

# SII

TEAM CYBERSECURITY

# Traccia Giorno 2

Lo scopo dell'esercizio di oggi è di acquisire esperienza con IDA, un tool fondamentale per l'analisi statica. A tal proposito, con riferimento al malware chiamato «Malware\_U3\_W3\_L2» presente all'interno della cartella «Esercizio\_Pratico\_U3\_W3\_L2» sul Desktop della macchina virtuale dedicata all'analisi dei malware, rispondere ai seguenti quesiti, utilizzando IDA Pro.

1 Individuare l'indirizzo della funzione DLLMain (così com'è, in esadecimale)

2 Dalla scheda «imports» individuare la funzione «gethostbyname». Qual è l'indirizzo dell'import? Cosa fa la funzione?

3 Quante sono le variabili locali della funzione alla locazione di memoria 0x10001656?

4 Quanti sono, invece, i parametri della funzione sopra?

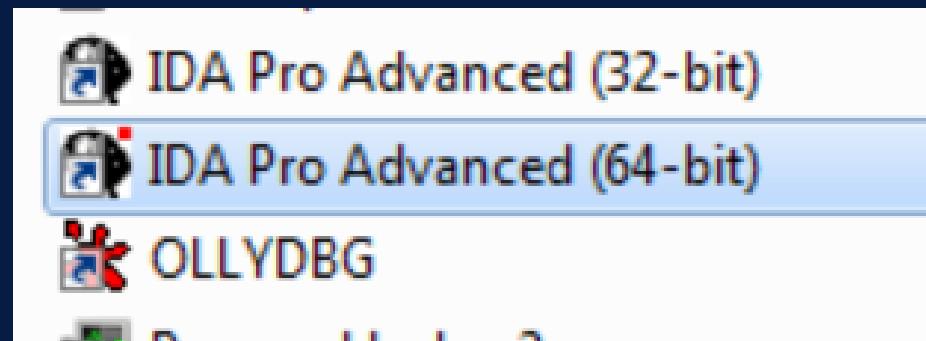
5 Inserire altre considerazioni macro livello sul malware (comportamento)

1

## Individuare l'indirizzo della funzione DLLMain (così com'è, in esadecimale)

DLLMain è una funzione utilizzata nella programmazione di Dynamic-Link Libraries (DLL) su Windows. Viene chiamata dal sistema operativo quando vengono inizializzati o terminati threads e processi. Questa funzione ritorna "TRUE" se ha successo, "FALSE" termina il processo.

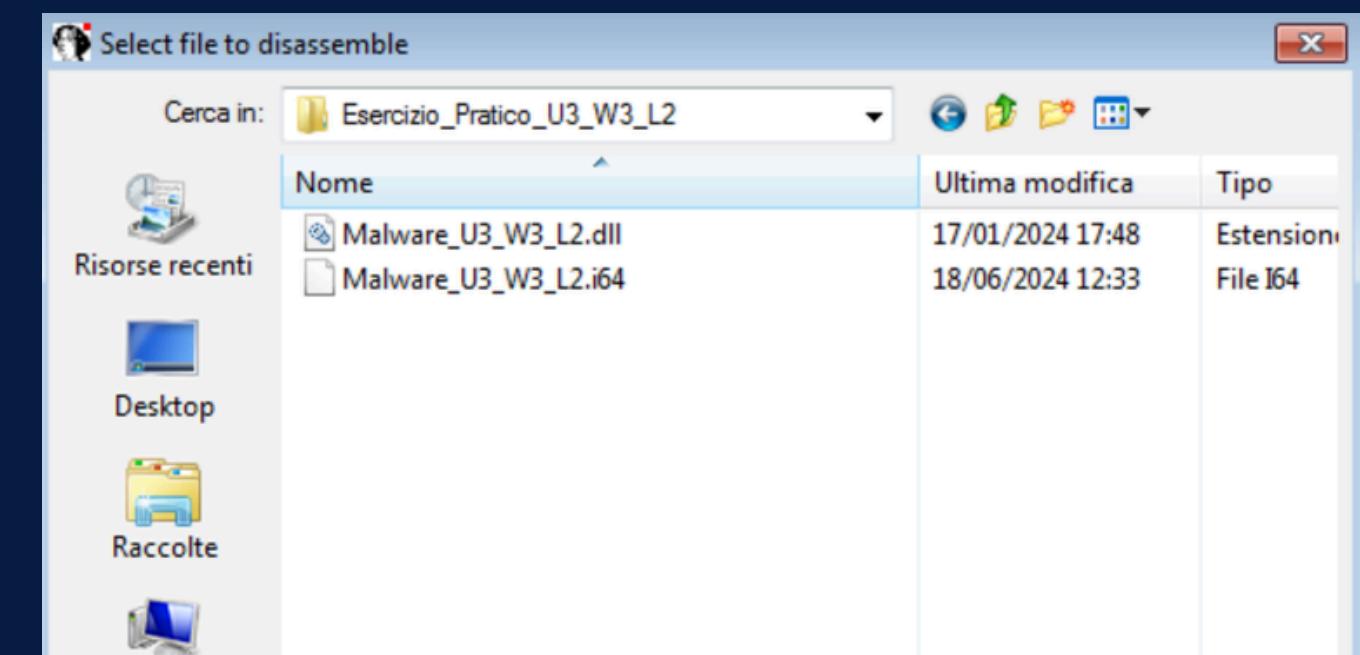
1



Per prima cosa abbiamo aperto il Disassembler IDA Pro Advanced (64-bit)

2

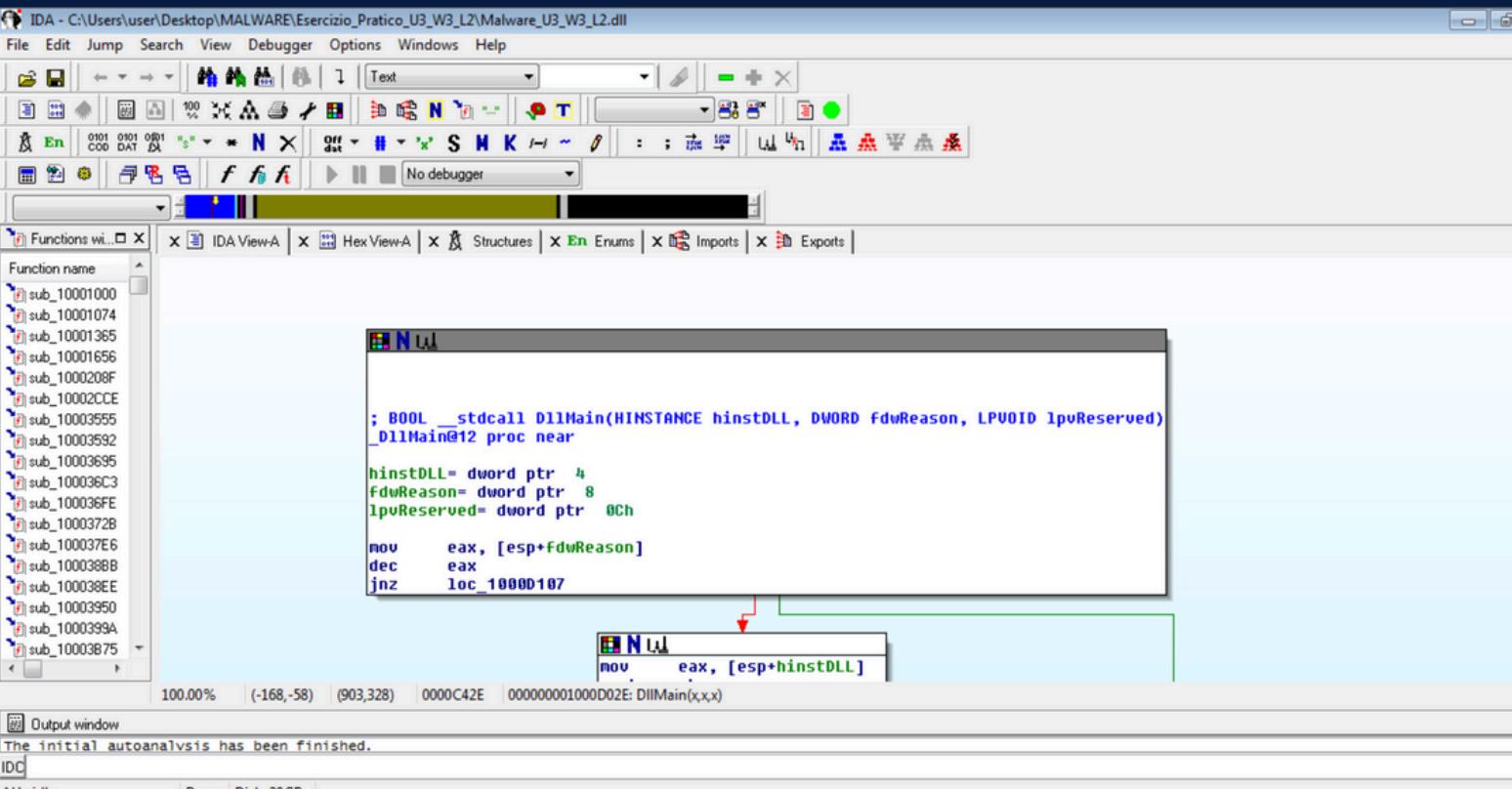
Successivamente abbiamo caricato il Malware\_U3\_W3\_L2 da analizzare



1

# Individuare l'indirizzo della funzione DLLMain (così com'è, in esadecimale)

3



IDA Pro Advanced carica l'eseguibile permettendo la visualizzazione grafica del programma

4

```

IDA View-A | Hex View-A | Structures | Enums | Imports | Exports

.text:1000D02E ; ===== S U B R O U T I N E =====
.text:1000D02E ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD FdwReason, LPVOID lpvReserved)
.text:1000D02E _DllMain@12 proc near ; CODE XREF: DllEntryPoint+4B↑
.text:1000D02E ; DATA XREF: sub_100110FF+2D↓
.text:1000D02E hinstDLL      = dword ptr 4
.text:1000D02E FdwReason     = dword ptr 8
.text:1000D02E lpvReserved   = dword ptr 0Ch
.text:1000D02E             mov    eax, [esp+FdwReason]

```

Premiamo sulla barra View per poter visualizzare la parte di codice in modalità testuale e recuperare l'indirizzo della funzione

2

## Dalla scheda «imports» individuare la funzione «`gethostbyname`». Qual è l'indirizzo dell'import? Cosa fa la funzione?

Nella sezione "Imports", cerchiamo la funzione `gethostbyname`. Facendo doppio clic su questa funzione, otterremo l'indirizzo di memoria in cui è allocata: 100163CC.

Per analizzare la struttura della funzione e comprenderne il funzionamento, possiamo cliccare sulle righe evidenziate in verde. Questo aprirà una vista grafica del codice, facilitando l'esame dettagliato della funzione.

Address	Ordinal	Name	Library
00000000		waveInPrepareHeader	WINMM
00000000		waveInAddBuffer	WINMM
00000000		waveInStart	WINMM
00000000	18	select	WS2_32
00000000	11	inet_addr	WS2_32
00000000	52	gethostbyname	WS2_32
00000000	12	inet_ntoa	WS2_32
00000000	16	recv	WS2_32
00000000	19	send	WS2_32

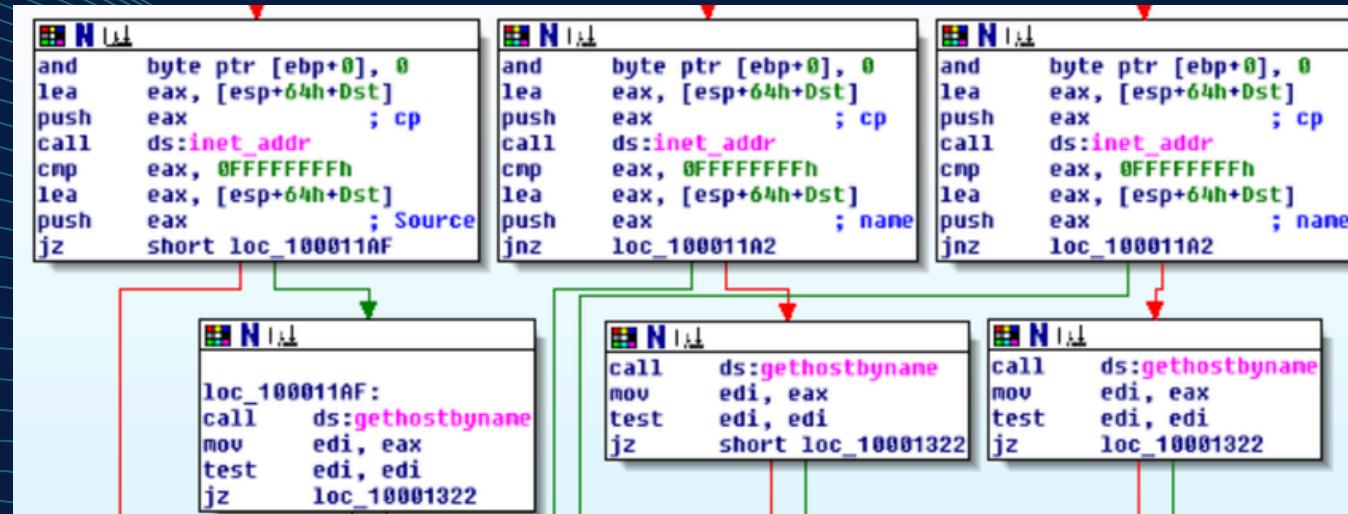
```

idata:100163CC ; struct hostent * __stdcall gethostbyname(const char *name)
idata:100163CC extrn gethostbyname:dword
; CODE XREF: sub_10001074:loc_100011AF↑p
; sub_10001074+1D3↑p ...

```

Dal grafico a sinistra possiamo osservare che il codice effettua un salto (indicato dalla freccia verde) alla locazione **loc\_100011AF**, che contiene la funzione **gethostbyname**. Prima di eseguire il salto, l'indirizzo del dominio (**Source**) viene inserito nello stack e, precisamente, nel registro **eax**, per essere passato come parametro alla funzione.

Quando `gethostbyname` viene chiamata, essa sposta il valore di `eax` nel registro **edi**. Successivamente, effettua un'operazione logica AND bit a bit utilizzando l'istruzione **test** tra `edi` e se stesso. Se il risultato dell'operazione è zero (indicando che il nome del dominio non è stato risolto o che non è stato trovato alcun indirizzo), il flag zero (**ZF**) viene impostato a **1**. In questo caso, viene eseguito un salto alla locazione `loc_10001322`.



# Quante sono le variabili locali della funzione alla locazione di memoria 0x10001656?

Nella versione testuale, ci siamo spostati sull'indirizzo di locazione di memoria richiesto 10001656 ed abbiamo trovato 23 variabili con offset negativo rispetto a EBP

```
.text:10001656 ; DWORD __stdcall sub_10001656(LPUUID)
.text:10001656 sub_10001656    proc near               ; DATA XREF: DllMain(x,x,x)+C8↓o
.text:10001656
.text:10001656 var_675
.text:10001656 var_674
.text:10001656 hLibModule
.text:10001656 timeout
.text:10001656 name
.text:10001656 var_654
.text:10001656 Dst
.text:10001656 Parameter
.text:10001656 var_640
.text:10001656 CommandLine
.text:10001656 Source
.text:10001656 Data
.text:10001656 var_637
.text:10001656 var_544
.text:10001656 var_500
.text:10001656 Buf2
.text:10001656 readfds
.text:10001656 phkResult
.text:10001656 var_3B0
.text:10001656 var_1A4
.text:10001656 var_194
.text:10001656 WSAData
.text:10001656 arg_0
.text:10001656
```

= byte ptr -675h  
= dword ptr -674h  
= dword ptr -670h  
= timeval ptr -66Ch  
= sockaddr ptr -664h  
= word ptr -654h  
= dword ptr -650h  
= byte ptr -644h  
= byte ptr -640h  
= byte ptr -63Fh  
= byte ptr -63Dh  
= byte ptr -638h  
= byte ptr -637h  
= dword ptr -544h  
= dword ptr -50Ch  
= dword ptr -500h  
= byte ptr -4FCh  
= fd\_set ptr -4BCh  
= byte ptr -3B8h  
= dword ptr -3B0h  
= dword ptr -1A4h  
= dword ptr -194h  
= WSAData ptr -190h  
= dword ptr 4

## Quanti sono, invece, i parametri della funzione sopra?

Dallo stesso codice, appena dopo le variabili, possiamo notare un solo argomento passato alla funzione come parametro, avente offset positivo rispetto ad EBP.

IDA ha chiamato questo parametro «arg\_0».

```
.text:10001656 var_194  
.text:10001656 WSAData  
.text:10001656 arg_0  
.text:10001656
```

= dword ptr -194h  
= WSAData ptr -190h  
= dword ptr 4

## Inserire altre considerazioni macro livello sul malware (comportamento)

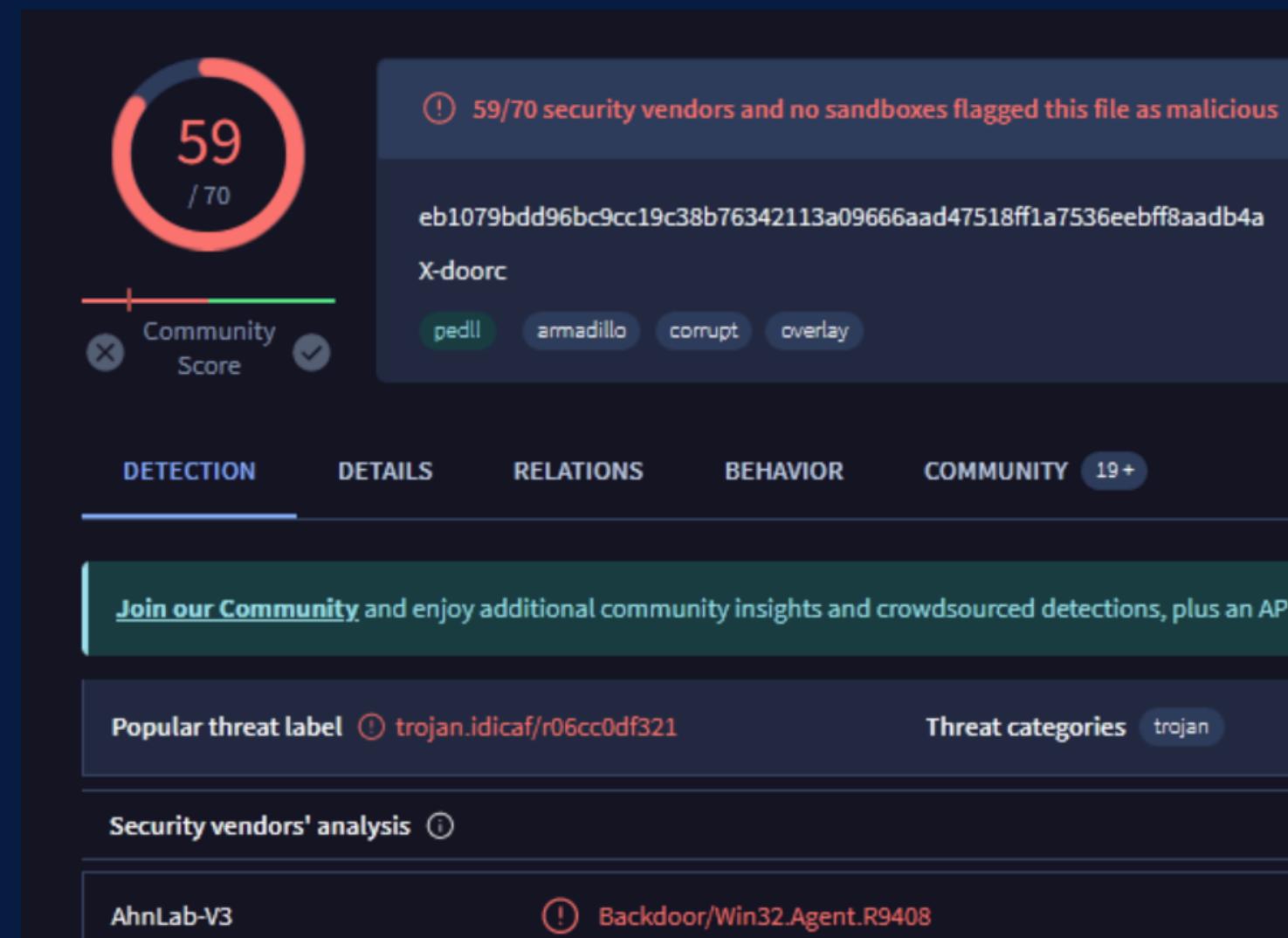
Per fare delle considerazioni più precise riguardo il malware analizzato, abbiamo utilizzato il tool VirusTotal, che ci ha fornito tutte le informazioni necessarie in modo più chiaro rispetto al codice Assembly x86. Come mostrato nell'immagine seguente, il malware è un trojan che installa una backdoor all'interno del sistema. Studiando le diverse funzioni che utilizza, abbiamo osservato i seguenti comportamenti:

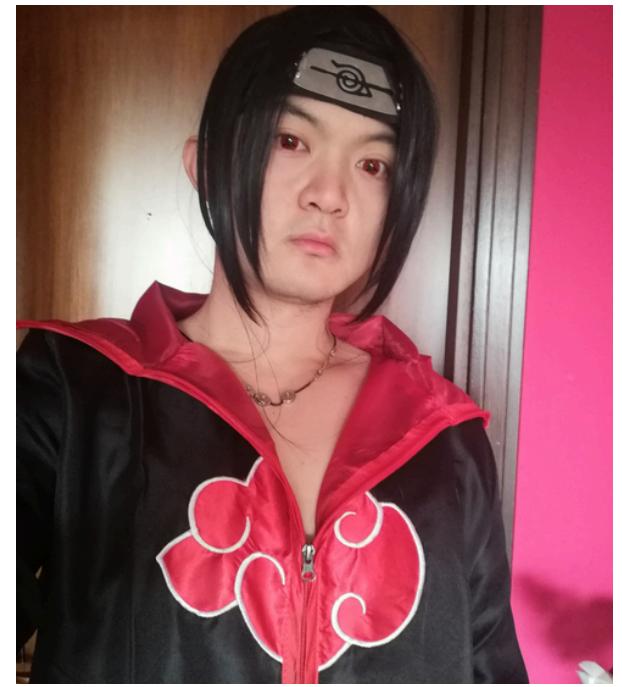
1.Creazione di un Mutex: Il malware utilizza `CreateMutexA` per garantire che solo un'istanza della backdoor sia eseguita alla volta. Questo previene conflitti e riduce la possibilità di rilevamento da parte di software di sicurezza.

2.Persistenza: Per ottenere la persistenza nel sistema, il malware utilizza `GetModuleFileNameA` per leggere il percorso della backdoor. Successivamente, aggiunge questo percorso a una chiave di registro di avvio, assicurando che il malware venga eseguito ogni volta che il sistema si avvia.

3.Comunicazione con il Server di Comando e Controllo (C&C): Il malware tenta di risolvere l'indirizzo IP del server C&C utilizzando `gethostbyname`. Questo consente al malware di stabilire una connessione di rete per ricevere comandi da remoto o inviare file rubati.

Queste osservazioni evidenziano l'efficacia e la pericolosità del trojan, che utilizza tecniche avanzate per mantenere il controllo sul sistema infetto in modo nascosto e persistente.

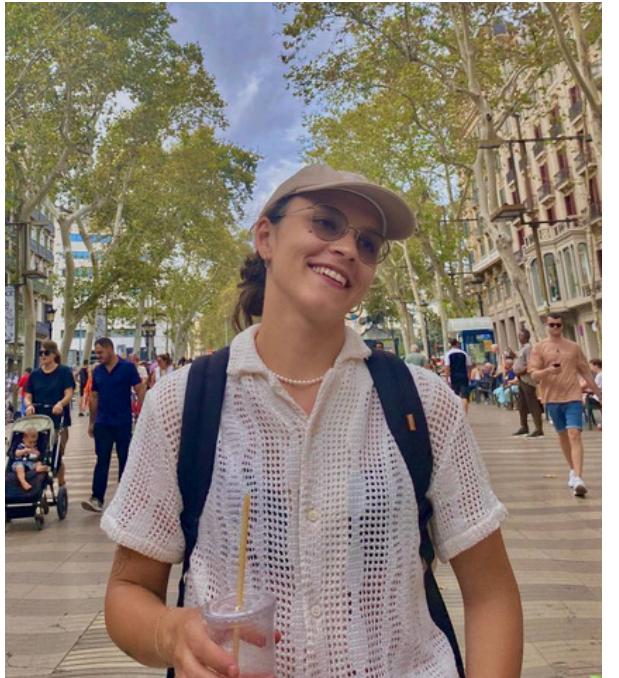




**ZhongShi Liu**



**Mario Marsicano**



**Mara Dello Russo**



**André**

# Our Team

Thank you!

