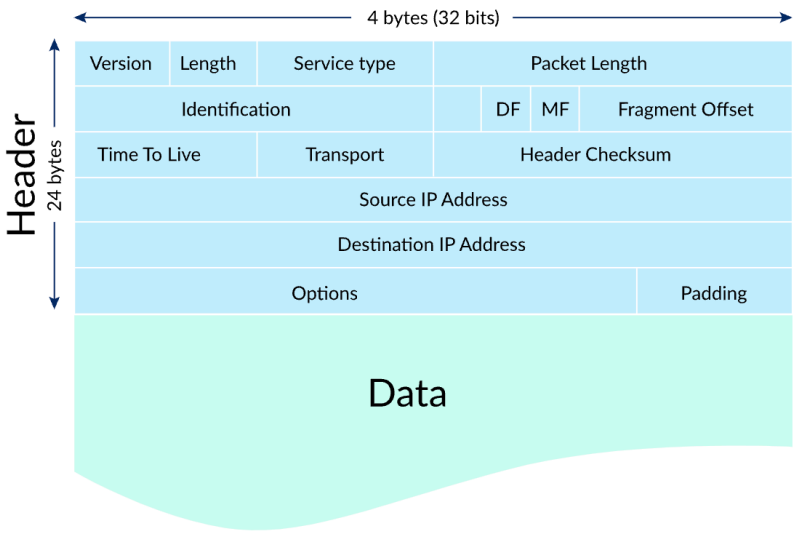
L18 - 25/11/2024

Encapsulation



IP packet

[...]



Unknown MAC address

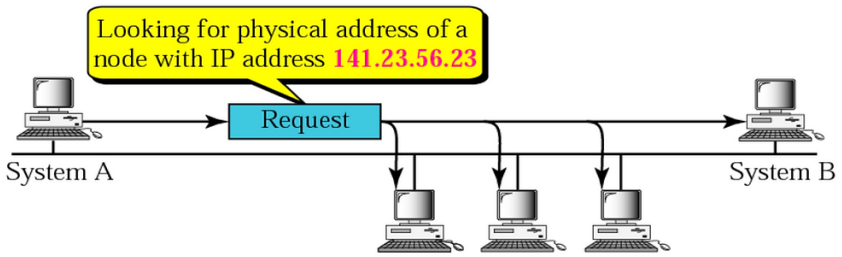
It may happen that the sender has no idea what the MAC address of the destination host is. Usually it has the destination IP address.

Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP)

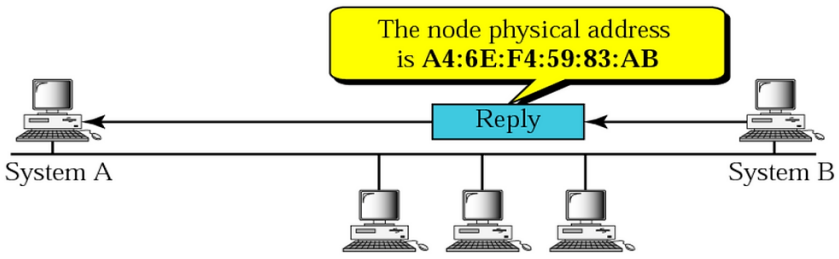
logical address -> ARP -> physical address
physical address -> RARP -> logical address

Un host manda a tutti i sistemi presenti sulla rete una ARP request dove si chiede chi abbia un determinato indirizzo IP. Chiunque abbia quell'indirizzo IP risponde al richiedente con il proprio MAC address.

- L'host conosce l'indirizzo IP e vuole sapere il MAC address di chiunque abbia quell'indirizzo.



a. ARP request is broadcast



b. ARP reply is unicast

TCP / UDP [cap 10.6 pag 304]

PORTS and SOCKETS
TCP HEADER ...

Three Way Handshake [cap 10.6 pag 304]

[...]

Wireshark

[...]

ICMP [pag 305-306]

Network Security

With respect to a single host, a networked environment introduces new assets which can be used by an attacker to compromise the hosts:

- the communication channel
- network devices: routers, switches, hubs
- the communication protocols
- network protocols & applications (ssh, sftp, http, smtp, web applications, etc)

Furthermore the network can be used for delivering malicious content

Capitolo 11 libro

Network attacks:

1. passive attacks: non hanno impatto sul sistema attaccato e sono quindi difficili da rilevare (es l'intercettazione)
2. active attacks: comportano la modifica o lo sfruttamento di una serie di debolezze dei protocolli (es MITM, TCP hijacking, ecc)

Spoofing

IP Spoofing pag 321

Communication channel

Cable

[...]

Radiation

Clever attackers can take advantage of a wire's properties and can read packets without any physical manipulation

Ordinary wire (and many other electronic components) emits radiation

By a process called inductance an intruder can tap a wire and read radiated signals without making physical contact with the cable; essentially, the intruder puts an antenna close to the cable and picks up the electromagnetic radiation of the signals passing through the wire.

Optical Fiber

Optical fiber offers two significant security advantages over other transmission media.

Can be intercepted only physically by cable or wifi

Eavesdropping in data network

[...]

Sniffing as Monitoring [cap 11.3 pag 317]

[...]

Packet sniffers [cap 11.3 pag 316]

[...]

TCDUMP & Wireshark

[...]

The best way to stop packet sniffing is to encrypt packets securely.

- the sniffers will still be able to capture packets, but they will be meaningless

Vulnerability assessment tools

Limitations, cautions

Port scanning and os fingerprinting

MITRE ATT&CK

non utile per l'esame ma utile per il futuro lavorativo

Smurf attack
[...]

Syn Flooding

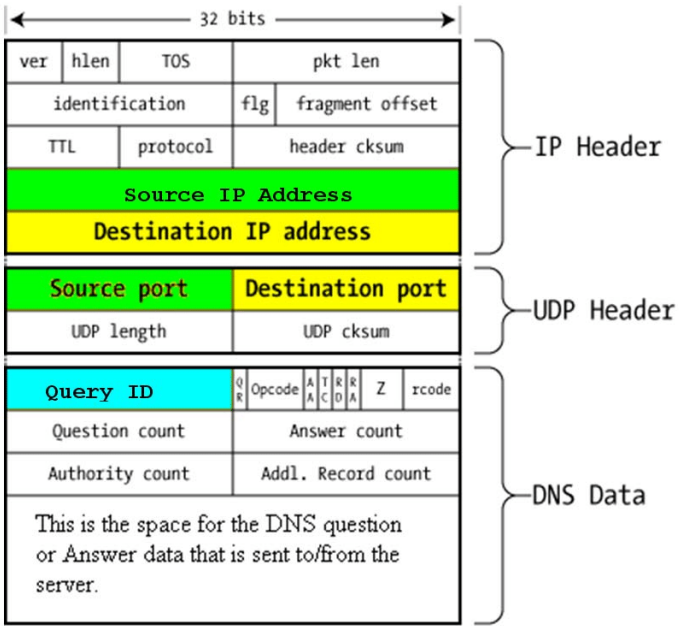
Ingress filtering

L19 - 28/11/2024

MITM

DNS Pharming [cap 11.5]

DNS Packet



DNS Query

Recursive Query to NS (nameservers)

Pharming attack

ARP Spoofing

ARP Poisoning

Figa non ho seguito niente mortacci mia

L20 - 02/12/2024 [cap 10?]

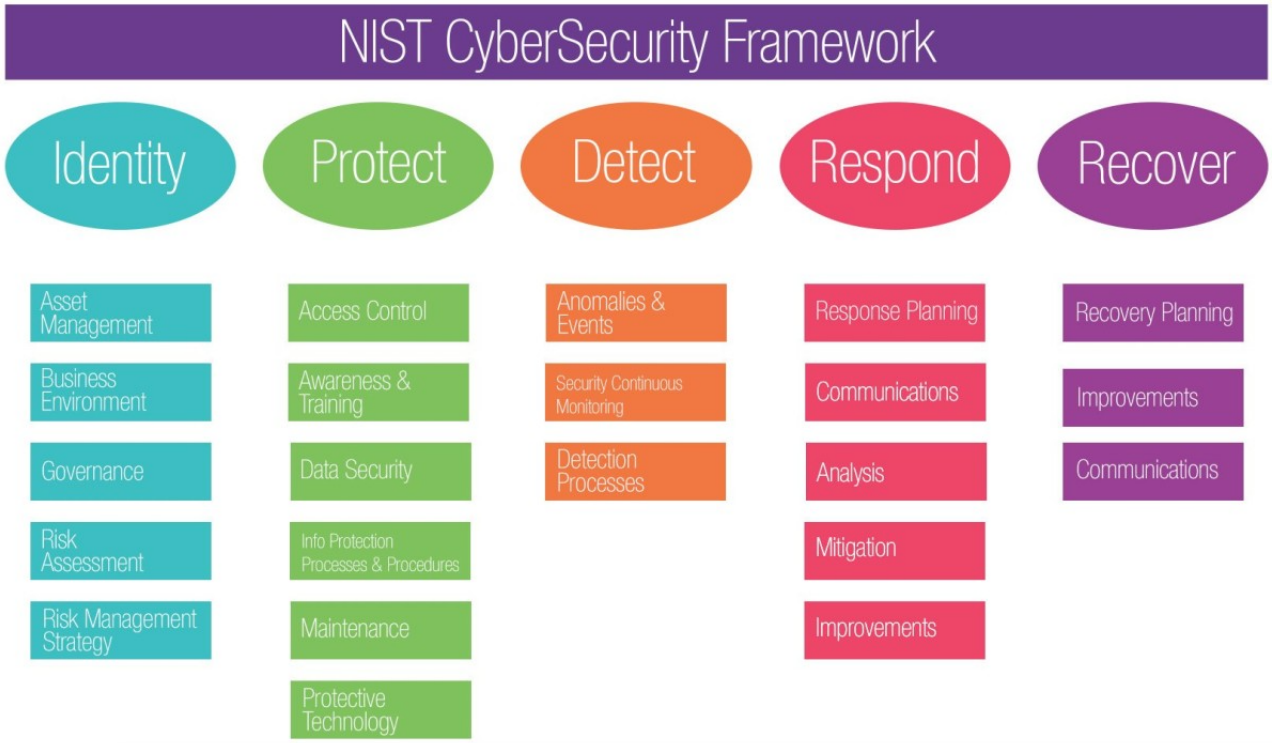
Defense Cybersecurity

Defense Cyber Program

The defensive cyber programmes are:

- defensive cyber operations (DCO) programme
- cuber resilience programme (CRP)

Between them they address cyber risk. They will also help deliver secure cyber fuoundations needed for the Digital Backbone, se out in the Digital Strategy for Defence.



Cyber Resilience Programme

Basic cybersecurity tools

- Cryptography
- cypto protocols (IPSEC, SSL/TLS)
- antivirus
- EDR
- ASLR, stackguard, DEP

Firewall [cap 10.1]

A network security firewall is a gateway providing access control functionality that can allow or deny, and optionally modify, data passing between two networks, or a network and a device.

Inbound and Outbound

Dedicated firewalls and hybrid appliances [pag 287]

[...]

Types of firewall

There are several types of firewall that will prevent potentially harmful information from getting through:

- packet filter
- stateful inspection
- proxy firewall
- personal firewall

Packet-fitler rules and actions [pag 283]

Introduction page 283 + TABLE 10.1 at page 285

Stateless and stateful filters

Packet filtering: cons [...]

Proxy firewalls [cap 10.2]

[...]

Advantages

- better logging handling of traffic
- state aware of services (FTP etc.)
- ecc

NGFW (next generation firewall)

Not in the book

IDS (intrusion detection system) [cap 11]

Intrusion

[...]

NIPS (Network based IPS)

[...]

L21 - 05/12/2024

Lxx - 16/12/2024

Wansview W4 exploitation