Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica (Classe L-31)

Tecniche e metodologie di attribuzione per la malware analysis

Laureando
Federico Casenove

Matricola 099210

Relatore
Prof. Fausto Marcantoni

Correlatore
Ing. Emanuele De Lucia

A.A. 2019/2020

Sommario

La progressiva evoluzione degli strumenti di cyber security ha portato ad un sostanzioso aumento degli standard di sicurezza, ma allo stesso tempo anche ad una proliferazione di malware sempre più complessi con altrettante tecniche di anti-detection. Nel tempo vengono create numerose versioni dello stesso malware che per facilità vengono categorizzate in un unico cluster chiamato *famiglia di malware*. In ognuna di queste versioni il malware dispiega diverse variazioni di funzionalità e implementazioni, mantenendo comunque un framework di base comune.

L'obiettivo di questo progetto di tirocinio è quello di categorizzare diversi malware della stessa famiglia ricercando similarità nel codice, da sfruttare poi per creare regole Yara per la detection. Quest'ultime verrano successivamente caricate all'interno di una specifica piattaforma che, attraverso tecniche di machine learning, permetterà la detection di varianti future della stessa famiglia.

Indice

1. Introduzione

Nell'ultimo decennio, di pari passo all'evoluzione degli apparati di sicurezza informatica, abbiamo assistito anche ad un'evoluzione dei malware e in particolar modo delle tecniche di anti-detection [1]. Il numero di malware presenti in rete è aumentato drasticamente con il passare degli anni, rendendo quasi impossibile per gli analisti analizzare ogni singolo sample. La dinamica sopra descritta ha portato inevitabilmente ad un'automatizzazione di questo processo [2], e attraverso l'impiego di svariate tecniche e personalizzazioni. Questo progetto affronta la tematica dal punto di vista del code reuse e l'estrazione di firme statiche estratte dal codice riutilizzato nelle varie versioni dei malware. Nel primo capitolo verrà fatta un'introduzione generale sul contesto all'interno del quale si inserisce il lavoro svolto. Nel secondo capitolo verranno approfondite le fasi della malware analysis, analizzando alcuni tools utili per il suo svolgimento. Nel terzo capitolo vengono trattati gli argomenti di Malware Detection in correlazione alla Machine Learning. Nel quarto capitolo si parlerà in modo specifico delle varie fasi del progetto. Nel quinto ed ultimo capitolo verranno trattate le conclusioni e i risultati ottenuti dal lavoro trattato.

1.1 Obiettivo

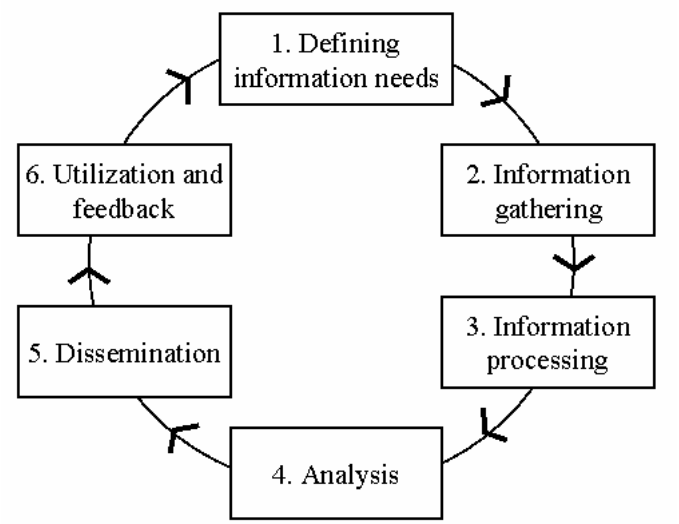
Lo scopo di questo tirocinio è stato analizzare una serie di malware appartenenti ad una determinata famiglia, nello specifico APT-38[1], individuando le funzioni che avevano in comune utilizzando un plugin per il decompilatore IDA Pro chiamato Diaphora [3]. Diaphora permette di confrontare due sample, funzione per funzione evidenziando la percentuale di similarità che c'è tra ognuna di queste. Partendo da questo pre screening dei sample, si identificano le funzioni più caratterizzanti e poi si estraggono le regole

[1] Attribuito da fonti online al gruppo Lazarus Group e piú in generale alla Corea del Nord

Yara dal codice. Queste regole sono state successivamente elaborate da una piattaforma messa a disposizione dall'azienda Telsy con la quale, attraverso degli algoritmi di Machine Learning, la stessa sarà capace di fronteggiare future analisi sapendo in quale famiglia categorizzare il malware in discussione, mettendo a disposizione le similitudini riscontrate.

1.2 Cyber Threat Intelligence

La Cyber Threat Intelligence (CTI) [4] è un processo circolare di analisi di dati appartenenti al dominio cyber e prevede 6 fasi ben distinte:



1. Fase di planning: in cui vengono definiti gli obiettivi dell'analisi, quindi a quale domande si vuole rispondere.

2. Fase di raccolta: in questa fase vengono raccolti i dati grezzi da varie sorgenti (antivirus, sistemi perimetrali, internet, ecc.)

3. Fase di correlazione e valorizzazione dei dati raccolti: in questa fase si trasformano i "dati grezzi" raccolti precedentemente, in "ïnformazione".

4. Fase di analisi e produzione: le informazioni raccolte vengono organizzate in un report riassuntivo.

5. Fase di disseminazione: le informazioni e il report di analisi viene disseminato ai vari stakeholders al fine di supportare il processo di detection.

6. Fase di feedback: gli stakeholders forniscono delle valutazioni su ciò che è stato fornito nella fase precedente così da riattivare il ciclo.

Questo processo serve per comprendere, predire e anticipare una possibile minaccia, che provenga da un singola persona, un gruppo di criminali o uno stato. Il processo di analisi può essere di tre tipologie:

- Strategico: ha lo scopo di analizzare le minacce da un ponto di vista strategico di alto livello in cui non viene affrontato alcun aspetto tecnico.
- Tattico: si prefissa l'obiettivo di analizzare la minaccia mettendo a sistema i vari elementi tecnici identificando le cosí dette TTP (Tattica, Tecnica, e Procedura).
- Operativo: ha lo scopo di analizzare le informazioni di uno specifico attacco da un punto di vista puramente tecnico, isolando gli identificatori utili alla detection.

Per utilizzare tutto il potenziale della CTI per lo studio di cyber attacchi, è necessario basarsi sugli indicatori di compromissione (IOC). Sono delle informazioni che possono essere usate per identificare una specifica compromissione. Gli IOC sono classificati in due tipologie:

- Basati sull'host: informazioni trovate su un host, come ad esempio una chiave di registro, il nome di un processo, una stringa, ecc.;
- Basati sulla rete: informazioni trovate nella rete, come il nome di un dominio, un URL, un indirizzo IPv4 o IPv6, ecc.;

Le informazioni raccolte durante il processo di CTI vengono poi condivise al fine di permettere alle organizzazioni di comprendere meglio i rischi a cui sono esposte, allargare i contatti tra le varie organizzazioni condividendo le proprie conoscenze, ma cosa più importante, condividendo queste informazioni si arricchisce il bagaglio informativo riguardante un determinato argomento in modo tale da poter rispondere più velocemente a minacce future.

Prima di essere condivise vengono categorizzate per livelli di accessibilità in funzione della delicatezza delle informazioni da condividere. Questa fase prende il nome di Classification of Information Sharing [5]. Vengono classificate in quattro colori:

- Red: le informazioni non possono essere divulgate al di fuori di una cerchia ristretta di partecipanti.
- Amber: le informazioni possono essere divulgate ma solo ad una cerchia ristretta di persone che fanno parte dell'organizzazione.
- Green: le informazioni possono essere divulgate ma solo tra organizzazioni o persone conosciute.
- White: le informazioni non hanno restrizioni e possono essere divulgate in modo pubblico.

1.3 Malware Attribution

La Malware Attribution [1] si riferisce al processo di identificazione dell'attore responsabile dell'attacco o lo sviluppatore. Durante lo svolgimento di questo lavoro utilizzeremo il termine in relazione al processo di identificazione della famiglia di appartenenza del malware in analisi. Quando si sta effettuando un'analisi su di un malware si analizzano le sue caratteristiche, queste possono essere ritrovate su versioni future di quello stesso malware, alla fine dell'analisi viene assegnato un nome, e qualsiasi sample che verrà riscontrato in futuro con delle caratteristiche simili potrà quindi essere riconducibile a quella stessa famiglia. Negli anni, il costante monitoraggio e il complesso lavoro di analisi delle compagnie di CTI ha permesso di identificare molte campagne e famiglie di malware e inoltre di ipotizzare una possibile riconducibilità (spesso piú certa di altre) al singolo attore, gruppo hacker o anche Stati. Di seguito possiamo vedere una figura [6] che rappresenta alcuni nomi di famiglie che vengono utilizzati nel mondo della Malware Attribution, tra cui possiamo trovare la famiglia analizzata in questo scritto.

APT Groups and Operations

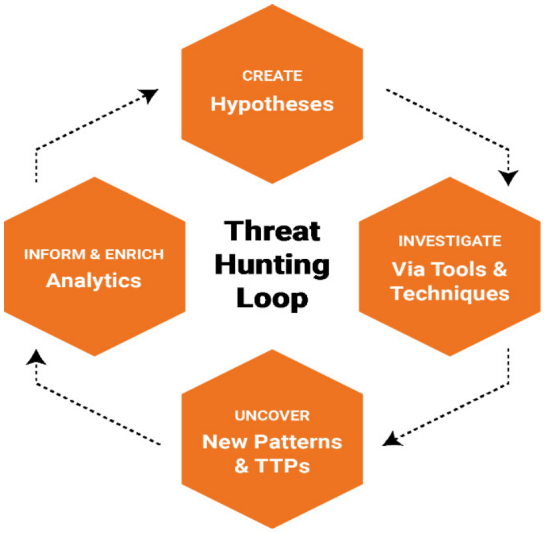
README | China | Russia | North Korea | Iran | Israel | NATO | Middle East | Others | Unknown | _DLL Sideloading | _Download | _Schemes | _Malware | _Sources

North Korea

Common Name	CrowdStrike	Talos Group	Dell Secure Works	Other Name 1	Other Name 2	Other Name 3	Other Name 4	Other Name 5	Other Name 6	Other Name 7	Other Name 8	Rep. of Korea FSI	MITRE AT&CK
Lazarus Group	Labyrinth Chollima	Group 77	Hastati Group	Bureau 121	Unit 121	Whois Hacking Team	NewRomanic Cyber Army Team	Zinc	Appleworm	Hidden Cobra	Nickel Academy		G0032
Group123	Ricochet Chollima	Group 123	ScarCruft	APT37	Red Eyes	Reaper	Venus 121 (금성121)	THALLIUM					G0067
DarkHotel				Luder	Karba	Tapaoux	Nemim	APT-C-06	SIG25	Dubnium	Fallout Team		G0012
Andariel	Silent Chollima											Andariel	
Kimsuki	Velvet Chollima			Kimsuky									G0094
NoName													
OnionDog													
TEMP.Hermit				APT38									G0082
?													
Stardust Chollima	Stardust Chollima			APT38	ElectricFish	BlueNoroff							G0082

1.4 Cyber Threat Hunting

Quando si parla di Cyber Threat Hunting [7] ci si riferisce a tutte le tecniche manuali o machine-assisted utilizzate da un analista per rilevare un incidente di sicurezza. L'analista può usare diverse tecniche per individuare un attacco. Una volta completata l'analisi, l'analista migliora le sue capacità di automatizzare il rilevamento di quell'attacco attraverso la comprensione del relativo TTP (Tactics, Techniques e Procedures). Un Cyber Threat Hunter di solito segue quattro step principali:



1. Si crea un'ipotesi sulla base delle attività sospette che sono state osservate sull'infrastruttura;

2. Si avvia un'investigazione per validare che l'ipotesi iniziale sia corretta. L'investigazione può essere svolta utilizzando differenti tecniche e strumenti (machine learning, linked analysis, data traversal analysis, pivoting, stacking, searching, grouping, etc.)

3. L'investigazione individua un nuovo pattern comportamentale e viene definito (o associato) il TTP (Tactics, Techniques e Procedures) all'attaccante. Il nuovo TTP viene immediatamente condiviso internamente al gruppo e con le strutture esterne di cui ci si possono fidare e viene usato per verificare che non sia stato utilizzato in altre campagne di attacco.

4. Sulla base di un valido TTP, l'hunter può arricchire il pattern integrando nuove informazioni e infine, si potrà creare un metodo autonomo che permetta l'individuazione dello stesso schema di attacco.

Esistono tre tipologie di approccio relative al Cyber Threat Hunting:

- Guidato da ipotesi: in base alla conoscenza delle vulnerabilità, si fanno delle ipotesi sulla possibilità di essere potenzialmente attaccati o essere attaccabili.
- Basato su Indicatori di Compromissione (IoC): gli IoC sono informazioni tecniche che ad esempio un SOC può subito applicare per verificare se è avvenuta una compromissione o se è presente un malware sui sistemi. Comprendono anche delle regole sulle attività da svolgere in caso di compromissione, rendendo quindi tutto il processo di Incident Management molto più efficiente.
- Security Analytic: ovvero basato su analisi comportamentali e su dati statistici/-Machine learning, in caso di anomalie rispetto a comportamenti, osservate su reti o sistemi, scattano in automatico degli alert da gestire.

Il migliore approccio è quello di potersi dotare di una piattaforma per la gestione della Cyber Threat Intelligence (TIP) e integrare i cyber threat feed, suddivisi per tipologia di competenza e know-how ma anche su base territoriale, quindi informazioni provenienti da diverse fonti geografiche quali ad esempio USA, Russia, Cina, Israele, ecc.

1.5 YARA

Una delle tecniche più efficaci e largamente utilizzate nella Malware Detection/Malware Attribution è quella basata sulla firma. Inizialmente le firme erano i semplici hash dei file capaci di identificare in maniera univoca un particolare sample del malware, ma con l'evoluzione dei malware si sono evoluti anche i vari meccanismi di rilevazione. Per firma oggi si identifica una qualsiasi caratteristica capace di identificare un sample o una famiglia di malware, come per esempio una serie di stringhe, una squenza di byte, ecc. Ultimamente è aumentato l'uso di YARA [8], uno strumento open source con il fine di aiutare gli analisti di malware ad identificare e classificare campioni di malware. YARA è multipiattaforma, quindi supportato da Windows, Linux e Mac OS X, e può essere

utilizzato da riga di comando o dai propri script Python. Con YARA è possibile creare descrizioni di famiglie di malware (o qualunque cosa si desideri descrivere) in base a schemi testuali o binari. Ogni regola consiste in un insieme di stringhe e un'espressione booleana che ne determina la logica.

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

La regola sopra riportata dice a YARA che qualsiasi file analizzato contenente una delle tre stringhe deve essere segnalato come "silent banker". Questo è solo un semplice esempio, è possibile creare regole più complesse e potenti utilizzando stringhe senza distinzione tra maiuscole e minuscole, espressioni regolari, operatori speciali. Per un maggiore approfondimento su come possono essere implementate le regole YARA è possibile consultare la documentazione YARA.

YARA viene principalmente utilizzato dai ricercatori e dagli analisti, ed essendo open-source permette ad un organizzazione privata di poter sviluppare una piattaforma per la detection di malware autonomamente. Nella fase di detection attraverso le Yara rules, il numero di falsi positivi è inversamente proporzionale alla qualità della scrittura delle regole. Una regola dettagliata e precisa permette di avere un ottimo ed affidabile livello di detection. Andando a studiare le varie versioni dei sample all'interno della stessa famiglia è possibile identificare quella parte di codice riciclato per poi scrivere una regola YARA, limitando così la detection di falsi positivi.

2. Malware Analysis

La Malware Analysis [9] è il processo di studio del funzionamento di un determinato malware e dei suoi effetti collaterali. Si suddivide in due approcci utili a comprendere meglio il funzionamento del malware che si sta analizzando, ovvero l'Analisi Statica e l'Analisi Dinamica. Prima di scendere nel dettaglio delle due metodologie di analisi è necessario specificare alcune delle tipologie più comuni di malware:

- RAT (Remote Access Tool): sono una tipologia di malware che creano una back door virtuale sulla macchina, la quale garantisce un accesso remoto all'attaccante.
- Spyware: vengono usati per raccogliere informazioni dal sistema su cui sono installati e per trasmetterle al diretto interessato. Le informazioni ottenute possono variare in base alle necessità dell'attaccante.
- Adware: software che presentano all'utente messaggi pubblicitari durante l'uso, a fronte di un prezzo ridotto o nullo. Possono causare danni quali rallentamenti del PC e rischi per la privacy in quanto comunicano le abitudini di navigazione ad un server remoto.
- Backdoor/bot: Sono dei programmi che consentono un accesso non autorizzato al sistema su cui sono in esecuzione. Tipicamente si diffondono in abbinamento ad un trojan o ad un worm, oppure costituiscono una forma di accesso lecita di emergenza ad un sistema, inserita per permettere ad esempio il recupero di una password dimenticata.
- Rootkit: sono composti da un driver e a volte da copie modificate di programmi normalmente presenti nel sistema. I rootkit non sono dannosi in sé, ma hanno la funzione di nascondere, sia all'utente che a programmi tipo antivirus, la pre-

senza di particolari file o impostazioni del sistema. Vengono quindi utilizzati per mascherare spyware e trojan.

- Ransomware: Virus che cripta tutti i dati presenti su un disco, secondo una chiave di cifratura complessa; per ottenerla e decrittografare il computer, bisogna pagare l'attaccante che ha infettato il pc e quindi ottenere la chiave di cifratura per "tradurre" i dati.

- Cryptojacking: è un malware che ha lo scopo di rubare la potenza di calcolo della CPU, per minare criptovalute.

Lo scopo dell'analisi del malware è quello di studiarne il funzionamento, estrarne le caratteristiche e scrivere delle regole per la sua detection. Quando si esegue un analisi su un malware esso sarà illeggibile all'occhio umano. Per capire il suo comportamento sará necessario utilizzare degli strumenti specifici e alcuni trucchi al fine di identificare le informazioni necessarie.

2.1 Analisi Statica

L'analisi statica consiste nell'esaminare il file senza eseguirlo. L'analisi statica può confermare se un file è dannoso, fornire informazioni sulla sua funzionalità e talvolta fornire informazioni che permetteranno di produrre semplici firme. Di base l'analisi statica è semplice e può essere rapida, ma è in gran parte inefficace contro malware sofisticati e può lasciarsi scappare comportamenti importanti. Consiste anche nel processo di reverse engeneering del codice del malware, caricando il file in un disassembler e guardando il programma istruzione per istruzione al fine di scoprire il suo funzionamento. Le istruzioni sono eseguite dalla CPU, quindi un'analisi statica dirá esattamente cosa fa il programma.

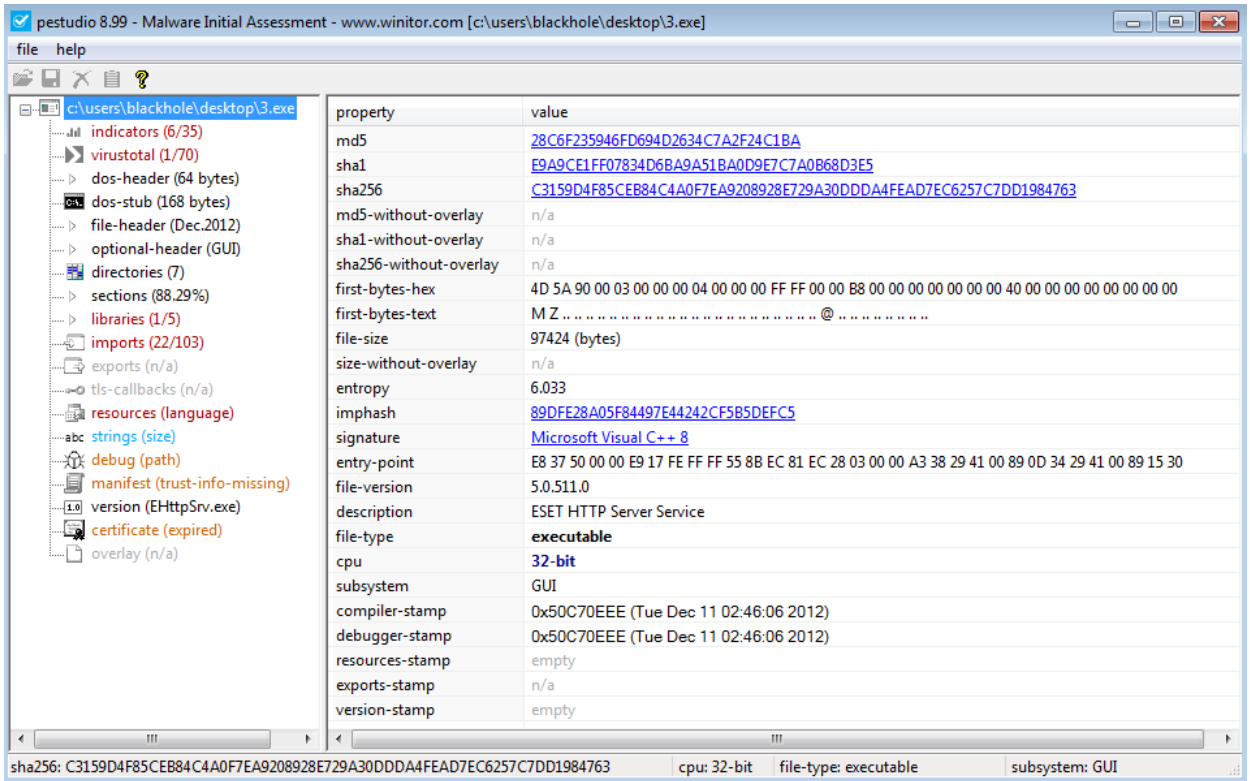
Di seguito vengono riportati alcuni passaggi:

- Ispezione del formato del file: i metadati del file possono fornire delle informazioni molto utili, per esempio, i file Windows PE (portable executable) possono fornire molte informazioni sul tempo di compilazione, funzioni importate ed esportate, ecc.

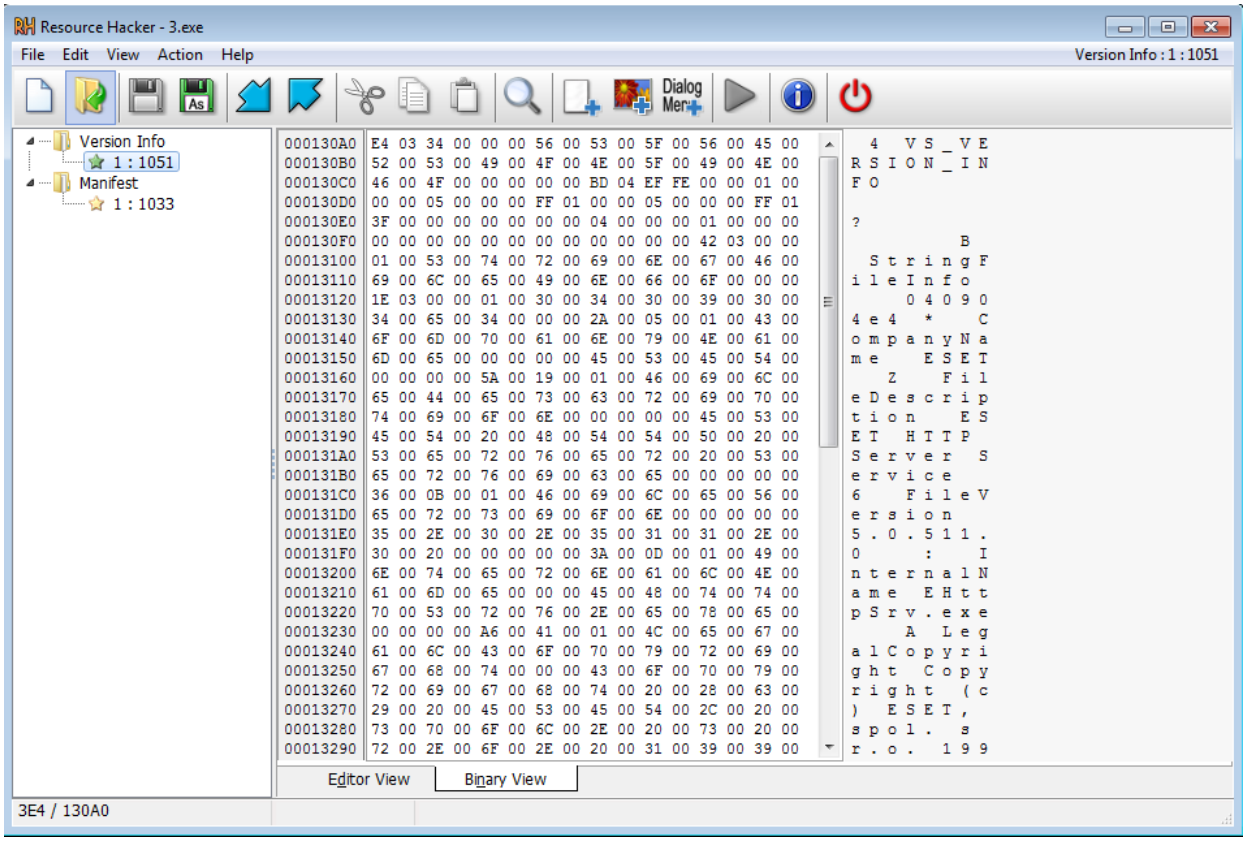
- Estrazione delle stringhe: ci si riferisce all'esaminazione dell'output che genera il software e informazioni riguardanti le operazioni fatte dal malware.
- Fingerprinting: questa fase include gli hash del file crittografato, trovare gli artefatti, come il nome del file o le stringhe dei registri.
- AV scanning: se il file che si sta analizzando porta con sé delle firme di una famiglia di malware già analizzati, o è un malware noto, molti antivirus saranno capaci di rilevarlo.
- Disassembly: in questa fase si fa reverse engeneering sul codice assembly del file al fine di dedurre la logica e le intenzioni del file analizzato.

I passaggi sopra elencati possono essere effettuati attraverso l'utilizzo di alcuni tools utili per questo tipo di analisi, come ad esempio:

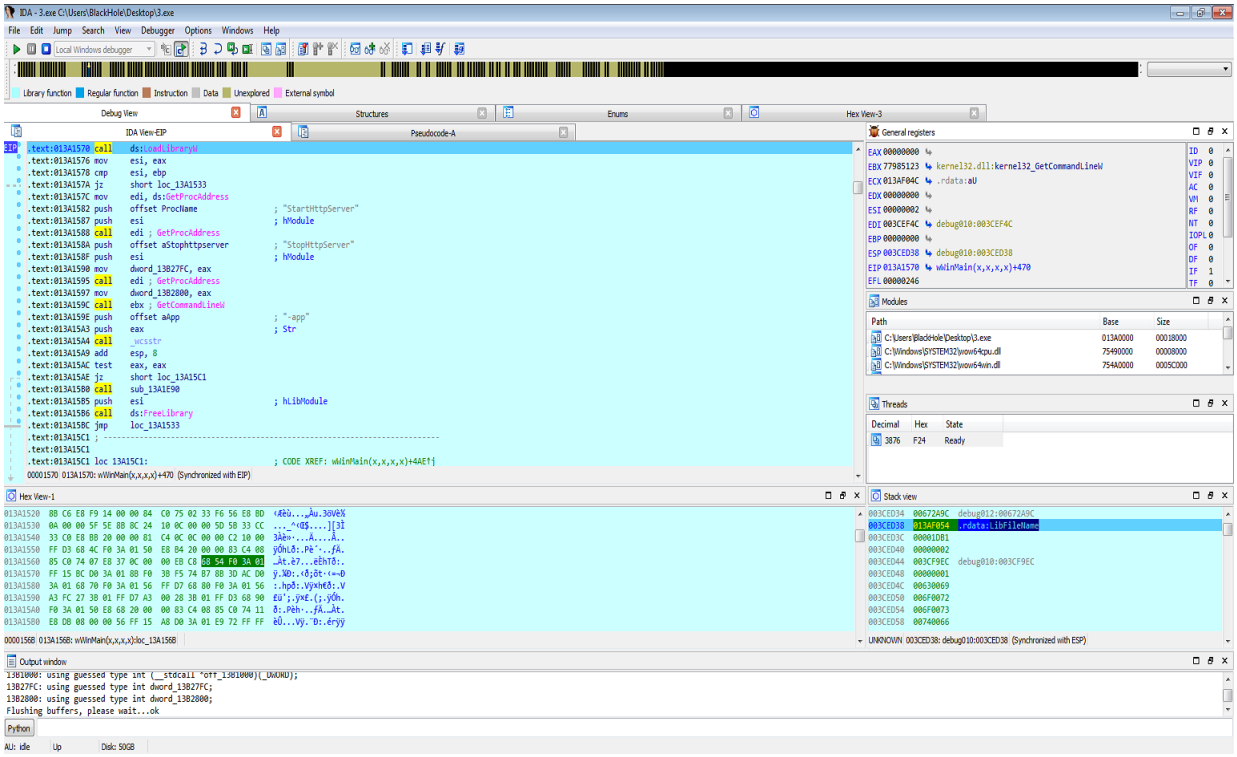
- PEStudio [10]: è un tool che permette l'analisi statica del file che si sta analizzando, estrapolando molte informazioni utili, come ad esempio la signature e il formato. Permette l'analisi con Virustotal [11] e cosa molto utile estrapola tutte le stringhe presenti nel file.



- Resource Hacker [12]: é un editor di risorse per applicazioni a 32 e 64 bit. È sia un compilatore di risorse sia un decompilatore, che consente la visualizzazione e la modifica delle risorse negli eseguibili e le librerie di risorse compilate.



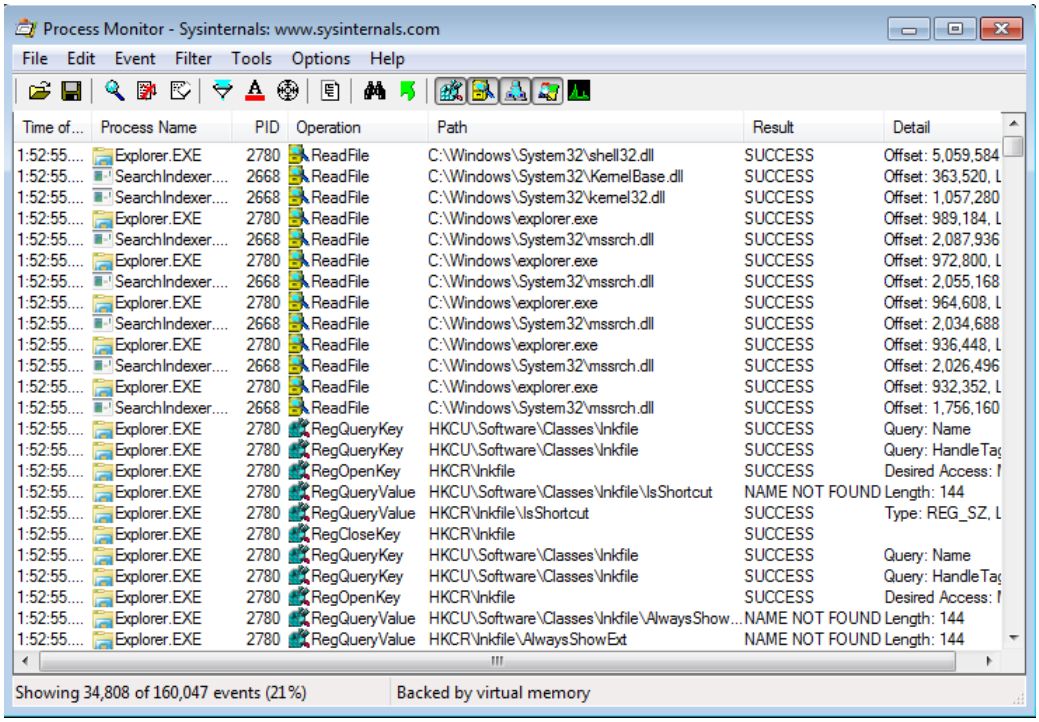
- IDA Pro [13]: è un disassembler comunemente usato per il reverse engineering. Supporta numerosi formati di file eseguibili per diversi processori e sistemi operativi. IDA fornisce strumenti semplificati per supportare le attività di reverse engineering, ottenendo informazioni sui riferimenti incrociati (XREFs) tra le varie sezioni, sui parametri delle chiamate API, ecc. Questo tool viene utilizzato anche nella fase di analisi dinamica poiché mette a disposizione un debugger.



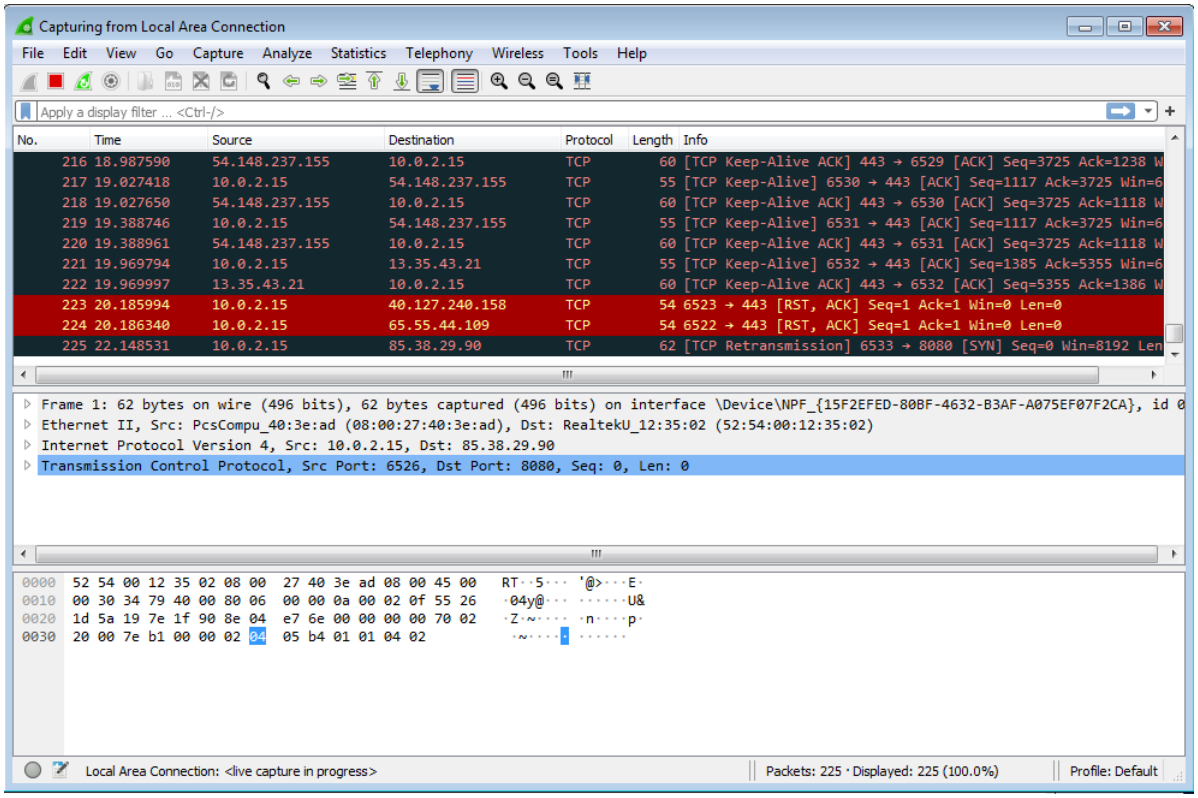
2.2 Analisi dinamica

L'analisi dinamica prevede l'esecuzione del malware e l'osservazione del suo comportamento sul sistema al fine di capire cosa fa. Prima di poter eseguire il malware in modo sicuro, è necessario configurare un ambiente che permetta di effettuare questo tipo di analisi, senza il rischio di danneggiare il sistema o la rete. L'analisi dinamica puó comprendere anche l'utilizzo di un debugger per esaminare lo stato in esecuzione del file dannoso. Questo tipo di analisi permette l'estrazione di ulteriori informazioni dettagliate sul file malevolo difficili da estrarre con altre tecniche. Alcuni tools utilizzati durante un'analisi dinamica possono essere i seguenti:

- Process Monitor [14]: è uno strumento di monitoraggio avanzato per Windows che mostra il file system in tempo reale, il registro e l'attività di processo/thread. Questo tool combina le funzionalità di due utility Sysinternals legacy, Filemon e Regmon, permette il filtraggio dei vari processi, proprietà complete di eventi come ID sessione e nomi utente, informazioni di processo affidabili, stack di thread completi con supporto simboli integrato per ogni operazione, registrazione simultanea su un file ecc.



- Wireshark [15]: è uno sniffer open source, uno strumento di acquisizione di pacchetti che intercetta e registra il traffico di rete. Wireshark fornisce la visualizzazione, l'analisi del flusso di pacchetti e l'analisi approfondita dei singoli pacchetti. Può essere utilizzato per analizzare le reti interne e l'utilizzo della rete, problemi di debug delle applicazioni e studio dei protocolli in azione. Ma può anche essere utilizzato per catturare password, fare reverse-engineer sui protocolli di rete, rubare i dati sensibili o informazioni.



Oltre a questi tools sono presenti anche dei sistemi automatici di analisi dinamica, chiamati Sandbox, che permettono l'esecuzione del sample su di una macchina in modo da monitorare dettagliatamente il comportamento del sample in analisi. Alcuni di questi sistemi sono:

- Cuckoo [16]: è un sistema di analsi del malware automatizzato, avanzato e modulare ed è open source. Di base permette di analizizare diverse tipologie di file malevoli (eseguibili, documenti di Office, file pdf, e-mail, ecc.). Traccia le chiamate API e il comportamento generale del file in analisi ed estrae informazioni e firme. Scarica ed analizza il traffico di rete, anche se crittografato con SSL/-

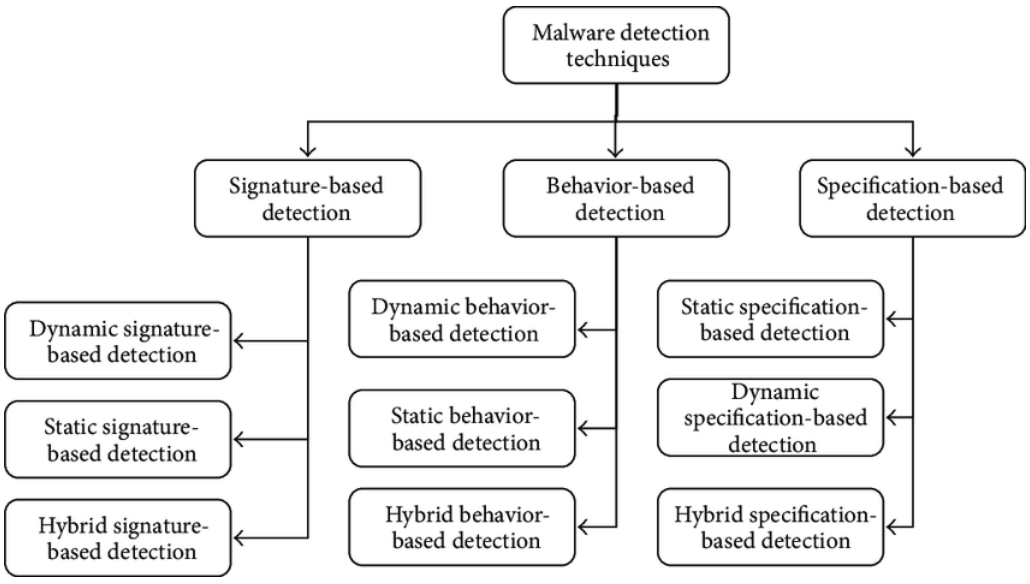
TLS. Permette anche di eseguire un analisi avanzata della memoria del sistema infettato.

- Any.run [17]: è un servizio online interattivo per l'analisi statica e dinamica del malware, è molto simile al servizio messo a disposizione con Cuckoo, l'unica differenza è che questo servizio ha una versione gratuita e una versione a pagamento completa di tutti i servizi necessari per un'analisi dettagliata.

3. Malware Detection e Machine Learning

3.1 Malware Detection

La Malware detection [18] comprende diverse tecniche che vengono utilizzate con lo scopo di rilevare il malware e prevenire l'infezione del sistema informatico, proteggendolo da potenziali perdite di informazioni o compromissione del sistema.



Di seguito vengono elencate alcune tecniche utilizzate per la detection dei malware [19]:

- Signature based detection: Il metodo più comune utilizzato nella malware detection è il rilevamento basato sulla firma. Una firma è una stringa di bit che identifica in modo univoco uno specifico virus. Il rilevamento basato sulla firma esegue la scansione dei file del computer e effettua un controllo incrociato del loro contenuto con un dizionario di firme dei virus. Con questo metodo il software

antivirus può mettere in quarantena, riparare o eliminare il file infetto e risolvere il problema del virus. Tuttavia, questa tecnica è in grado di identificare un piccolo sottoinsieme di minacce emergenti ma non è in grado di identificare nuovi virus.

- Anomaly Detection: è responsabile dell'osservazione di eventuali comportamenti insoliti o potenzialmente dannosi. Il rilevamento di anomalie può essere descritto come un allarme per strani comportamenti del sistema. Questo processo può essere eseguito in due fasi: la fase di addestramento e di rilevamento. Per la prima, il software è addestrato per apprendere il normale comportamento dell'host. Mentre per la seconda si utilizza il modello addestrato per riconoscere eventuali comportamenti imprevisti o insoliti che avvengono nel sistema. Uno dei vantaggi del rilevamento delle anomalie è che può prevedere il rilevamento di malware mai visti prima.

- Change Detection/Integrity Checking: Lo scopo principale del controllo di integrità è quello di rilevare eventuali cambiamenti nei file che potrebbero essere stati causati da qualche malware. Per raggiungere questo obiettivo, viene fatto l'hash di tutti i file presenti nel sistema, i quali vengono monitorati periodicamente. Se un file è stato modificato, l'hash calcolato non corrisponderà con quello precedente.

3.2 Machine Learning

La Machine learning [20] è un ramo dell'Informatica che si ricollega anche all'intelligenza artificiale (AI), essa si occupa di creare sistemi che apprendano o migliorino le performance in base ai dati che gli vengono dati da processare. In questo modo l'algoritmo puó comportarsi di conseguenza di fronte a situazioni mai riscontrate prima. Nell'ambito del malware detection un esempio potrebbe essere un nuovo sample di malware mai analizzato prima dove quindi le sue azioni sono sconosciute. Inizialmente era possibile e a volte lo è tutt'ora, utilizzare la firma del file come metodo di rilevazione se si tratta di un file malevolo o no. Purtroppo questo non è più sufficiente poiché gli attaccanti hanno iniziato a sviluppare malware capaci di modificare la loro signature utilizzando la tecnica del poliformismo, oppure utilizzando il packing, rendendo cosi inutile l'analisi citata sopra. Per affrontare queste nuove minacce è stato

necessario passare ad un'analisi più complessa, la quale può anche prevedere l'utilizzo della machine learning, andando così ad analizzare il comportamento del malware durante l'esecuzione, cercando dei pattern comportamentali malevoli: modifiche a file host, chiavi di registro e connessioni sospette. Il vantaggio principale di questo metodo è la capacitá di identificare se un sample è malevolo oppure no, ma anche, attraverso specifiche tecniche di clustering la famiglia a cui il malware potrebbe appartenere.

3.2.1 Come funziona?

Al fine di costruire un rilevatore di malware con la machine learning generalmente si ripercorrono i seguenti passaggi:

- Raccolta di sample di malware utilizzati per addestrare il sistema a riconoscere malware futuri.
- Estrazione di determinate caratteristiche per ogni sample in modo da rendere piú preciso il sistema di rilevamento.
- Addestramento del sistema di machine learning a riconoscere malware attraverso le funzionalitá estratte.
- Verifica tramite test su alcuni sample non presenti nel dataset originario di riferimento al fine di vedere se sono efficaci o meno.

Nella Machine learning sono presenti due algoritmi principali: gli algoritmi di machine learning supervisionati e non supervisionati, la loro differenza sta nel modo in cui essi si approcciano ai dati messi a loro disposizione per arrivare ad una determinata conclusione o previsione.

- La machine learning supervisionata: è una tecnica di apprendimento automatico che ha lo scopo di istruire un sistema informatico in modo da consentirgli di elaborare automaticamente previsioni o conclusioni rispetto ad un input, basandosi su di una serie di esempi inizialmente forniti costituiti da coppie di input e di output.
- La machine learning non supervisionata: è una tecnica di apprendimento automatico che consiste nel fornire al sistema informatico una serie di input che la

macchina riclassificherà ed organizzerà sulla base di caratteristiche comuni per cercare di effettuare conclusioni e previsioni sugli input successivi. Al contrario dell'apprendimento supervisionato, vengono forniti degli esempi non categorizzati dato che questo passaggio deve essere eseguito in modo autonomo dall'algoritmo.

3.2.2 Training

Per l'implementazione di un modello in grado di rilevare malware e file non dannosi è necessario in primo luogo raccogliere malware e campioni benigni in modo da disporre di diversi esempi per prevenire eventuali falsi positivi. Grazie alla comunità open source reperire questi sample è abbastanza semplice, il dataset di malware può essere reperito online o attraverso la piattaforma a pagamento Virus Total. In seguito si passa all'estrazione delle funzionalità, che dipendono dal formato del file che si sta utilizzando e dal tipo che si può ottenere.

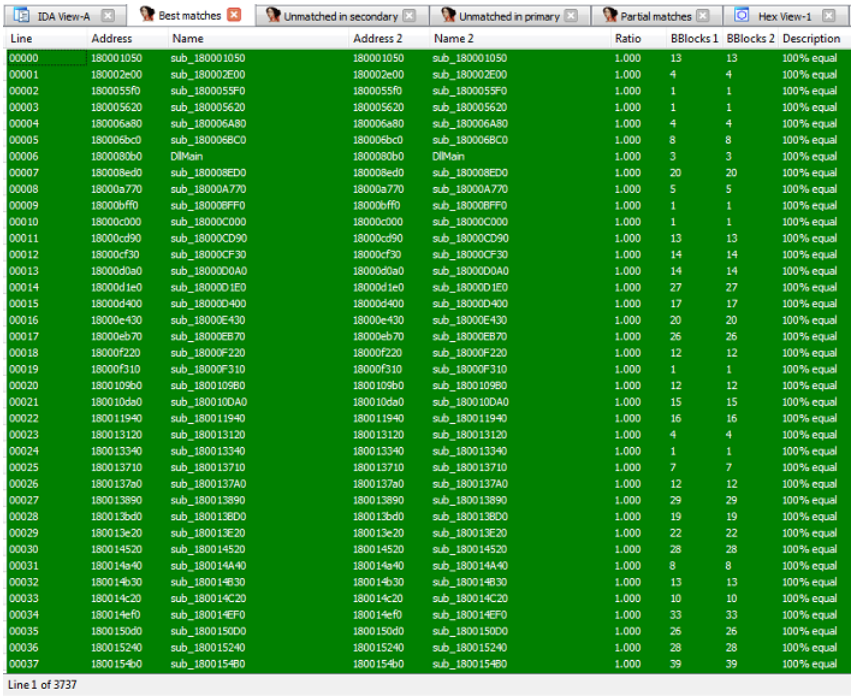
Infine si passa all'apprendimento automatico, dividendo il dataset in due parti (malevolo e non malevolo), uno per la formazione e l'altro che verrà utilizzato in seguito per addestrare il modello e testare quanto esso sia efficiente nel rilevare file dannosi. In base ai risultati ottenuti si valuterà se sia il caso di aggiungere funzionalità o toglierle.

4. Malware Attribution APT-38

Il lavoro svolto si è focalizzato sull'analisi della famiglia di malware denominata APT-38. Di questa sono stati analizzati cinque sample, al fine di estrarre le loro caratteristiche e scrivere una regola Yara per la detection di futuri malware. Durante l'analisi è stato cruciale l'utilizzo del plug-in Diaphora, il quale ha permesso il confronto di due sample al fine di trovare le funzioni con una percentuale maggiore di similarità in modo tale da poter evidenziare il codice che si ripete (code reuse) sulle varie versioni del malware.

4.1 Estrazione caratteristiche via analisi statica

Per poter attribuire un malware ad una determinata famiglia è necessario estrapolare le sue caratteristiche, ciò è possibile attraverso un'analisi statica del sample, in particolare attraverso l'utilizzo di tools come per esempio PEStudio, IDA Pro e il plugin Diaphora. Durante lo svolgimento di questo lavoro è stato necessario analizzare il codice assembly di cinque sample appartenenti alla stessa famiglia. Inizialmente sono state trovate 3700 funzioni all'interno di ogni singolo sample, per poter poi diminuire il numero di funzioni utili da studiare si è ricorso al plug-in per IDA Pro, Diaphora, che permette di confrontare due sample, in modo da trovare le funzioni simili tra loro.



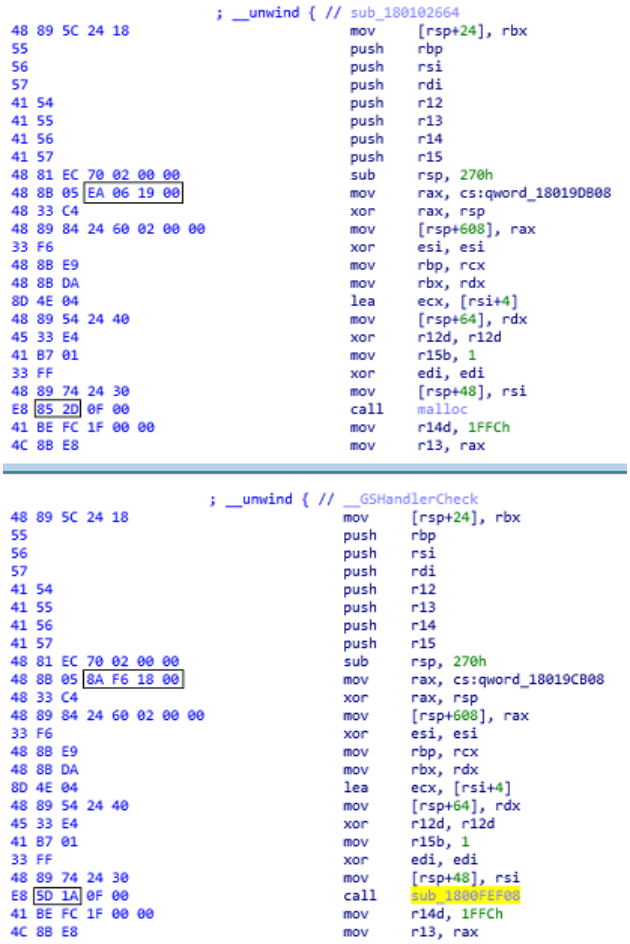
IDA View-A | Best matches | Unmatched in secondary | Unmatched in primary | Partial matches | Hex View-1

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00000	180001050	sub_180001050	180001050	sub_180001050	1.000	13	13	100% equal
00001	180002e00	sub_180002E00	180002e00	sub_180002E00	1.000	4	4	100% equal
00002	1800055f0	sub_1800055F0	1800055f0	sub_1800055F0	1.000	1	1	100% equal
00003	180005620	sub_180005620	180005620	sub_180005620	1.000	1	1	100% equal
00004	180006a80	sub_180006A80	180006a80	sub_180006A80	1.000	4	4	100% equal
00005	180006bc0	sub_180006BC0	180006bc0	sub_180006BC0	1.000	8	8	100% equal
00006	1800080b0	DllMain	1800080b0	DllMain	1.000	3	3	100% equal
00007	180008ed0	sub_180008ED0	180008ed0	sub_180008ED0	1.000	20	20	100% equal
00008	18000a770	sub_18000A770	18000a770	sub_18000A770	1.000	5	5	100% equal
00009	18000bff0	sub_18000BFF0	18000bff0	sub_18000BFF0	1.000	1	1	100% equal
00010	18000c000	sub_18000C000	18000c000	sub_18000C000	1.000	1	1	100% equal
00011	18000cd90	sub_18000CD90	18000cd90	sub_18000CD90	1.000	13	13	100% equal
00012	18000cf30	sub_18000CF30	18000cf30	sub_18000CF30	1.000	14	14	100% equal
00013	18000d0a0	sub_18000D0A0	18000d0a0	sub_18000D0A0	1.000	14	14	100% equal
00014	18000d1e0	sub_18000D1E0	18000d1e0	sub_18000D1E0	1.000	27	27	100% equal
00015	18000d400	sub_18000D400	18000d400	sub_18000D400	1.000	17	17	100% equal
00016	18000e430	sub_18000E430	18000e430	sub_18000E430	1.000	20	20	100% equal
00017	18000eb70	sub_18000EB70	18000eb70	sub_18000EB70	1.000	26	26	100% equal
00018	18000f220	sub_18000F220	18000f220	sub_18000F220	1.000	12	12	100% equal
00019	18000f310	sub_18000F310	18000f310	sub_18000F310	1.000	1	1	100% equal
00020	1800109b0	sub_1800109B0	1800109b0	sub_1800109B0	1.000	12	12	100% equal
00021	180010da0	sub_180010DA0	180010da0	sub_180010DA0	1.000	15	15	100% equal
00022	180011940	sub_180011940	180011940	sub_180011940	1.000	16	16	100% equal
00023	180013120	sub_180013120	180013120	sub_180013120	1.000	4	4	100% equal
00024	180013340	sub_180013340	180013340	sub_180013340	1.000	1	1	100% equal
00025	180013710	sub_180013710	180013710	sub_180013710	1.000	7	7	100% equal
00026	1800137a0	sub_1800137A0	1800137a0	sub_1800137A0	1.000	12	12	100% equal
00027	180013890	sub_180013890	180013890	sub_180013890	1.000	29	29	100% equal
00028	180013bd0	sub_180013BD0	180013bd0	sub_180013BD0	1.000	19	19	100% equal
00029	180013e20	sub_180013E20	180013e20	sub_180013E20	1.000	22	22	100% equal
00030	180014520	sub_180014520	180014520	sub_180014520	1.000	28	28	100% equal
00031	180014a40	sub_180014A40	180014a40	sub_180014A40	1.000	8	8	100% equal
00032	180014b30	sub_180014B30	180014b30	sub_180014B30	1.000	13	13	100% equal
00033	180014c20	sub_180014C20	180014c20	sub_180014C20	1.000	10	10	100% equal
00034	180014ef0	sub_180014EF0	180014ef0	sub_180014EF0	1.000	33	33	100% equal
00035	1800150d0	sub_1800150D0	1800150d0	sub_1800150D0	1.000	26	26	100% equal
00036	180015240	sub_180015240	180015240	sub_180015240	1.000	28	28	100% equal
00037	1800154b0	sub_1800154B0	1800154b0	sub_1800154B0	1.000	39	39	100% equal

Line 1 of 3737

Nella figura sopra possiamo vedere l'output di Diaphora durante il confronto tra il primo e il secondo sample, dove sono state estratte le 126 funzioni che hanno in comune. Successivamente ad un analisi dettagliata di ogniuna di queste, è stata verificata la loro presenza anche negli altri sample. Questo riutilizzo della stessa porzione di codice viene chiamato Code Reuse.

Nella figura riportata di seguito è illustrata una funzione che è presente in tutti i sample analizzati e sono inoltre evidenziate le parti di opcode che differiscono tra i due sample in analisi.



Queste parti di opcode che differiscono tra loro sono determinanti nello scrivere una buona regola YARA che possa identificare più sample possibili, poiché quei caratteri che differiscono tra le due funzioni potrebbero cambiare in più versioni dello stesso malware.

La scelta su quale funzione utilizzare per la regola YARA viene fatta in base alla tipologia del codice identificato: potrebbe trattarsi di un metodo di cifratura dei dati, come questi vengono salvati in memoria o come vengono effettuate determinate comunicazioni esterne. Una volta estatta una caratteristica determinante all'interno del sample, è necessario scrivere una regola YARA in grado di identificarla anche quando riutilizzata su varianti della stessa famiglia.

Durante la stesura della regola YARA viene fatto un confronto della stessa funzione

tra i sample in modo da poter posizionare al meglio le varie wildcard. Le wildcard sono caratteri che vanno a sostituire i byte che potrebbero variare da sample a sample nella funzione che si sta studiando. Nel linguaggio YARA vengono utilizzati i caratteri "??" per identificare un qualsiasi valore di tipo byte.

Il corretto posizionamento delle varie wildcard durante la scrittura della regola Yara è un passo cruciale perchè un posizionamento errato potrebbe portare alla rilevazione di falsi positivi.

Andando ad analizzare la funzione riportata di seguito possiamo notare un corretto posizionamento delle wildcard con conseguente rilevazione di tutti e cinque i sample in analisi.

```
48 89 5C 24 10                  mov     [rsp+16], rbx
48 89 74 24 18                  mov     [rsp+24], rsi
57                              push    rdi
48 81 EC 50 01 00 00            sub     rsp, 336
48 8B 05 4F CD 18 00            mov     rax, cs:qword_18019DB08
48 33 C4                        xor     rax, rsp
48 89 84 24 40 01 00 00         mov     [rsp+320], rax
45 33 C0                        xor     r8d, r8d        ; protocol
8B D9                           mov     ebx, ecx
BE 02 00 00 00                  mov     esi, 2
41 8D 50 01                     lea     edx, [r8+1]     ; type
8B CE                           mov     ecx, esi        ; af
FF 15 56 D7 10 00               call    cs:socket
48 8B F8                        mov     rdi, rax
33 C0                           xor     eax, eax
48 89 44 24 20                  mov     [rsp+32], rax
48 89 44 24 28                  mov     [rsp+40], rax
0F B6 05 00 4A 19 00            movzx   eax, cs:name
66 89 74 24 20                  mov     [rsp+32], si
3C 30                           cmp     al, 48
7C 0D                           jl      short loc_180010E06
3C 39                           cmp     al, 57
7F 09                           jg      short loc_180010E06
48 8D 0D EC 49 19 00            lea     rcx, name
EB 5E                           jmp     short loc_180010E64
```

Questa funzione si occupa di aprire una connessione socket. Al fine di creare una buona regola Yara in grado di rilevare meno falsi positivi possibili è stato necessario posizionare le wildcard ai byte che potrebbero variare in altre varianti.

```
rule apt_38{
        meta:
                actor = "APT-38"
                author = "Federico Casenove"
        strings:
                $xor_socket_string = {
                        48 89 5C 24 ??
                        48 89 74 24 ??
                        57
                        48 81 EC ?? ?? ?? ??
                        48 8B 05 ?? ?? ?? ??
                        48 33 C4
```

```
                48 89 84 24 ?? ?? ?? ??
                45 33 C0
                8B D9
                BE ?? ?? ?? ??
                41 8D 50 ??
                8B CE
                FF 15 ?? ?? ?? ??
                48 8B F8
                33 C0
                48 89 44 24 ??
                48 89 44 24 ??
                0F B6 05 ?? ?? ?? ??
                66 89 74 24 ??
                3C ??
                7C ??
                3C ??
                7F ??
                48 8D 0D ?? ?? ?? ??
                EB 5E
        }
    condition:
        $xor_socket_string
}
```

```
C:\Users\BlackHole\Desktop>.\yara64.exe -s -r \\VBOXSVR\telsy_folder\fun_example.yar "C:\Users\BlackHole\Desktop\malware
 yara test"
apt_38 C:\Users\BlackHole\Desktop\malware yara test\644db11f28243b026d9858dc9560b248a015bed8eb61ec5f43c10e4e551013b6.dll
.bin
0xf130:$xor_socket_string: 48 89 5C 24 10 48 89 74 24 18 57 48 81 EC 50 01 00 00 48 8B 05 BF AD 18 00 48 33 C4 48 89 84
24 ...
apt_38 C:\Users\BlackHole\Desktop\malware yara test\3df7d613434b4b66295b0b83cc2bd1fce1b13661099a1df59a4b900e87ca0b14.dll
.bin
0x101a0:$xor_socket_string: 48 89 5C 24 10 48 89 74 24 18 57 48 81 EC 50 01 00 00 48 8B 05 4F CD 18 00 48 33 C4 48 89 84
 24 ...
apt_38 C:\Users\BlackHole\Desktop\malware yara test\a17e6634ddbb192ae03a11603c44687dfc99d8968b66057487761a601fa2b159.dll
.bin
0x10210:$xor_socket_string: 48 89 5C 24 10 48 89 74 24 18 57 48 81 EC 50 01 00 00 48 8B 05 DF BC 18 00 48 33 C4 48 89 84
 24 ...
apt_38 C:\Users\BlackHole\Desktop\malware yara test\927ec6575482f38b5c832aea665fe4d17c79ad9cac78e563f660ab1c858496c8.dll
.bin
0x101a0:$xor_socket_string: 48 89 5C 24 10 48 89 74 24 18 57 48 81 EC 50 01 00 00 48 8B 05 4F CD 18 00 48 33 C4 48 89 84
 24 ...
apt_38 C:\Users\BlackHole\Desktop\malware yara test\209d6c39e187e53339b77571f47baa9e1b9b9630c19f7631166805542127 32b6.dll
.bin
0x101a0:$xor_socket_string: 48 89 5C 24 10 48 89 74 24 18 57 48 81 EC 50 01 00 00 48 8B 05 4F CD 18 00 48 33 C4 48 89 84
 24 ...

C:\Users\BlackHole\Desktop>_
```

Mentre andando a posizionare le wildcard in modo errato possiamo notare che gli stessi sample utilizzati sopra non vengono rilevati con questa regola Yara. Le wildcard che sono state posizionate erroneamente sono quelle presenti nella chiamata alla funzione socket.

```
rule apt_38{
        meta:
                actor = "APT-38"
                author = "Federico Casenove"
        strings:
                $xor_socket_string = {
                        48 89 5C 24 ??
                        48 89 74 24 ??
                        57
                        48 81 EC ?? ?? ?? ??
                        48 8B 05 ?? ?? ?? ??
                        48 33 C4
                        48 89 84 24 ?? ?? ?? ??
                        45 33 C0
                        8B D9
                        BE ?? ?? ?? ??
                        41 8D 50 ??
                        8B CE
                        FF 15 56 D7 ?? ??
                        48 8B F8
                        33 C0
                        48 89 44 24 ??
                        48 89 44 24 ??
                        0F B6 05 ?? ?? ?? ??
                        66 89 74 24 ??
                        3C ??
                        7C ??
                        3C ??
                        7F ??
                        48 8D 0D ?? ?? ?? ??
                        EB 5E
                }
        condition:
                $xor_socket_string
}
```

```
C:\Users\BlackHole\Desktop>.\yara64.exe -s -r \\VBOXSVR\telsy_folder\fun_example.yar "C:\Users\BlackHole\Desktop\malware
 yara test"
apt_38 C:\Users\BlackHole\Desktop\malware yara test\209d6c39e187e53339b77571f47baa9e1b9b9630c19f763116680554212732b6.dll
.bin
0x101a0:$xor_socket_string: 48 89 5C 24 10 48 89 74 24 18 57 48 81 EC 50 01 00 00 48 8B 05 4F CD 18 00 48 33 C4 48 89 84
 24 ...
apt_38 C:\Users\BlackHole\Desktop\malware yara test\3df7d613434b4b66295b0b83cc2bd1fce1b13661099a1df59a4b900e87ca0b14.dll
.bin
0x101a0:$xor_socket_string: 48 89 5C 24 10 48 89 74 24 18 57 48 81 EC 50 01 00 00 48 8B 05 4F CD 18 00 48 33 C4 48 89 84
 24 ...
apt_38 C:\Users\BlackHole\Desktop\malware yara test\927ec6575482f38b5c832aea665fe4d17c79ad9cac78e563f660ab1c858496c8.dll
.bin
0x101a0:$xor_socket_string: 48 89 5C 24 10 48 89 74 24 18 57 48 81 EC 50 01 00 00 48 8B 05 4F CD 18 00 48 33 C4 48 89 84
 24 ...

C:\Users\BlackHole\Desktop>
```

Come possiamo vedere dall'output, utilizzando la stessa regola ma con due wildcard posizionate erroneamente sono stati rilevati solo tre sample su cinque.

4.2 Regola YARA

Una volta estratte le funzioni più importanti che si ripresentavano nei vari sample, sono state studiate in modo da capire se potevano essere valide per la scrittura di una regola Yara. Sono state prese in esame quattro funzioni e da queste è stata scritta un'unica regola Yara, come riportato di seguito:

```
rule apt_38{
        meta:
                actor = "APT-38"
                author = "Federico Casenove"
        strings:
                $malloc_string = {
                        48 89 5C 24 ??
                        48 89 6C 24 ??
                        48 89 74 24 ??
                        57
                        41 54
                        41 55
                        48 83 EC ??
                        8B 5C 24 ??
                        49 63 F9
                        49 63 E8
                        4C 8B E2
                        4C 8B E9
                        85 DB
                        75 ??
                        48 8B CF
                        E8 ?? ?? 0F 00
                        49 8D 14 2C
                        4C 8B C7
                        48 8B C8
                        48 8B F0
                        E8 ?? ?? 0F 00
                        44 8B 44 24 ??
                        41 83 ?? ??
                        74 ??
                        80 3D ?? ?? ?? ?? ??
                        75 ??
                        4C 8B ?? ?? ?? ?? ??
                        8B D7
                        48 8B CE
                        E8 ?? ?? ?? ??
                }
                $xor_malloc_string = {
                        48 89 5C 24 ??
                        55
```

```
        56
        57
        41 54
        41 55
        41 56
        41 57
        48 81 EC ?? ?? ?? ??
        48 8B 05 ?? ?? ?? ??
        48 33 C4
        48 89 84 24 ?? ?? ?? ??
        33 F6
        48 8B E9
        48 8B DA
        8D 4E ??
        48 89 54 24 ??
        45 33 E4
        41 B7 ??
        33 FF
        48 89 74 24 ??
        E8 ?? ?? 0F 00
        41 BE ?? ?? ?? ??
        4C 8B E8
}
$create_thread = {
        48 89 5C 24 ??
        57
        48 83 EC ??
        48 8B 41 ??
        33 FF
        48 8B D9
        48 89 41 ??
        48 8B 41 ??
        4C 8B C9
        48 89 41 ??
        4C 8D ?? ?? ?? ?? ??
        33 D2
        33 C9
        48 89 7C 24 ??
        89 7C 24 ??
        FF 15 ?? ?? ?? ??
        48 89 7B ??
        48 89 7B ??
        48 8B 5C 24 ??
        48 83 C4 ??
        5F
        C3
}
$xor_socket_string = {
        48 89 5C 24 ??
```

```
                48 89 74 24 ??
                57
                48 81 EC ?? ?? ?? ??
                48 8B 05 ?? ?? ?? ??
                48 33 C4
                48 89 84 24 ?? ?? ?? ??
                45 33 C0
                8B D9
                BE ?? ?? ?? ??
                41 8D 50 ??
                8B CE
                FF 15 ?? ?? ?? ??
                48 8B F8
                33 C0
                48 89 44 24 ??
                48 89 44 24 ??
                0F B6 05 ?? ?? ?? ??
                66 89 74 24 ??
                3C ??
                7C ??
                3C ??
                7F ??
                48 8D 0D ?? ?? ?? ??
                EB 5E
        }
    condition:
        any of them
}
```

In questa regola Yara le prime due stringhe descrivono funzioni che si occupano di allocazione di memoria, la terza stringa è una funzione che si occupa di creare un thread e l'ultima è una funzione che si occupa di aprire una connessione tramite socket. La condizione fa si che nel momento in cui vengono trovate una o più stringhe di quelle citate sopra la regola classifica il malware come possibile APT-38. In ogni stringa sono state posizionate le varie wildcard al fine di poter rilevare meno falsi positivi possibili. Prima di mandare in produzione la regola Yara è stata testata sui cinque sample studiati ed è stato poi eseguito un ulteriore test lanciando questa regola su tutto il disco della macchina, in modo da poter essere sicuri della sua correttezza. Un'operazione che non è stata effettuata in questa regola è la "Rule splitting". Essa consiste nel dividere la regola in sottoregole più specifiche, sia dal punto di vista di descrizione che di stringhe, in modo tale che nella fase di detection sarà possibile capire di preciso di quale caratteristica e di quale famiglia si sta parlando. Mentre mantenendo più stringhe nella stessa regola e categorizzarla come "APT-38", rende la rilevazione molto generale.

4.3 Ciclo vita di una hunting rule

Al fine di testare la regola Yara scritta fin'ora, la società Telsy ha messo a disposizione la sua piattaforma proprietaria di Cyber Threat Intelligence, questa riceve giornalmente uno stream di malware provenienti da diverse sorgenti, tra cui VirusTotal e processa dai 2000 ai 15000 sample. I sample vengono elaborati uno alla volta in modo statico e dinamico e poi classificati attraverso algoritmi di machine learning e regole Yara.
Prima di caricare definitivamente la regola Yara all'interno della piattaforma deve superare una prima fase di test, chiamata fase di pre-produzione, per essere poi deployata nella fase finale chiamata di produzione. Queste due fasi si definiscono come segue:

- Fase di pre-produzione: in questa fase la regola Yara viene testata con tre tipologie di dataset di malware. Un dataset comprende dei sample di malware, simili a quelli analizzati ma non della stessa famiglia, un altro dataset comprende dei goodware, come ad esempio file di sistema e l'ultimo è composto da 500.000 sample di malware di diverse tipologie, questo per controllare se magari la regola possa matchare dei falsi positivi o sample non rilevati in passato facenti parte della famiglia in analisi. Se la regola è idonea a questi test allora può essere presa in considerazione per passare alla fase di produzione.

- Fase di produzione: la regola Yara, una volta superati i test della fase precedente, viene caricata all'interno della piattaforma dove potrà essere utilizzata per analizzare i sample di malware che si troverà di fronte.

La regola Yara scritta e citata precedentemente, è riuscita a passare tutti i test ed entrare in fase di produzione, uscendo dalla fase di pre-produzione con un rateo di falsi positivi:

$$fp < 1/500.000^{1}$$

Dalla sua messa in produzione la regola Yara è stata capace di rilevare due nuovi sample mai analizzati prima d'ora, andandoli così a categorizzare come APT-38.

[1]Il risultato è stato testato su di un dataset di 500.000 sample non facenti parte della famiglia APT-38, portando alla rilevazione di 0 falsi positivi

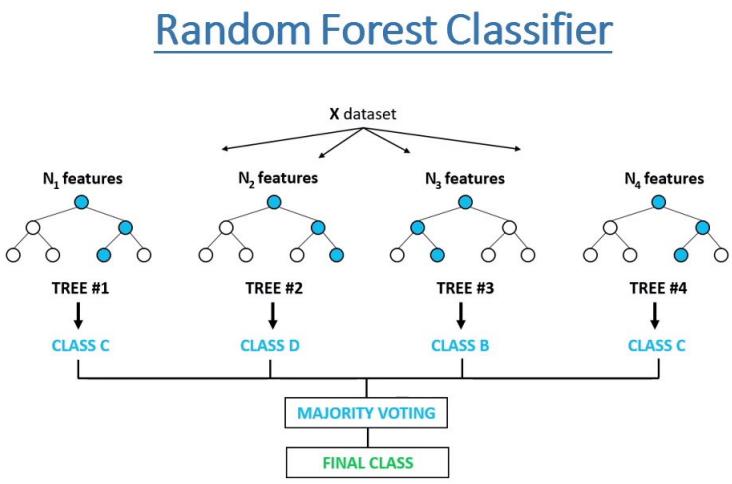
4.4 Machine learning per la malware attribution

Il processo di detection e attribution di un malware può essere automatizzato attraverso l'uso della Machine learning, che può essere utilizzata sia per la classica detection di file malevoli che per l'attribuzione di malware a determinate famiglie. Questo risultato però passa attraverso uno dei due processi di training elencati di seguito:

- Il processo generale: vengono presi due dataset, uno contenente dei goodware e l'altro dei malware, da ognuno di essi vengono estratte delle caratteristiche come l'entropia di un sample, i nomi delle sezioni, il numero delle sezioni, ecc.. Una volta estratte queste caratteristiche l'algoritmo riuscirà a distinguere quando si tratta di un file malevolo o no.

- Il processo per la Malware Attribution: come nel processo generale, vengono considerati i due dataset, di goodware e di malware, arricchiti ora con le informazioni della famiglia, provenienti dalle regole Yara. In questo modo un sample non solo viene categorizzato in goodware o malware ma anche nella specifica categoria di malware.

Non è possibile scendere in dettaglio nei processi di elaborazione che utilizzano algoritmi di machine learning per la malware detection in quanto questi sono sperimentali e appartengono alle aziende di Cyber Threat Intelligence che ne fanno uso, queste li costudiscono come propria proprietà intellettuale.
È noto però che uno degli algoritmi di machine learning più utilizzati, come per il caso della società Telsy, ma anche nella letteratura scientifica come ad esempio Garcia et al. [21], sia l'algoritmo di Random Forest.



Questo si basa sul concetto di alberi decisionali con un approccio top-down partendo quindi dalla radice, i quali vengono estratti in base alle varie esecuzioni dei sample appartenenti ad un sottoinsieme del dataset totale.

5. Conclusioni

La Cyber Threat Intelligence e in particolare la Malware Family Attribution è un argomento importante nell'ambito della Cyber Security. In letteratura e in commercio esistono diversi approcci per assolvere questo compito uno dei più utilizzati vi è quello dell'analisi statica in combinazione con le regole Yara per la detection di malware. In particolare in questo lavoro si è andati alla ricerca del code reuse che una famiglia di malware tende ad avere durante la propria evoluzione. È stato utilizzato il plugin Diaphora per il software IDA Pro, per poter identificare il code reuse presente tra i cinque sample analizzati, appartenenti alla famiglia APT-38. Una volta estratte le caratteristiche presenti nei sample è stato possibile scrivere una regola Yara, capace di rilevare un numero di falsi positivi pari a:

$$fp < 1/500.000.$$

Questo rateo è stato estratto da un test effettuato su di un dataset contenente 500.000 sample, nessuno dei quali faceva parte della famiglia in analisi.
Dalla sua messa in produzione la regola Yara è stata capace fin ora di rilverare due sample mai riscontrati prima d'ora, andandoli a categorizzare come "APT-38".
Questo lavoro è stato svolto in particolare per la detection della famiglia di malware denominata "APT-38", ma esso può essere svolto per qualsiasi altra famiglia, vecchia o nuova che sia, sempre che venga utilizzato del code reuse, andando così a creare una piattaforma di detection capace di poter fronteggiare qualsiasi nuovo sample e poter fornire informazioni sempre più dettagliate e utili all'analista.

Bibliografia

[1] Nour-Eddine Lakhdari, Amine Boukhtouta e Mourad Debbabi. ≪Inferring Malware Family through Application Protocol Sequences Signature≫. In: apr. 2014. DOI: 10.1109/NTMS.2014.6814026.

[2] Liu et al. ≪Automatic malware classification and new malware detection using machine learning≫. In: *Frontiers of Information Technology & Electronic Engineering* (2017).

[3] *Diaphora*. URL: https://github.com/joxeankoret/diaphora.

[4] Md Abu et al. ≪An Enhancement of Cyber Threat Intelligence Framework≫. In: *Journal of Advanced Research in Dynamical and Control Systems* 10 (nov. 2018), pp. 96–104.

[5] *Cyber Threat Information Sharing*. URL: https://www.hackinbo.it/slides/1508354164_SCHIFILLITI_Slides_HackinBO2017_DraftVer05.pdf.

[6] *Malware Family Document*. URL: https://docs.google.com/spreadsheets/d/1H9_xaxQHpWaa4O_Son4Gx0YOIzlcBWMsdvePFX68EKU/pubhtml#.

[7] *Cyber Threat Hunting*. URL: https://www.ictsecuritymagazine.com/articoli/cyber-threat-intelligence-hunting-maggiore-efficacia-al-contrasto-degli-attacchi-informatici/.

[8] *YARA*. URL: https://yara.readthedocs.io/en/stable/.

[9] Andrew Honig Michael Sikorski. *Practical Malware Analysis, The Hands-On Guide to Dissecting Malicious Software*. 38 Ringold Street, San Francisco, CA 94103: William Pollock, 2012. ISBN: 1-59327-290-1.

[10] *PEStudio*. URL: https://www.winitor.com/.

[11] *Virus Total*. URL: https://www.virustotal.com/gui/home/upload.

[12] *Resource Hacker*. URL: http://www.angusj.com/resourcehacker/.

[13] *IDA Pro - Hex Rays*. URL: https://www.hex-rays.com/products/ida/.

[14] *Process Monitor*. URL: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon.

[15] *WireShark*. URL: https://www.wireshark.org/.

[16] *Cuckoo Sandbox*. URL: https://cuckoosandbox.org/.

[17] *Any.run*. URL: https://any.run/.

[18] Om Samantray, Satya Tripathy e Susant Das. «A Theoretical Feature-wise Study of Malware Detection Techniques». In: *International Journal of Computer Sciences and Engineering* 6 (dic. 2018), pp. 879–887. DOI: 10.26438/ijcse/v6i12.879887.

[19] Swathi Pai. «A Comparison of Clustering Techniques for Malware Analysis». In: (mag. 2015).

[20] Nitish A e Hanumanthappa J. «Deep Learning for Malware Analysis: A Review». In: mag. 2019.

[21] Garcia et al. «Random Forest for Malware Classification». In: (set. 2016).