POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Studio e presentazione di un framework di Memory Forensics a supporto del rilevamento e l'analisi comportamentale di Stealthy Malware

Relatori
prof. Antonio Lioy
prof. Andrea Atzeni

Candidato
Marco Conti

Supervisore Aziendale
Dott. Claudio Paganelli

Anno accademico 2020-2022

Indice

Capitolo 1

Introduzione

La rivoluzione digitale, avviata negli Stati Uniti durante gli anni Sessanta, è sicuramente il punto di origine più rappresentativo nella storia per focalizzare il periodo di nascita e sviluppo della "cultura hacker", che ha distinto una selettiva classe di individui appassionati da sfide intellettuali e dal desiderio di superare i limiti imposti dai sistemi software e, proseguendo, dalla società. Questa comunità è composta da individui, la cui identità è spesso celata dietro a soprannomi, o "nickname", i cui principi si basano su un forte senso etico, semplicemente riassumibile nei valori di "condivisione" e "libertà", senza vincoli di età, razza o situazione. Nel corso della propria evoluzione, il tempo ha forse giocato a sfavore della comunità "hacker", legandone sempre più il termine ad ideali negativi. Attualmente, si pensa puramente agli "hacker" come soggetti spinti unicamente dal guadagno, senza alcun senso etico-morale o preoccupazione di recare danno ad altri individui per finalizzare i propri obiettivi. Seppur non bisogna far di tutta l'erba un fascio ci troviamo complessivamente lontani dalle motivazioni, al limite dei concetti di "libertà" e "giustizia", di entrar a far parte di una "Hall Of Fame", di mostrarsi al mondo come dei "Robin Hood" digitali. In questo scenario si sviluppa la necessità di esperti di Digital Forensics, analisti in grado di investigare sistemi attaccati digitalmente, al fine di arrivare al diretto colpevole e passare il testimone alle autorità giudiziarie. Questo percorso di tesi, reso possibile da Intellera Consulting, si focalizza sullo studio delle tecniche di analisi forense con l'obiettivo di proporre un framework per la rilevazione e analisi di malware, in particolar modo i corrispettivi di tipo "stealth", mediante l'esamina della memoria volatile di un sistema informatico. Il flusso della discussione si avvia con una descrizione sulle basi della Digital Forensics e del processo forense, mantenendo volutamente un vocabolario di alto livello, per rendere la lettura accessibile anche ad individui non puramente "tecnici". Si procede poi nella descrizione dello stato dell'arte delle tecniche di Memory Forensics, branca della Digital Forensics che pone l'attenzione nell'analisi di un'immagine di memoria RAM, contenente evidenze digitali che non possono essere recuperate altrove e il cui effimero ciclo di vita termina al momento dello spegnimento del sistema o della sovrascrittura della relativa pagina di memoria. Avanzando, ci si indentra in una dettagliata esposizione inerente i malware di tipo stealth e le tecniche che li rendono in grado di occultarsi nei sistemi infettati, senza rilasciare file sul disco, e persistere anche successivamente allo spegnimento del dispositivo. Vengono così descritti alcuni dei più interessanti framework rilasciati recentemente in letteratura per la rilevazione di stealthy malware tramite metodi di Digital Forensics e, in particolar modo, Memory Forensics. In risposta ad essi, viene presentato un framework per la rilevazione e analisi comportamentale di malware di tipo stealth, e non, focalizzato sulle procedure di Memory Forensics. Un valore aggiunto è rappresentato dall'ausilio di tecniche di Static Analysis, Dynamic Analysis, Network Analysis e OSINT, con i relativi strumenti selezionati allo scopo di fornire un'ampia visione dei movimenti del malware in analisi. Il framework è stato, infine, testato su tre distinti malware, seguendo con precisione le fasi del processo forense pubblicate dal NIST.

Capitolo 2

Digital Forensics

La Digital Forensics è un ramo della scienza Forense che emerge in risposta all'incremento di crimini eseguiti tramite l'utilizzo di computer e sistemi informatici, con lo scopo di collezionare e analizzare le informazioni digitali prodotte, memorizzate e trasmesse nell'era digitale ed utilizzarle come sorgenti di prova in un processo penale o civile. La scienza forense digitale esiste da quando i computer memorizzano dati che possono essere utilizzati come prove e, come solitamente accade nel settore informatico, nasce in ambienti governativi per trovare successivamente posto nel settore commerciale. La prima applicazione risale al 1984, quando nei laboratori del FBI e di altre agenzie di polizia americane si iniziò a sviluppare programmi e procedure per l'esamina di evidenze informatiche. Tra le diverse definizioni presenti in letteratura, volte a dettagliare il processo e i fini principali del settore forense digitale, R. Kaur [1] fornisce una delle più complete:

"La Digital Forensics può essere definita come l'uso di metodi scientificamente derivati e comprovati per la conservazione, la convalida, l'identificazione, l'analisi, l'interpretazione, la documentazione e la presentazione di prove digitali derivate da fonti digitali allo scopo di facilitare o favorire la ricostruzione di eventi ritenuti criminali, o di aiutare ad anticipare azioni non autorizzate che si dimostrano dirompenti per le operazioni pianificate".

Le pratiche forensi trovano luogo in un mondo sempre più scosso dai "cyber-attack" o "computer crime", definiti dal United States Department of Justice (DOJ) come "qualsiasi violazione del diritto penale che implichi la conoscenza di tecnologie informatiche per la loro perpetrazione, indagine o azione legale". Il Check Point Security Report 2021 [2], atto a esporre annualmente i più recenti rischi affrontati dalle Organizzazioni mondiali in ambito informatico, registra un aumento globale degli attacchi alle organizzazioni nel 2020 pari al 29% rispetto all'anno precedente con una netta preferenza per gli attacchi ransomware, aumentati del 93% con l'emergere del "Triple Extortion Ransomware".

La Figura 2.1 mostra le statistiche circa l'attuazione delle principali categorie di attacchi informatici prima su scala mondiale, poi sulle regioni economico-industriali di "Americas", "EMEA" e "APAC". La Figura 2.2 raffigura l'attuale "Global Threat Index Map", in grado di fornire la probabilità che un sistema informatico in un determinato paese possa essere vittima di un'infezione da malware.

Relativamente ai crimini informatici M. Al-Mousa [3] individua tre gruppi, a seconda della tipologia del target per cui sono pensati:

- **Individuo**: viene attaccato tramite diffusione di materiale osceno, il cyber-stalking, la diffamazione, l'esposizione indecente, l'imbroglio, il controllo non autorizzato, l'email spoofing e la frode.
- **Organizzazioni e Governi**: sono compromessi per via dell'accesso non autorizzato, del terrorismo informatico, del possesso di informazioni non autorizzate, della distribuzione di software pirata.
- **Proprietà**: esposta al vandalismo informatico, alla trasmissione di virus, all'accesso non autorizzato, ai reati contro la proprietà intellettuale e ai furti su Internet

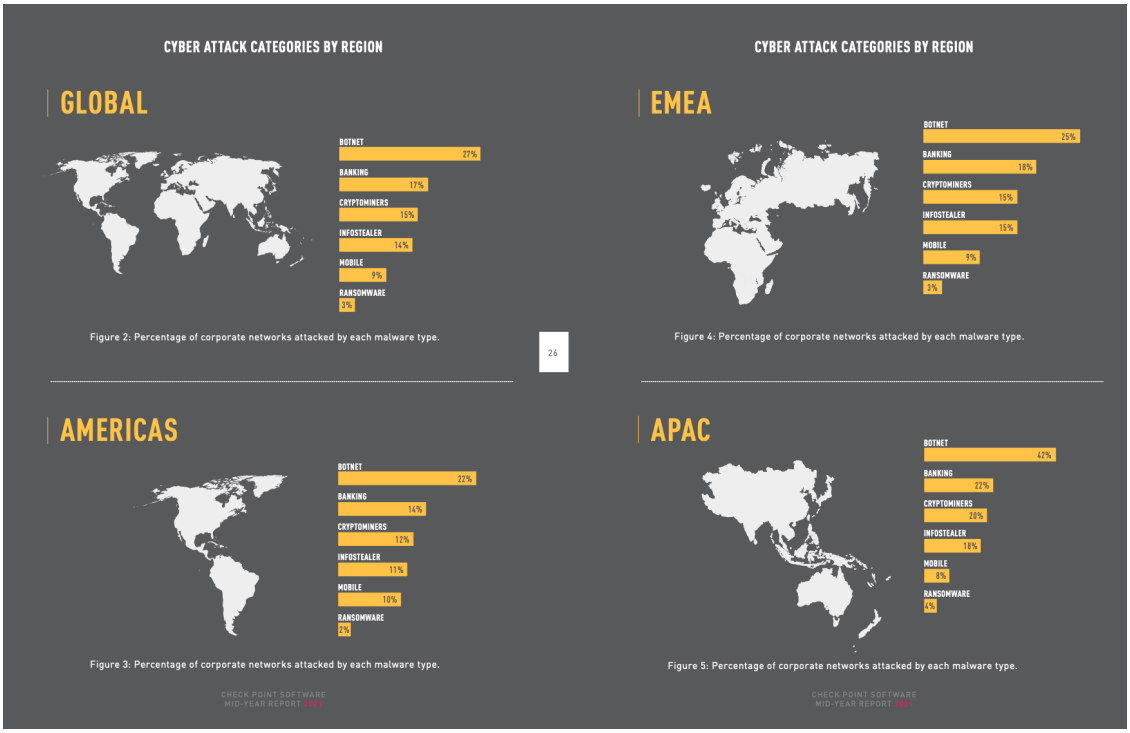


Figura 2.1. Categorie di attacco informatico per regione, CheckPoint Mid-Year Report 2021

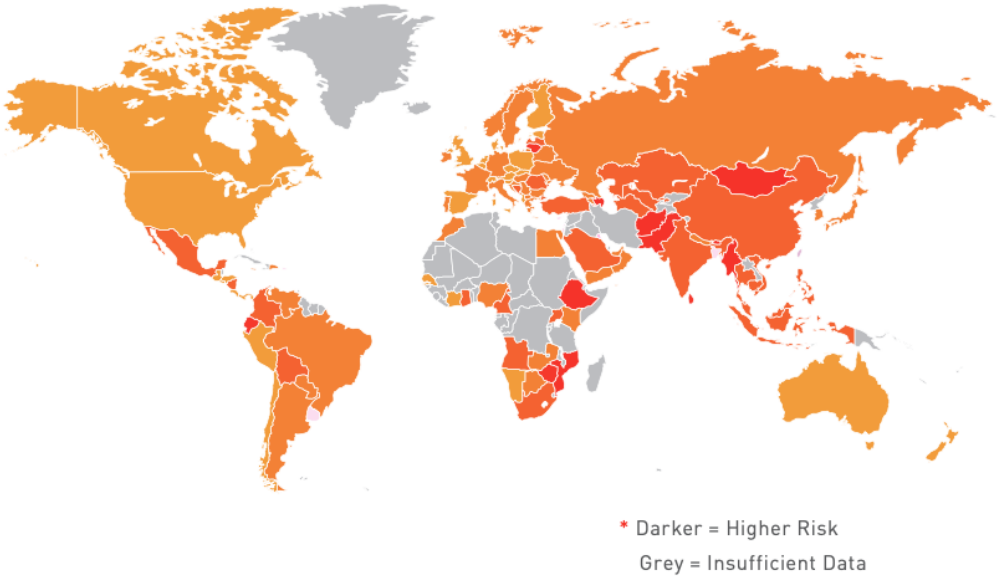


Figura 2.2. Mappa dell'indice di minaccia globale, CheckPoint Mid-Year Report 2021

Tra i sopra citati, M. Al-Mousa [3] identifica come cyber-crime più pericolosi quelli che attaccano la società, come la pedo-pornografia, l'esposizione indecente, l'inquinamento dei giovani, i crimini finanziari, la vendita di articoli illegali, il traffico di droga, la falsificazione, il gioco d'azzardo online. La distinzione delle tipologie di crimine informatico viene mappata da J. Yaacoub [4]

nella classificazione delle categorie di criminale informatico, descritte come segue:

- **Cyber-criminals**: solitamente gruppi organizzati di hacker o individui che conducono rapine informatiche, atti di bullismo, ricatti o divulgazione di informazioni private (finanziarie, militari, mediche o governative) a terzi malintenzionati attraverso il deep dark web per guadagni personali o monetari.
- **Hacktivists**: hacker che mirano a lanciare attacchi DOS o di defacement del web come cyber-protesta contro un partito politico o un governo.
- **Cyber-terrorists**: sfruttano il web-defacement e la fuga di informazioni contro organizzazioni, industrie petrolifere, installazioni governative e militari.
- **Cyber-spies**: attaccano organizzazioni, imprese, installazioni governative e militari per operazioni di spionaggio e/o sabotaggio.

Il processo forense che ha inizio con l'avvenimento di un crimine informatico, tenta di rispondere alle 5WHs circa le motivazioni dell'attacco e termina con la presentazione delle prove forensi durante un processo penale, viene descritto dallo standard IDIP. Il modello IDIP, Integrated Digital Investigation Process, riassunto da R. Kaur [1] e mostrato in figura 2.3, illustra il processo forense sottolineando la ricostruzione degli eventi che hanno portato all'incidente ed enfatizzando sulla revisione dell'intera investigazione. Il processo descritto è modellato su cinque fasi:

1. **Fase di Preparazione (Readiness)**: ha l'obiettivo di garantire che le operazioni e le infrastrutture siano in grado di supportare l'indagine.
2. **Fase di dispiegamento (Deployment)**: ha l'obiettivo di fornire un meccanismo che consenta di rilevare e confermare un incidente.
3. **Indagine fisica sulla scena del crimine**: ha l'obiettivo di raccogliere e analizzare le prove fisiche, al fine di ricostruire l'evento criminoso, e include le fasi di conservazione, sopralluogo, documentazione, ricerca e raccolta, ricostruzione, presentazione. Questa fase è condotta da "Physical Forensics Investigators".
4. **Indagine digitale sulla scena del crimine**: ha l'obiettivo di accumulare e analizzare le prove digitali ottenute dalla fase di indagine fisica, di cui riprende le fasi, spostando il focus prettamente sulle prove digitali. Questa fase è condotta da "Logical Forensics Investigators" e "Cyber Forensics Investigators".
5. **Presentazione (Review)**: ha l'obiettivo di presentare l'intera investigazione in un formato comprensibile ad individui non-tecnici e volto ad esaltare le "digital evidence" recuperate, da presentare durate il processo penale.

2.1 Le fasi del processo di Digital Forensics

Il processo di analisi forense inizia nel momento in cui si riscontra un attività illecita, condotta tramite l'ausilio di dispositivi digitali e risulta nella presentazione di evidenze digitali in un'aula di tribunale, al fine di provare o confutare un'ipotesi. Il primo compito fondamentale di un analista forense, secondo M. Pierce, autore delle procedure SANS per laptop computer circa le pratiche di Digital Forensics [5], è quello di creare un registro delle prove che documenti in modo accurato e completo il processo forense. Ogni accesso al computer requisito deve essere registrato con data e ora, oltre a indicare con precisione le operazioni eseguite. Questo documento è destinato a diventare una prova legale e come tale deve essere conservato in modo sicuro senza falsa testimonianza. L'esaminatore forense deve "mantenere una rigorosa imparzialità nel giudizio delle evidenze riscontrate, al fine di proteggere la riservatezza delle parti coinvolte nei dati". M. Pierce[5], descrive come questa riservatezza possa essere cruciale se le parti procedessero con una causa giudiziaria. La reputazione dell'esaminatore forense si riflette sulla qualità dei dati

Figura 2.3. Il modello Integrated Digital Investigation Process, IDIP

conservati. La presentazione inappropriata dei dati o la divulgazione di questi a terze parti non autorizzate può essere interpretata in campo giuridico come una distorsione delle prove e comportare procedimenti penali contro l'analista forense. Si necessita, inoltre, la verifica di integrità di ogni dato raccolto, in quanto i dati elettronici possono essere facilmente ripudiati a causa della loro natura mutevole. Quanto alle fasi del processo forense digitale, in letteratura non è presente uno standard che unifichi le correnti di pensiero. M. Reith [6] classifica il processo forense in nove fasi, che includono i quattro step principali, enunciati dal NIST nel SP 800-86 [7], di "collection", "examination", "analysis" e "reporting", comuni a tutte le fonti riscontrate in letteratura. Il modello di M. Reith [6] mostrato in Figura 2.4 è conosciuto come "Abstract digital forensics model" (ADFM) e presenta le seguenti fasi:

1. **Identification**: si individua un incidente tramite degli indicatori e se ne determina il tipo
2. **Preparation**: si raccolgono gli strumenti, tecniche, mandati di perquisizione, autorizzazioni al monitoraggio e supporto alla gestione.
3. **Approach Strategy**: si sviluppa una policy per massimizzare la collezione di evidenze seguendo delle linee guida rigorose, riducendo al minimo le possibilità di alterazione dei dati.
4. **Preservation**: si mettono in atto le procedure di isolamento, messa in sicurezza e conservazione delle prove fisiche e digitali raccolte
5. **Collection**: si rintracciano e raccolgono le informazioni digitali rilevanti per l'investigazione.
6. **Examination**: si ricercano le "digital evidence" dai dati raccolti nella fase di Collection.
7. **Analysis**: si costruiscono le conclusioni, sulla base delle prove trovate.
8. **Reporting**: si riassumono le azioni dell'intero processo e si forniscono i relativi risultati.
9. **Returning Evidence**: si restituiscono, se legittimo, gli asset fisici e digitali sequestrati al proprietario.

A supporto del processo forense il NIST [7] suggerisce delle buone pratiche che le Organizzazioni dovrebbero svolgere, al fine di gestire in modo efficiente l'analisi degli incidenti:

- Eseguire backup periodici e mantenerli per un periodo temporale definito
- Abilitare l'auditing su workstation, server e dispositivi di rete

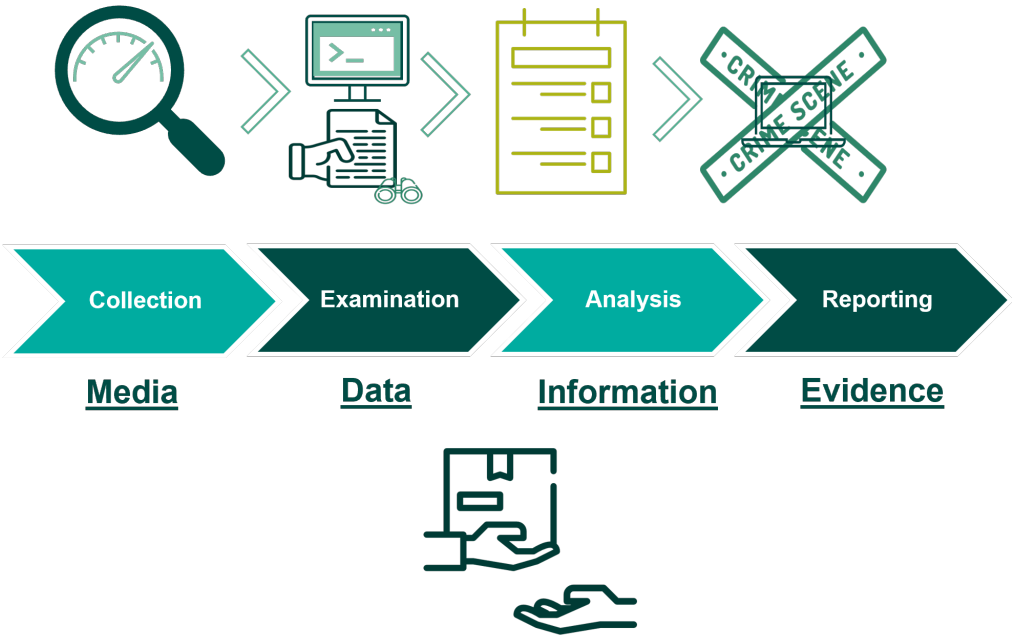


Figura 2.4. Il modello Abstract digital forensics model, ADFM

- Inoltrare i record di audit a server di log centralizzati e sicuri
- Mantenere un database contenente l'hash delle distribuzioni comuni di OS e applicazioni
- Mantenere dei record circa le configurazioni di rete e di sistema

2.1.1 Collection

La Data Collection rappresenta per diverse correnti di pensiero in letteratura il primo step all'interno del processo di Digital Forensics, successivamente all'identificazione di un cyber-crime. In primo luogo si individuano le potenziali sorgenti di dati utili al processo forense, per poi impiegarle per l'acquisizione dei dati. Esempi di "data sources" sono computer desktop, server, dispositivi di archiviazione di rete, i quali contengono unità di memorizzazione interna come HDD e SSD. A questi si sommano dispositivi di archiviazione esterna come USB, memory card, dischi ottici e dischi magnetici. I dati digitali possono essere contenuti, inoltre, in locazioni esterne rispetto all'ambiente di identificazione, come log di attività di rete memorizzati da un ISP o informazioni digitali memorizzate su Cloud. Prima di procedere con il processo di collezione, è necessaria la definizione di politiche per la registrazione di una "Chain of Custody", essenziale per evitare accuse di manipolazione o manomissione delle prove. Questa viene realizzata componendo un registro, contenente tutti gli individui che sono entrati in contatto con la custodia fisica delle prove, le operazioni che essi hanno svolto e i controlli di integrità sui dati che hanno esaminato. Inoltre, si dovrebbe mantenere un log di tutte le azioni svolte durante la collezione dei dati, includendo i tool utilizzati nel processo. In accordo con il NIST [7] l'acquisizione dei dati dovrebbe avvenire seguendo 3 fasi:

1. **Sviluppo di un piano di acquisizione dei dati**: si necessita di un piano di prioritizzazione delle acquisizioni, data la presenza di molteplici sorgenti di dato. Il piano dovrebbe essere stilato secondo i fattori di valore probabile, stimato dall'analista secondo le sue precedenti esperienze, volatilità, che rimanda alla perdita di diversi dati allo spegnimento del dispositivo, e sforzo richiesto, in termini di tempo speso dall'analista ed equipaggiamento necessario.

2. **Acquisizione dei dati**: si fa uso di tool commerciali o open source al fine di collezionare i dati volatili, duplicare le sorgenti di dati non-volatili per una successiva acquisizione e mettere in sicurezza le sorgenti di dati non-volatili. Questa fase può essere eseguita sia in locale sia sfruttando una rete sicura.
3. **Controllo di integrità sui dati acquisiti** si necessita, successivamente all'acquisizione dei dati, la verifica che essi non siano stati alterati dagli strumenti di acquisizione. A questo scopo, si confrontano i digest calcolati sui dati originali precedentemente e successivamente all'acquisizione e sui dati copiati, assicurandosi che i valori coincidano.

In accordo con gli studi e le best practice fornite dal NIST [7], l'analista forense dovrebbe raccogliere in primo luogo i dati "volatili" memorizzati in RAM, il quale contenuto andrà perso allo spegnimento del dispositivo. Per di più è bene definire una politica di prioritizzazione di acquisizione, circoscritta al trattamento dei dati volatili, e, considerata la loro natura altamente mutevole, una possibile sequenza potrebbe essere:

1. Connessioni di rete
2. Sessioni di Login
3. Contenuto della memoria
4. Processi in esecuzione
5. File aperti
6. Configurazione di rete
7. Tempo del Sistema Operativo

La collezione dei dati può avvenire per mezzo di due tecniche:

- **Backup Logico**: si copiano unicamente file e directory di interesse in un dispositivo di memorizzazione esterno.
- **Bit Stream Imaging**: si genera una copia bit-per-bit del media originale, restituendo un'immagine di dimensione elevate.

La tecnica del Bit Stream Imaging permette di copiare, diversamente dal backup logico, locazioni di memoria come "slack space" e "free space", utili per il recovery di file eliminati o nascosti. Il Bit Stream Imaging, inoltre, può essere eseguito tramite l'ausilio di tool Hardware, in grado di acquisire dati da drives con controller IDE o SCSI, o tool Software, in esecuzione sulle workstation su cui i drive sorgenti vengono collegati. La copia bit-per-bit può avvenire in modalità "disk-to-disk", nel caso si desideri copiare direttamente il contenuto su un altro dispositivo fisico, o in modalità "disk-to-file", nel caso si preferisca un file come destinazione per semplificare il backup. Terminato il processo di acquisizione l'analista forense dovrebbe creare due copie di backup dei dati rilevanti o del filesystem, tipicamente una "master copy", da conservare in un luogo sicuro, e una "working copy", utile per la generazione di multiple "working copies", nel caso di alterazione della versione su cui procede il processo forense.

2.1.2 Examination

La fase di Examination è orientata a valutare e filtrare i dati digitali raccolti nella fase di Data Collection, al fine di minimizzarne l'insieme allo stretto necessario ai fini dell'investigazione forense. Un disco rigido acquisito può contenere centinaia di migliaia di file di dati e, in assenza di una classificazione univoca in letteratura, W. Halboob [8] propone una distinzione in "Directly accessible data" (DAD), "Privacy-preserved accessible data" (PAD), o "Non-accessible data" (NAD). Più vicina alla fase di Examination, interna al processo di Digital Forensics, è la classificazione fornita da J. Yaacoub [4] in:

- **Public & Irrelevant Data**: tipi di dato che non presentano informazioni utili.
- **Public & Relevant Data**: tipi di dato che possono essere alterati o eliminati da un cyber-criminale, perchè possibile fonte di prova.
- **Private & Irrelevant Data** tipi di dato che possono essere crittati o offuscati, per esempio tramite steganografia.
- **Private & Relevant Data**: tipi di dato dall'importanza focale nel processo investigativo, in grado di rivelare informazioni circa l'attaccante e la strategia usata.

I dati raccolti e pronti per essere esaminati, indipendentemente dalla modalità di ripristino di questi, devono essere accessibili in sola-lettura al fine di assicurare l'impossibilità di modifica e alterazione, garantendo risultati coerenti alle fasi successive. La fase di Examination prevede, in seguito al ripristino dei dati collezionati da una copia di backup, una fase di assessment volta a localizzare tutti i file e identificare la porzione rilevante ai fini dell'investigazione, in particolare accedendo a locazioni di slack space e free space, che potrebbero contenere porzioni di file eliminati o nascosti. Una caratteristica importante per filtrare i dati rilevanti è rappresentata dall'analisi della tipologia dei file, utile ai fini di ricerche basate su testo o pattern. Questa può essere rivelata con precisone analizzando i file header, che per di più sono in grado di indicare facilmente se un file è crittato e il metodo di crittatura, oltre ad informazioni sui relativi metadata. Ai fini di accelerare questa fase gli analisti forensi sono soliti utilizzare tool specifici per il recupero e la visualizzazione dello slack space, come Autopsy e hex editor. A questo scopo il NIST [7] fornisce una lista di processi che l'analista dovrebbe eseguire utilizzando un Forensic Toolkit, installato in un disco ottico o in una workstation forense:

- **Utilizzo dei File Viewers**: al fine di visualizzare il contenuto di alcuni tipi di file più comuni e fornire un'anteprima dei dati
- **Decomprimere i file**: al fine di mostrare i file compressi, che possono contenere informazioni utili e rilevanti all'investigazione. Questa azione dovrebbe essere eseguita nelle prime fasi del processo forense, in modo da garantire ai file decompressi di essere inclusi nelle ricerche successive.
- **Visualizzazione grafica delle strutture delle directory**: al fine di ottenere informazioni generali sul contenuto dei media
- **Identificazione dei "Known Files"**: utile a selezionare di file irrilevanti all'analisi, l'analista dovrebbe consultare set di hash di "known good files" come il NSRL fornito dal NIST
- **Eseguire "String searches" e "Pattern Matches"**: utile a restringere il set di dati collezionati, tramite la ricerca di keyword
- **Accedere ai metadata dei file**: utile per collezionare informazioni sui file raccolti, come la reale estensione, l'autore o il timestamp dell'ultima modifica effettuata

2.1.3 Analysis

La fase di Data Analysis entra nel processo di Digital Forensics in seguito alla selezione dei dati rilevanti all'investigazione, durante lo step di Data Examination. Questo stadio permette di identificare e collezionare le "digital evidence", al fine di realizzare una timeline delle attività svolte dall'indagato e presentarne le conclusioni nella seguente fase di Reporting. M. Rafique [9] distingue due possibilità di analisi forense: l'analisi statica e l'analisi live. L'analisi statica, o tradizionale, si basa sull'esecuzione all'interno di un laboratorio forense di differenti tipi di operazioni sui dati raccolti, al fine di rilevare le evidenze digitali. In questa soluzione la fase di Data Collection viene realizzata eseguendo un memory dump del dispositivo sorgente, per poi spegnerlo brutalmente, spostando l'analisi in un laboratorio forense. Contrariamente, l'analisi live viene svolta direttamente sul dispositivo target, che perdura in "running mode", fornendo un'immagine più chiara e completa rispetto all'analisi statica, accedendo a dati inottenibili con la sola analisi statica e garantendo consistenza ed integrità ai dati forensi

2.1.4 Reporting

La fase di Reporting, finale del processo forense, prevede la stesura di un documento contenente le conclusioni riscontrate dall'indagine forense e i record delle le attività conservati dagli investigatori forensi. In accordo con il NIST [7] tre fattori devono essere considerati circa il formato della presentazione:

- **Spiegazioni alternative**: nel procedere la discussione l'analista può riscontrare informazioni incomplete o di duplice interpretazione. In questi casi è d'obbligo fornire considerazioni per ogni interpretazione appurata, utilizzando un approccio metodico al fine di provare o confutare ogni nota.
- **Considerazione del pubblico**: il documento deve essere stilato basandosi sulla tipologia del lettore. Questo fattore prevede una distinzione sul formato del documento, la rappresentazione dei dati di interesse e sul livello di dettaglio.
- **Informazioni utilizzabili**: l'analista dovrebbe considerare l'identificazione delle informazioni utilizzabili nel documento di presentazione. La presenza di certi dati potrebbe portare ad ulteriori conclusioni o analisi in fase di presentazione.

2.2 I rami della Digital Forensics

La Digital Forensics, come branca della scienza Forense, comprende i processi svolti al fine di interpretare i dati digitali relativi ad un crimine informatico.

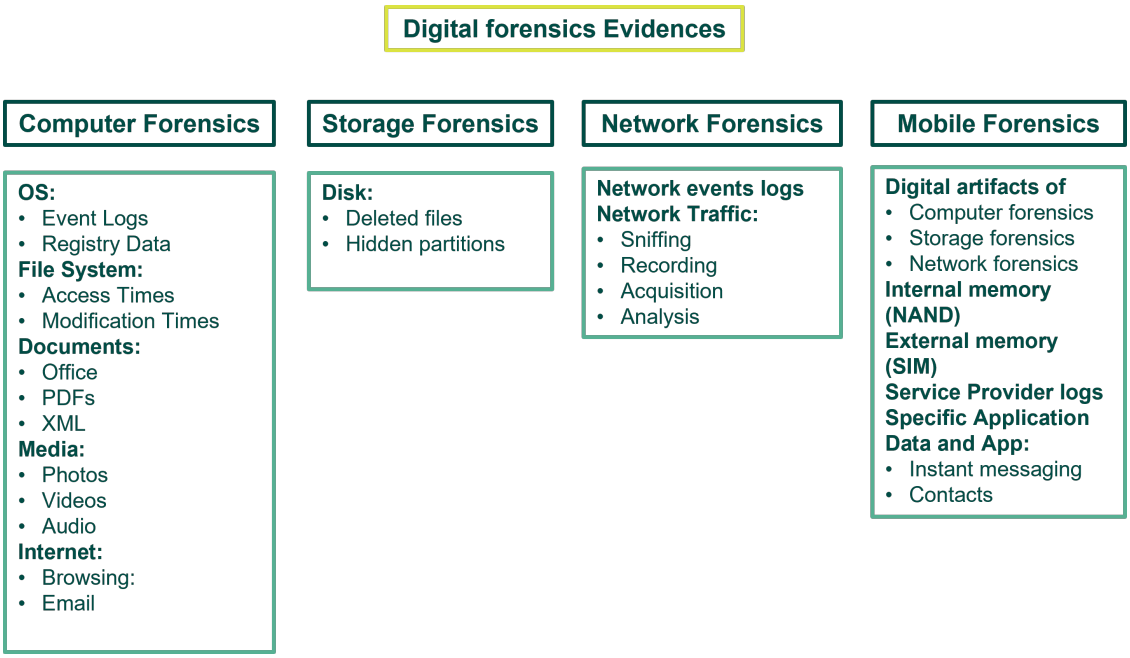


Figura 2.5. La Classificazione dei principali artefatti digitali secondo le categorie di Digital Forensics

A seconda della tipologia di evidenze digitali, mostrate in Figura 2.5, e dei dispositivi sorgente, è riscontrata in letteratura una classificazione delle attività riguardanti la forensica digitale in sette categorie, ampiamente analizzate da J. Yaacoub [4]:

- **Computer Forensics**: si collezionano i dati digitali da dispositivi informatici come computer o laptop, allo scopo di tracciare un determinato attacco verificando i processi in esecuzione ed estraendo file di sistema. In particolare l'analisi si concentra sulla Computer

Activity Timeline Detection (CAT Detect), basata sulla valutazione della linea temporale delle attività svolte sul dispositivo per rilevare le incongruenze nella cronologia di un sistema informatico, e sulla Computer Forensics Timeline Visualization (Cat Visual), basata sulla generazione di grafi temporali in grado di mostrare le attività riscontrare precedentemente e successivamente un dato evento.

- **Memory Forensics**: si collezionano i dati digitali dalla memoria fisica ad accesso casuale (RAM) di un dispositivo informatico. Questa tecnica permette di rilevare le informazioni cruciali per l'indagine sul cyber-crimine. La sfida principale della memory forensics è quella di riuscire a collezionare i dati presenti in RAM prima che svaniscano a causa della loro volatilità o per operazioni di sovrascrittura.

- **Network Forensics**: si collezionano i dati digitali derivanti da strumenti di monitoraggio del traffico di rete, come access point, switch, server, firewall, IDS, web proxy. L'analisi viene svolta tramite Network Forensics Analysis Tools (NFAT) o Network Security and Monitoring Tools (NSMT). La Network Forensics, descrive J. Yaacoub [4], può essere suddivisa in due categorie: "Catch It As You Can" (CIAYC), dove i pacchetti sono inviati ad un traffico point prima di essere archiviati in un database, e "Stop Look And Listen" (SLAL), dove i pacchetti sono memorizzati direttamente in un database per un'analisi futura.

- **Cloud Forensics**: si collezionano dati memorizzati in ambienti Cloud, ove possibile in mancanza della responsabilità dei Cloud provider. Il Cloud, strumento di archiviazione online ampiamente utilizzato e in continua crescita, offre all'utente scalabilità, ampia capacità e accesso on-demand, ma contemporaneamente può espone i dati degli utenti trasmessi sulla rete a vari attacchi. Attualmente la Cloud Forensics è strettamente legata al concetto di Blockchain, in particolar modo quanto al supporto per la conservazione delle evidenze digitali, garantendo affidabilità ed integrità. Un esempio di tecniche di Cloud Forensics strettamente legate alla blockchain è rappresentato dal Blockchain Cloud Forensic Logging (BCFL) sviluppato da K. Awuson-David [10]

- **E-mail Forensics**: si collezionano le email di un individuo o un'Organizzazione in seguito ad attacchi di phishing, attuati allo scopo di apprendere informazioni sensibili e confidenziali. L'analisi forense delle mail si presta ad esaminare il contenuto dei messaggi per determinare la legittimità, la fonte, la data, l'ora, l'effettivo mittente e i destinatari tramite un processo forense.

- **Malware Forensics**: si tenta di rilevare la presenza di malware all'interno di un dispositivo informatico tramite un processo forense. H. Khurana [11] propone una divisione della malware analysis in due aree focali: la "behavioral analysis", volta ad esaminare in che modo un malware interagisce con l'ambiente, e la "code analysis", prettamente interessata allo studio del codice del programma malevolo per osservarne il moto.

- **Mobile Forensics**: si collezionano dati digitali da dispositivi mobili con prudenza, in modo da preservarne l'integrità. Il processo forense sui dispositivi mobili, capaci di generare grandi quantità di informazioni digitali, si basa su un approccio quantitativo al fine di localizzare le evidenze e identificare i "digital artifact" dalla memoria interna (NAND), memoria esterna (SIM) e dai log del Service Provider.

2.3 Tecniche di Anti-Digital Forensics

L'attenzione dei cyber-criminali è sempre più focalizzata sullo sviluppo di sofisticate tecniche in grado di perfezionare i loro attacchi. Tra queste sussiste come interesse primario la capacità di coprire le proprie tracce ed evitare di essere rilevati. L'Anti-Forensics, o Counter-Forensics, è un'insieme di metodologie pensate per rimuovere, alterare, disturbare o interferire illegalmente con le evidenze presenti in un dispositivo digitale, interrompendo o ostacolando il processo di indagine forense, facendo uso di tecniche e tool abili nell'alterare o eliminare i file di log e audit. J. Yaacoub [4] fa un'ampia selezione delle tecniche di anti-forensics maggiormente utilizzate dai cyber-criminali, di cui si citano:

- **Hiding Data**: tecnica di offuscamento dei dati tramite crittografia o steganografia, al fine di complicare l'investigazione forense.
- **Hiding Network**: tecnica di offuscamento del traffico di rete per garantire la difficoltà di risalire all'attaccante, facendo uso di VPN, Proxy o TOR.
- **Encrypting Data, Disk, Database**: tecnica per prevenire l'accesso alle evidenze digitali ad utenti non autorizzati, facendo uso della crittografia e risultando in una perdita di tempo e risorse per l'investigatore forense.
- **Secure-Deletion**: tecnica di rimozione delle evidenze digitali dalla sorgente di acquisizione tramite sovrascrittura randomica delle locazioni di memoria.
- **Steganography**: tecnica di offuscamento delle evidenze digitali all'interno di elementi multimediali digitali, come immagini, video e file di audio. La rilevazione dell'uso di steganografia è uno degli aspetti più complicati del processo forense.
- **Timestamp Modification**: tecnica di alterazione del timestamp delle evidenze digitali al fine di disorientare l'investigatore forense.
- **File Signature Manipulation**: tecnica di alterazione della firma digitale presente all'inizio di ogni file al fine di disorientare l'investigatore forense.
- **Dummy Hard-Disk**: tecnica che prevede il boot del sistema operativo da USB e la memorizzazione dei dati su Cloud. L'Hard-Disk interno viene periodicamente aggiornato mediante scritture randomiche, così da indurre un investigatore forense a pensare che sia stato usato recentemente, risultando in uno spreco di tempo e risorse.

Di fronte alle procedure sopra citate, in contrasto al processo forense, si posizionano, in rapida evoluzione, le tecniche di Anti-Anti-Forensics. L'Anti-Anti-Forensics nasce con lo scopo di proteggere la forensica digitale, puntando ad individuare i tentativi di ostacolo dell'Anti-Forensics. Un esempio dell'attuazione di queste tecniche viene descritto da K. Conlan che, in [12] propone un riconoscimento delle tecniche di anti-forensics, formando un dataset di file che le implementano e confrontando i relativi hash con i valori malevoli conosciuti memorizzati nel NIST NSRL: in caso non vi siano corrispondenze, si evidenzia l'esistenza di una tecnica di anti-forensics non registrata dal NIST

Capitolo 3

Memory Forensics

La principale attenzione di un criminale informatico, ai giorni d'oggi, è rappresentata dal compiere i propri fini malevoli evitando che le sue attività vengano identificate o interrotte. A questo scopo sono in continua evoluzione metodi che permettano ad un programma intrusivo di sottrarsi al blocco di anti-virus e anti-malware, nascondendosi nella memoria RAM del sistema, senza lasciare traccia nei dispositivi di memorizzazione non-volatili. La Memory Forensics è una divisione della Digital Forensics centralizzata nell'acquisizione e analisi di artefatti della memoria volatile di un sistema compromesso. Le basi della materia risalgono ai primi anni 2000, quando i metodi erano sperimentali e l'attenzione era prettamente rivolta alle tecniche di Storage Forensics, focalizzata principalmente sull'immediato spegnimento del dispositivo, procedendo con l'analisi "post-mortem" del sistema mediante copia bit-per-bit del disco di archiviazione. Solo nell'ultimo decennio le tecniche di Memory Forensics hanno guadagnato terreno, evolvendosi e sviluppandosi in risposta al dinamismo della tecnologia odierna. La particolarità di questa branca è rappresentata dalla disponibilità di evidenze che non possono essere recuperate altrove e il cui effimero ciclo di vita termina al momento dello spegnimento del sistema o della sovrascrittura della relativa pagina di memoria. La pratica delle attività forensi sulla memoria ad accesso casuale (RAM), viene definita da M. Ligh [13] come una forma d'arte, l'ambito più fruttuoso, interessante e provocatorio della Digital Forensics, per cui certi dettagli possono risultare immediatamente ovvi, e altri potrebbero richiedere del tempo per essere notati mentre si continua a esplorare e imparare, come accade nell'analisi di un dipinto. Ogni funzione eseguita da un sistema operativo o da un'applicazione comporta modifiche nella memoria RAM, che di conseguenza conterrà i dati critici relativi alle ultime azioni svolte, dall'utente o da un processo, sul sistema. Ed è proprio dall'estrazione degli artefatti dalla memoria volatile che diviene possibile la ricostruzione completa degli eventi, al fine di definire gli accadimenti precedenti e successivi all'infezione del malware o all'intrusione del criminale informatico. Gli attacchi in questione vengono eseguiti prettamente con l'ausilio di "memory-resident malware" o "file-less malware", software dannosi in grado di memorizzarsi direttamente nella memoria volatile di un sistema, senza lasciare traccia di infezione altrove. La peculiarità dei dati volatili è che sono temporaneamente disponibili sul dispositivo mentre è in funzione, ma in caso di riavvio o spegnimento vengono eliminati dalla memoria del sistema. Di conseguenza, è di cruciale importanza operare in modalità "live" e "time-sensitive" al momento della fase di acquisizione forense, confutando le linee guida pubblicate nel 2008 dal National Institute of Justice in materia di conservazione degli artefatti digitali e focalizzando l'attenzione sull'Ordine Di Volatilità (OOV) mostrato in Figura 3.1.

Nel campo della Memory Forensics sono stati sviluppati diversi formati di acquisizione della memoria RAM, tra cui C. De Alwis [14] cita:

- **RAW Format**: comunemente utilizzato dai moderni strumenti di cattura prevede l'acquisizione della memoria in un ambiente "live".
- **Windows Crash Dump**: usato di default dal sistema operativo Windows per acquisire informazioni sullo stato del computer in caso di crash. Sono riconosciute tre tipologie a seconda dei dati raccolti: complete memory dump, kernel memory dump, small memory dump.

- **Windows Hibernation File**: usato in fase di ibernazione dei sistemi Windows, che consiste nello spegnimento del sistema salvandone lo stato della memoria RAM su disco rigido, in modo da ripristinarne il contenuto in fase di ripresa.
- **Expert Witness Format**: formato commerciale usato da EnCase Forensics, per cui sono disponibili pochi tool di analisi

L'analista forense, al momento dell'acquisizione della memoria volatile, deve essere in grado di riconoscere il livello di accuratezza dei dati raccolti, in quanto le pagine di RAM sono soggette a qualsiasi tipo di alterazione e sovrascrittura, a causa dei processi in esecuzione in background. I dati volatili, che risiedono nei registri, nella cache e nella RAM, che si rivelano utili all'analisi forense sono stati classificati da V. Mishra [15] come segue:

- **Connessioni di rete passate e correnti**: si rilevano l'indirizzo IP remoto, con cui il malware sta comunicando e inoltrando i dati catturati, e il numero di porta utilizzato nelle connessioni di rete. Dall'indirizzo IP è possibile rilevare la fonte dell'attività criminale o la posizione fisica in cui vengono trasferite le informazioni.
- **Elenco dei processi attivi durante l'acquisizione della RAM**: si rileva l'insieme dei processi in esecuzione in memoria, al fine di fornire un giudizio sulle attività svolte dal sistema. Il livello di astrazione è più basso rispetto a quello di un Task Manager, incapace di rilevare processi malevoli come rootkit e trojan.
- **Username e password**: si rilevano username e password inseriti dall'utente per l'autenticazione a diversi servizi.
- **DLL**: si rileva l'elenco delle "Dinamically Linked Library" (DLL) associate ad un processo in esecuzione, rendendo possibile identificare le DLL malevole iniettate in processi benigni.
- **Contenuto di una finestra aperta**: si rilevano le sequenze di tasti digitati su diversi servizi, come client di posta elettronica o chat di Instant Messaging, dando accesso ad informazioni sul contenuto e i partecipanti delle conversazioni.
- **Registry Keys relative ad un processo**: si rilevano le "registry keys" aperte per un processo in esecuzione su sistema operativo Windows. Queste permettono di individuare le funzionalità di un processo, come le funzionalità di rete o crittografiche o il metodo per la gestione del riavvio.

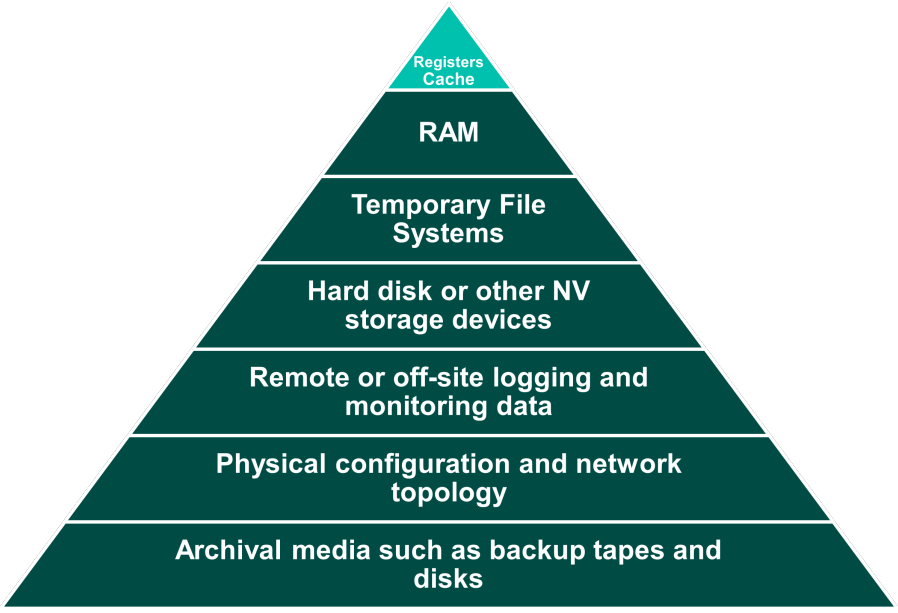


Figura 3.1. L'Ordine di volatilità, OOV

- **File aperti per un processo**: si rilevano i file aperti associati ad un processo, tra cui i file di configurazione e i file aperti in scrittura, dove potrebbero essere registrati i dati catturati.
- **Versioni decompresse/decifrate di un programma**: si rileva la presenza di file decompressi o decifrati, sfruttando la caratteristica che ogni file prima di essere eseguito deve passare per la RAM in chiaro. Si rende possibile, quindi, estrarre ed analizzare il contenuto di file malevoli, altrimenti inattuabile nelle versioni compresse e crittate all'interno del disco rigido.
- **Memory-resident malware**: si rilevano le tracce di malware residenti esclusivamente all'interno della memoria volatile, nella quale si salvano dati prima di essere trasmessi in modalità remota, senza lasciare tracce sul disco rigido.

3.1 Le principali tecniche di memory forensics

La Live Forensics fornisce la collezione di evidenze digitali, catturate mentre il sistema è in esecuzione, per mezzo di un processo che segue l'Ordine Di Volatilità delle risorse in memoria, al fine di mantenere quanto più integri e accurati i risultati prodotti. Nel campo della Memory Forensics sono stati sviluppati nell'ultimo ventennio una serie di strumenti e procedure per catturare i "memory dump", utili per analizzare un sistema compromesso: una rappresentazione grafica viene fornita nella Figura 3.2.

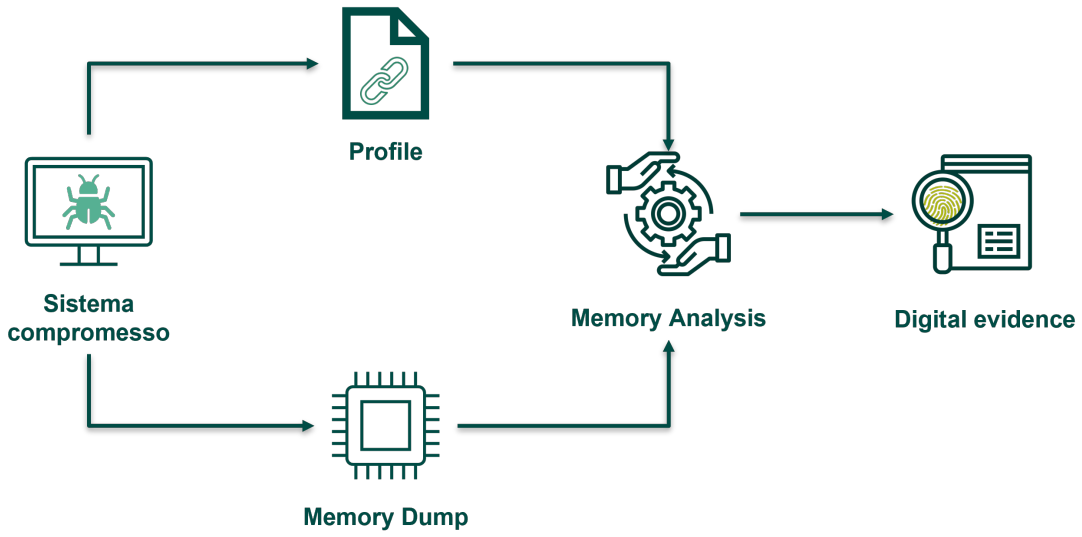


Figura 3.2. Rappresentazione generale del processo di Memory Forensics

V. Mishra [15] riconosce, dal punto di vista forense, un elenco di requisiti essenziali che ogni strumento di acquisizione della memoria volatile dovrebbe rispettare rigorosamente, tra cui:

- **Kernel-mode operation**: è necessario che il metodo di acquisizione venga eseguito in modalità-kernel al fine di oltrepassare, dati gli alti privilegi, le protezioni attuate da servizi di anti-debugger e anti-dumping, che potrebbero avviare una procedura di reboot del sistema
- **Smallest Footprint possible**: è necessario che il metodo di acquisizione lasci una nulla o minima serie di "tracce", al fine di evitare una sovrascrittura delle pagine di memoria e mantenere l'integrità dei dati in estrazione.
- **Portability**: è necessario che lo strumento di acquisizione sia "ready-to-run", memorizzato in una unità Flash USB o una locazione di memoria. Strumenti che richiedano l'installazione in loco sono illogici, ai fini della memory forensics, in quanto andrebbero ad accedere in RAM e sovrascrivere locazioni di memoria, contenenti potenziali artefatti digitali

- **Read-only access**: è necessario che lo strumento si limiti all'acquisizione di informazioni, evitando la scrittura di dati nel disco del sistema analizzato.

Il processo di Memory Forensics, secondo F. Pagani [16], può essere diviso nelle due fasi di "memory-imaging", in cui si ottiene un'immagine della memoria volatile del sistema, e "structured memory-analysis", che consente l'estrazione delle evidenze digitali successivamente al restauro dello stato del kernel in analisi. Discostandoci dalla procedura standard di acquisizione forense, da cui segue lo spegnimento della macchina, l'estrazione del disco, la copia ed analisi mediante una workstation forense, il processo di acquisizione della memoria volatile deve essere eseguito mentre la macchina è in funzione, poiché il contenuto di un chip di memoria decade rapidamente quando viene scollegato dal relativo socket. Del resto, il contenuto di un disco è pressoché statico, mentre il processo di imaging sulla memoria volatile non andrà a restituire mai lo stesso valore, a causa dell'elevata dinamicità dei processi in memoria. A questo scopo S. Vömel [17] suggerisce tre parametri per valutare la completezza di uno strumento di memory-acquisition:

- **Correctness**: misura quanto l'immagine acquisita rappresenti il contenuto originale della memoria
- **Atomicity**: misura l'impatto sul sistema durante il processo di acquisizione
- **Integrity**: misura l'integrità del processo di acquisizione confrontando i digest calcolati sull'immagine acquisita e sul contenuto originale della memoria volatile.

Il processo di acquisizione della memoria può essere condotto facendo uso di strumenti hardware o software. Quanto alle tecniche hardware-based, viene fornita in Figura 3.3 la rappresentazione dell'organizzazione fisica di un sistema moderno, con cui la citata tipologia di tool interagisce.

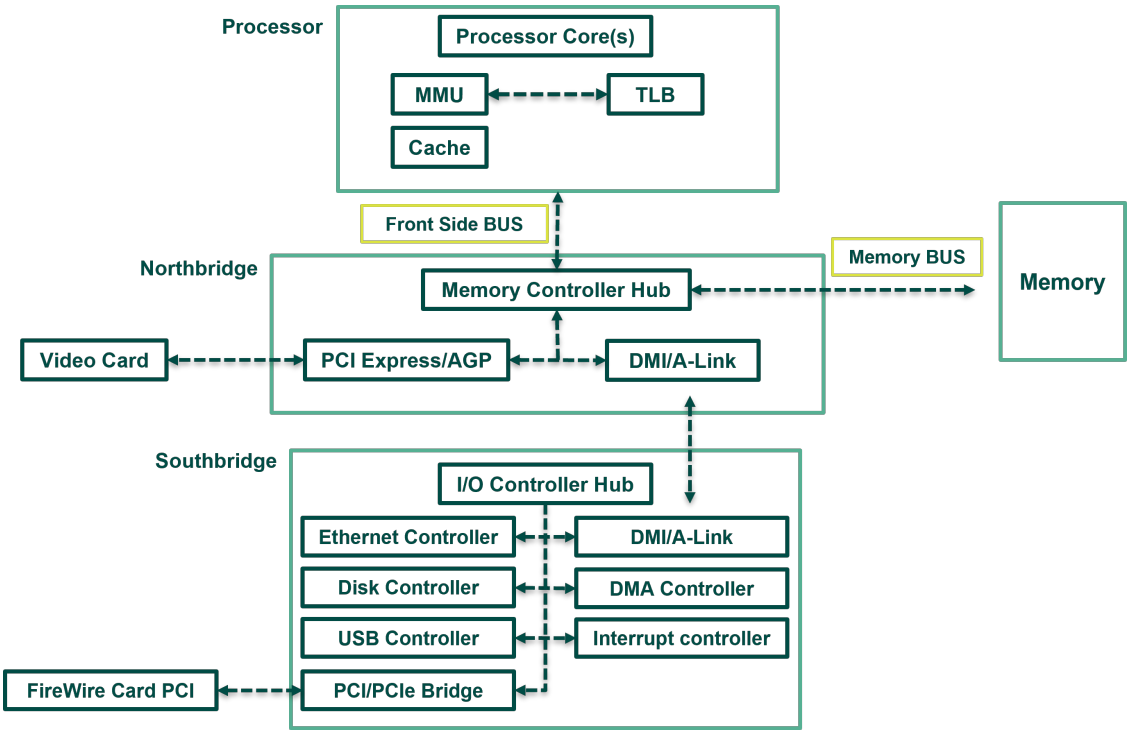


Figura 3.3. L'organizzazione fisica di un sistema moderno

B. Carrier [18] fu uno dei primi a proporre nei primi anni 2000 una procedura non dipendente dal sistema operativo, quanto da una scheda di espansione PCI capace di accedere alla memoria fisica via Direct Memory Access (DMA), aggirando la CPU e acquisendo la memoria per mezzo

di una semplice operazione di commutazione. Lo svantaggio di questa soluzione risiede principalmente nel notare che il metodo è funzionale solo se la scheda viene installata precedentemente all'incidente. In aggiunta, il prezzo elevato e la scarsa stabilità delle schede PCI hanno spostato l'attenzione su soluzioni alternative. Verso la fine del primo decennio del 2000 L. Zhang [19] presenta una tecnica di acquisizione della memoria volatile adatta ai dispositivi Windows, facendo uso del protocollo Firewire (IEEE 1934). In particolare, al fine di evitare l'evento Windows del Blue Screen Of Death (BSoD), ricorrente nella soluzione di B. Carrier [18] basata su scheda PCI, il processo di acquisizione è svolto sulla base dello spazio di indirizzamento della memoria, acquisito analizzando il valore ".Translated" nel registro di sistema. Specificamente, il valore ".Translated", raffigurato in Figura 3.4 contiene il numero di spazi di memoria, l'indirizzo iniziale e le informazioni sulla lunghezza di ogni spazio di memoria. Se da un lato la porta FireWire è comune in molti sistemi, garantendo ottime prestazioni di acquisizione, dall'altro si riscontrano problemi di accesso, per alcune configurazioni di sistema, in una sezione di memoria detta Upper Memory Area (UMA).

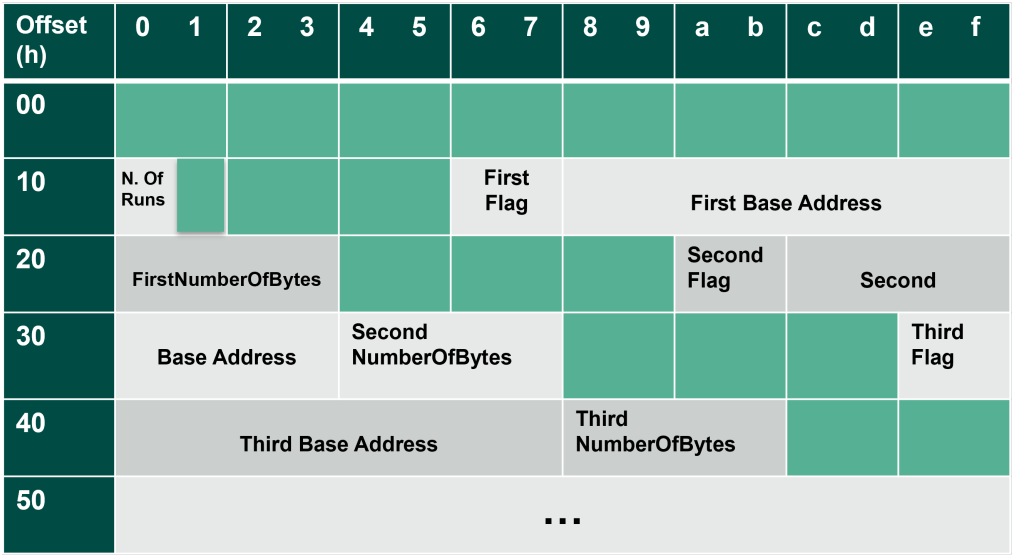


Figura 3.4. La struttura del .Translated, il valore della registry key relativa al memory layout

Il vantaggio principale delle tecniche hardware-based è rappresentato dall'assenza di interazione con il sistema operativo, annullando il rischio di sovrascrittura dei dati nel sistema sotto analisi. Quanto alle soluzioni software-based esiste una vasta scelta di applicativi in grado di supportare il processo di memory-imaging. La composizione di toolkit, largamente adottati dalla comunità forense, innalza notevolmente il livello di flessibilità del processo, garantendo sia la possibilità di scegliere l'applicativo adatto alla situazione corrente, sia la proprietà di portabilità tracciata da S. Vömel [17]. A discapito di ciò, i software di acquisizione possono risultare in memory dump inconsistenti e, quindi, causare errori nella fase di analisi successiva. I tool, disponibili su licenza o in open-source, si dividono in base allo spazio di indirizzamento in cui operano. Da una parte si definiscono i tool operanti nello spazio kernel, configurati mediante un modulo kernel inserito nel sistema operativo, con i permessi per accedere liberamente allo spazio di memoria. Il primo passo eseguito da questa tipologia di programmi è il recupero del memory-layout di sistema, rilasciato dal BIOS durante la fase di boot. Queste strutture sono mantenute in copia, dal kernel del sistema operativo, accessibili propriamente tramite iomem_resource nei sistemi Linux e per mezzo dell'API MmGetPhysicalMemoryRanges nei sistemi Windows. Successivamente, per ogni regione della RAM, il tool di acquisizione mappa una pagina di memoria fisica e la memorizza in un file. In particolare nei sistemi Linux questo processo viene svolto sfruttando l'API del kernel "kmap", utile per la traduzione degli indirizzi. D'altra parte i tool operanti nello spazio user sono limitati alla lettura di file prodotti dal kernel, accedendo ad esempio ai file /dev/kmem in Linux o \.\Device\PhysicalMemory in Windows. I file citati venivano compilati e salvati su disco in caso di crash del sistema operativo o di ibernazione del sistema. Attualmente, in seguito alla pubblicazione di Windows 8, l'accesso a questi file è raramente reso possibile dallo spazio utente. In

letteratura sono presenti diversi studi, volti a confrontare le risposte dei tool di acquisizione della memoria volatile: nel 2019 M. Faiz [20] si è focalizzato sulla valutazione in termini di DLL caricate e impatto sul registro di sistema; nel 2021 M. Martinez [21] ha spostato l'interesse sull'analisi del tempo di acquisizione e le tracce lasciate nel sistema di 6 tool open-source. Tra i più utilizzati in campo forense si citano Belkasoft Live RAM Capturer, FtkImager, DumpIt e Madiant Memoryze. In particolare Belkasoft Live RAM Capturer è un tool open-source di cattura live della memoria volatile, in grado di operare in modalità privilegiata (kernel-mode) e progettato per aggirare le protezioni anti-debugging e anti-dumping. La fase successiva tracciata da F. Pagani [16] è quella dell'analisi strutturata della memoria in cui, partendo dall'immagine di memoria acquisita e memorizzata in un file .mem, si tenta di ricostruire lo stato del kernel e di localizzare le relative strutture. L'accesso alle strutture del kernel avviene tramite dereferenziazione di "entry point", che possono trovarsi ad un offset fisso o all'interno di variabili globali. In letteratura sono presenti diversi approcci volti a scoprire il "path" dell'oggetto nella memoria del computer. D. Likhar [22] cita tra tutti il tracciamento del Virtual Address Descriptor (VAD), un albero generato dalla memoria che permette di recuperare la locazione dei file, il nome, l'ora di creazione ed altri metadati. La funzionalità di questa soluzione è limitata ai soli processi attivi. Per superare questa limitazione D. Likhar [22] propone un'alternativa all'estrazione dei metadati dei file in memoria, basata sull'accessi alle directory della File Allocation Table (FAT). Nonostante il kernel venga aggiornato con una certa dinamicità, le strutture di interesse e la modalità di salvataggio delle informazioni trovano una stabilità tra le varie versioni. Il passo successivo alla scoperta dei nodi principali per l'analisi, consiste nell'esecuzione di plugin progettati per restituire particolari informazioni, come ad esempio l'elenco dei processi in esecuzione. L'architettura basata su plugin, che trova spazio in popolari applicativi come Volatility e Rekall, permette anche agli analisti meno esperti di ottenere gli artefatti digitali di alto livello, senza la necessità di comprendere a basso livello le varie strutture, come il processo di traduzione degli indirizzi virtuali in fisici per mezzo della kernel page table. Un'altra specifica fondamentale richiesta dai vari tool di analisi è rappresentata dalla conoscenza del profilo del sistema operativo in studio. Un profilo è una categorizzazione di specifici dati strutturali, algoritmi e simboli usati nel kernel di uno specifico sistema operativo e permette l'analisi spaziale di uno specifico memory dump. Un applicativo in grado di operare entrambe le fasi risultando in ottime prestazioni è Forenscope, proposto come framework per la forensica sulla memoria volatile da E. Chan [23]. La particolarità di Forenscope risiede nel fatto di essere progettato per ridurre al minimo le alterazioni del sistema ospitante, evitando di provocare uno stato di "Forensics Blurriness". L'applicativo opera per mezzo di cinque plugin:

- **RootShell**: si tratta di una shell super-user usata per esplorare il sistema o eseguire strumenti di analisi personalizzati, in modalità non-persistente e senza lasciare tracce nel sistema.
- **Cloner**: si tratta di uno strumento in grado di catturare la memoria volatile del sistema, prevenendo le contaminazioni da rootkit.
- **Scribe**: si tratta di un modulo in grado di registrare informazioni relative all'indagine, allo stato del sistema, ai parametri hardware.
- **Terminator**: si tratta di un modulo in grado di assistere l'attività forense mandando segnali di terminazione SIGKILL a processi di sicurezza, logging e anti-forensics. In particolare vengono terminati il system daemon logger, il software antivirus e i tool di rilevazione delle intrusioni.
- **BitBlocker**: si tratta di uno strumento in grado di minimizzare le alterazioni che l'applicativo può provocare sul sistema.

3.2 Problemi e limitazioni

Le tecniche di acquisizione della memoria volatile prese in analisi hanno subito una continua evoluzione nel corso degli ultimi vent'anni, supportando l'analisi forense di sistemi composti da una quantità di processori e memoria ram in continua crescita. Recentemente F. Pagani [24],

ha pienamente analizzato le limitazioni degli strumenti di acquisizione della memoria volatile adottati nei tempi odierni, evidenziando come principale la mancanza della proprietà atomicità nel processo di memory dump.

Diversi studi in materia, tra i quali si citano i rispettivi di S. Vömel [17], M. Gruhn [25] e Le Berre [26], hanno confermato che un'acquisizione su cinque risulta in un memory dump inutilizzabile a causa dell'inconsistenza delle page table acquisite con il contenuto delle relative pagine fisiche, che viene modificato durante il processo di acquisizione. Questo fenomeno, detto "page smearing" e raffigurato in Figura 3.5, è stato analizzato, inoltre, da A. Case [27] nella sua classificazione di problemi, limitazioni e direzioni future riguardanti il campo della memory Forensics.

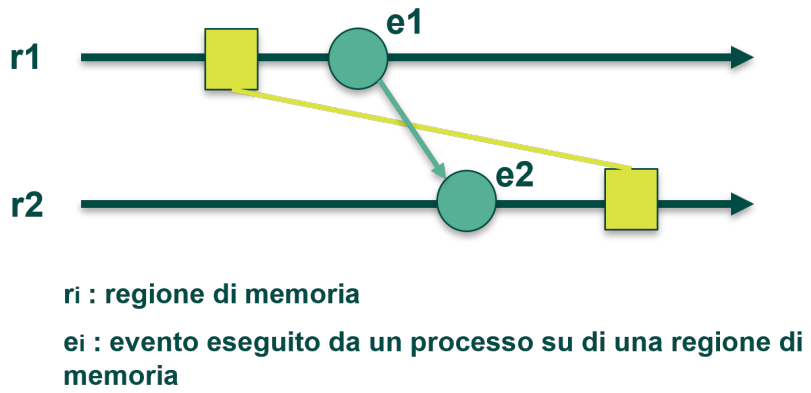


Figura 3.5. Snapshot di memoria RAM causalmente inconsistente, S. Vömel [17]

La proprietà di atomicità, nella forma più ideale, si realizzerebbe raccogliendo la memoria volatile in un'unica operazione atomica, rendendo il processo di acquisizione immediato. Nella pratica questo si approssima nella verifica che il contenuto della memoria rimanga stabile tra l'inizio e la fine del processo, come se venisse acquisito in un'unica operazione. In relazione all'indicazione di M. Gruhn [25] circa la dimostrazione che nessuna acquisizione a livello kernel risulterebbe atomica, F. Pagani [24], considerando la misura di atomicità proposta da S. Vömel [17] estremamente complicata da misurare con metodi pratici, introduce il concetto di consistenza temporale, più realistica nel valutare la qualità di un memory dump. L'autore definisce un insieme di pagine fisiche "time-consistent" se, durante il processo di acquisizione, è esistito un momento in cui il contenuto delle pagine è coesistito nella memoria del sistema. La proprietà descritta potrebbe, comunque, essere soddisfatta localmente da singoli processi o particolari strutture dati, garantendo la correttezza di diverse operazioni forensi anche in mancanza di una acquisizione istantanea completa. Sono definite in letteratura tre tipologie di inconsistenze che possono influire sulla qualità del processo di acquisizione non-atomica della memoria volatile:

- **Inconsistenza dei frammenti**: caratteristica che influisce sull'acquisizione di oggetti di grandi dimensioni, allocati su più pagine fisiche, acquisite in tempi diversi. Se il contenuto di queste non si rivela time-consistent, risultando valori diversi raccolti in frame temporali adiacenti, l'intero oggetto diviene inconsistente.
- **Inconsistenza del puntatore**: caratteristiche che influisce sull'acquisizione del valore di un puntatore e dei dati puntati in tempi diversi. Se durante il processo di acquisizione il valore del puntatore venisse modificato, il risultato sarebbe imprevedibile. Ovvero, uno strumento forense potrebbe tentare di seguire il puntatore, risultando in un codice di errore o, nella peggiore situazione, giungendo a dati validi, depistando l'analisi.
- **Inconsistenza del valore**: caratteristica che influisce sull'acquisizione di oggetti collegati tra loro, quando il valore di un oggetto non risulta coerente con il contenuto di uno o più oggetti. In particolare questa situazione trova luogo nel caso in cui una pagina cambi valore successivamente all'acquisizione dell'oggetto che la punta.

Al fine di stimare un degno grado di atomicità del processo di acquisizione, in modo che un'analista forense sia in grado di valutare il memory dump prodotto prima di procedere alla fase di analisi, F. Pagani [24] propone una soluzione, ottenuta modificando il codice sorgente del tool LiME, in grado di memorizzare per ogni pagina fisica il timestamp di acquisizione. Questa soluzione mitiga il problema di rilevamento delle inconsistenze all'interno di un memory dump, ma non previene l'attuale presenza di esse. Il problema della mancanza di atomicità trova origine in egual misura nella modalità in cui vengono acquisite le pagine di memoria fisica dalla maggioranza dei software di acquisizione. L'approccio comunemente usato trova applicazione nello scansionare la memoria dall'indirizzo fisico più basso fino a quello più alto, memorizzando il contenuto di ogni pagina in modo sequenziale. La peculiarità di questa soluzione sta nel trattare allo stesso livello ogni pagina di memoria, senza considerare se sia effettivamente utilizzata dal sistema o se abbia valore forense. Una modalità per risolvere questo problema, che dovrebbe essere implementata alla base dai tool moderni, viene citata da F. Pagani [24]. La soluzione si basa sull'esecuzione di uno smart-dump, sviluppato al fine di privilegiare in un primo momento l'acquisizione di pagine contenenti informazioni di valore forense, come la lista dei processi e dei moduli del kernel caricati. Per ogni processo viene eseguita una raccolta delle relativi informazioni in modalità localmente "atomica", in modo che due processi possano essere acquisiti in frame temporali distanti, ma il loro contenuto rimanga consistente. Al fine di tenere traccia delle pagine acquisite durante il processo di smart-dump, per ognuna, viene mantenuto un riferimento tramite una bitmap. Successivamente si acquisiscono le restanti pagine fisiche per mezzo di un memory-dump sequenziale, accedendo alla bitmap in modo che non si passi per la stessa pagina più di una volta. A. Case [27] propone un metodo alternativo, sviluppato in una prima fase di memorizzazione dello stato delle page table prima, durante e dopo l'acquisizione delle pagine fisiche, in modo da valutare possibili alterazioni e valutare eventuali scarti. A. Case [27] individua altri due ostacoli, in aggiunta alle limitazioni dovute ad acquisizioni non-atomiche, nei processi di page-swapping e demand-paging. Il page-swapping è una tecnica adottata dal sistema operativo nel momento in cui viene esaurita la memoria fisica e si cerca di far spazio ad un nuovo processo trasferendo nel disco pagine di memoria meno utilizzate, per poi essere ripristinate in caso di futuro utilizzo. Il demand-paging, è una tecnica usata dal sistema operativo al fine di caricare i dati dal disco in memoria solo al momento di assoluta necessità. Queste due tecniche aumentano l'efficenza del sistema in risposta alle attività richieste, ma impediscono agli analisti di ottenere un'immagine completa della situazione in essere sul sistema al momento dell'acquisizione. M. Ligh [13] individua la maggioranza delle pagine scambiate nei processi di swapping in quelle relative ai processi user, di alto valore forense in quanto contenenti dettagli dell'attività dell'utente come i movimenti online e di posta elettronica. I problemi della paginazione a richiesta si individuano nella mancanza in memoria di pagine e DLL relative a processi in esecuzione, rendendo impossibile, in questi, l'analisi della presenza di funzioni malevole e intrusive. A. Case [27] suggerisce una direzione futura per risolvere i problemi citati andando a memorizzare nei metadati dell'acquisizione le pagine di memoria interessate dal processo di swapping, prima della loro sovrascrittura.

3.3 Stato dell'arte

In questo capitolo si discute l'avanzamento degli ultimi anni relativo alle tecniche di acquisizione e analisi della memoria volatile di un sistema informatico, fino ad arrivare allo stato attuale della ricerca. Partendo dal 2016, K. Gupta [28] analizzò il problema di acquisizione della memoria ram in sistemi che richiedono la password dell'utente per sbloccare il dispositivo, ostacolando le procedure di memory-imaging. La peculiarità della questione in analisi, risiede nell'impossibilità di spegnimento del sistema, al fine di evitare la perdita dei dati volatili. K. Gupta [28] introdusse la possibilità di rimuovere la RAM dal dispositivo in analisi, sfruttando metodi di raffreddamento della memoria, utili a mantenere l'integrità dei dati per il periodo sufficiente al trasferimento di questa in un sistema privo di blocchi e procedere alla fase di acquisizione. L'autore effettuò 4 esperimenti, a seconda del metodo di raffreddamento utilizzato:

- **Nessun metodo di raffreddamento**: la memoria RAM è stata estratta dal dispositivo sorgente e inserita in un dispositivo senza blocchi, in un totale di dieci minuti. La temperatura della RAM al momento dell'estrazione è stata misurati pari a 35°. La percentuale di

recupero dei dati è stata misurata pari al 0.2%.

- **Azoto liquido**: la memoria RAM è stata estratta dal dispositivo sorgente e immersa immediatamente nell'azoto liquido, raggiungendo una temperatura di -196°. Al momento dell'inserimento in un dispositivo privo di blocchi, si è misurata una percentuale di dati recuperati pari al 99.81%.
- **Spray Congelante**: la memoria RAM è stata estratta dal dispositivo sorgente e ricoperta di aria compressa congelante, raggiungendo una temperatura pari a -40°. Al momento dell'inserimento in un dispositivo privo di blocchi, si è misurata una percentuale di dati recuperati pari al 96.45%.
- **Ghiaccio**: la memoria RAM è stata estratta dal dispositivo sorgente, inserita in un sacchetto antistatico e ricoperta di ghiaccio, raggiungendo una temperatura pari a 10°. Al momento dell'inserimento in un dispositivo privo di blocchi, si è misurata una percentuale di dati recuperati pari al 99.71%.

Gli esperimenti descritti hanno reso possibile rilevare come l'utilizzo del ghiaccio come metodo di raffreddamento sia il giusto compromesso in un'analisi forense, in termini di flessibilità e percentuale di dati recuperabili: i risultati sono descritti nel grafico in Figura 3.6.

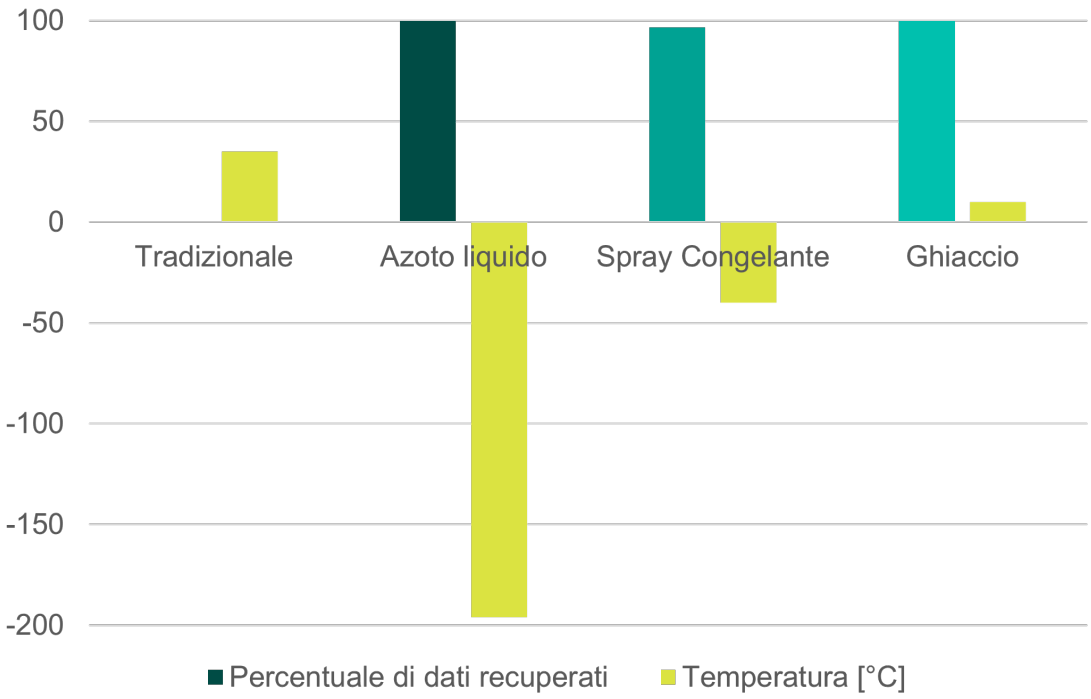


Figura 3.6. I risultati in termini di percentuale di dati recuperati e temperatura degli esperimenti di raffreddamento della memoria RAM

Nel 2017, A. Case [27] prese in analisi le diverse problematiche legate ai processi di acquisizione della memoria volatile, identificando come principali i medesimi legati al page-smearing, al page-swapping e al demand-paging. In seguito all'approfondimento di ogni classe di problemi, seguono proposte di possibili soluzioni a essi. La pubblicazione fu una delle basi per gli studi di F. Pagani [24], presentati nel 2019, circa l'importanza della dimensione temporale nel processo forense, in mancanza di un approccio atomico di acquisizione della memoria volatile nella maggioranza dei tool di memory-imaging. F. Pagani [24] sottolineò la necessità di tool che siano in grado di registrare il timestamp di memorizzazione di ogni pagina acquisita durante il processo, al fine di confermare la consistenza temporale dell'immagine di memoria collezionata. Inoltre, al fine di mitigare gli effetti della mancanza di atomicità e ridurre le inconsistenze descritte, introduce un framework di acquisizione che si discosta dall'approccio comune di collezione sequenziale

delle pagine fisiche, spostando la priorità sul contenuto e il valore forense di esse. Nel 2020 D. Prakoso [29] presenta un framework di rilevazione di attacchi generati con l'ausilio del tool Metasploit, comunemente usato ai fini di audit e penetration testing. Gli esperimenti, con l'ausilio dei tool DumpIt, FtkImager e Magnet Ram Capturer hanno reso possibile individuare: l'indirizzo IP dell'attaccante, la locazione del malware, il sistema operativo della macchina e i processi in esecuzione. Proseguendo, nel 2021 T. Thomas [30] promuove due plugin open-source per l'applicativo Volatility in grado di estrarre artefatti digitali generati da attacchi USB-based: usbhunt e dhcphunt. Il plugin usbhunt punta all'estrazione da Windows 10 delle strutture JSON DeviceConfig e InventoryDevicePnpAdd, contenenti timestamp relativi alla connettività del dispositivo e altri metadati di valore forense. Il plugin dhcphunt, mira invece all'estrazione dal processo "svchost" di Log descriventi l'attività del client Dhcp. A. Orgah [31] ha analizzato l'attuale limitazione nella mancanza di archivi su larga scala di catture di memoria "pulite", che potrebbero rendersi utili ad investigatori per il confronto con corrispettivi infetti o a sviluppatori per provare il corretto funzionamento di plugin in costruzione. Una delle motivazioni principali alla base della pubblicazione risiede nel dispendio di tempo generato da ogni ricercatore, fino ai giorni odierni, per produrre un dataset di memory-dump valido per la propria sperimentazione. In risposta a questa problematica, A. Orgah [31] sviluppa un framework open-source, MemForC, per la creazione di un "corpus" di memory-dump, successivi all'esecuzione di diverse tipologie di malware. Ancora nel 2021, S. Jung [32] propone uno studio per l'acquisizione e la verifica del layout di memoria, accedendo alle funzioni API del kernel, in modo da consentire un processo di acquisizione affidabile. A. Case [33] sposta l'attenzione su un'ulteriore limitazione dei tool di acquisizione moderni, ovvero l'assenza di analisi dell'infrastruttura di tracciamento del Kernel Linux. Nel particolare queste funzioni di tracciamento, che al livello più basso sono rappresentate da ftrace, kprobes, e tracepoints, forniscono una visione significativa delle prestazioni e del comportamento delle applicazioni e dei componenti del sistema e possono essere sfruttate da malware, che per via delle tecniche odierne resteranno inosservati. In risoluzione, A. Case [33] pubblica una serie di Volatility plugin volti a rilevare l'abuso di queste funzionalità all'interno dei campioni di memoria analizzati. Nel campo forense, inoltre, sono in continua evoluzione framework in grado di fondere il processo di acquisizione della memoria volatile con i concetti di Blockchain, allo scopo di mantenere l'integrità nel tempo dell'intero processo. M. Hari [34] introduce un framework, integrato al tool di acquisizione LiME e al tool di analisi Volatility, consistente nel memorizzare l'hash del memory dump acquisito e altri attributi necessari del file all'interno di una blockchain e successivamente allegare in essa i dati dell'investigatore. Al momento dell'analisi del memory dump, l'investigatore potrà controllare l'integrità del file consultando il valore all'interno della blockchain. Infine, uno degli studi più recenti sull'inconsistenza nell'acquisizione della memoria volatile viene pubblicato nel 2022 da J. Ottmann [35]. Dopo aver preso in esame le diverse pubblicazioni in materia, specialmente i concetti di "correctness, integrity, e atomicity" di S. Vömel [17] e di "time-consistency" di F. Pagani [24], l'autore fornisce una propria versione ai citati concetti. Quanto alla prima definizione di S. Vömel [17] circa l'atomicità, prettamente basata su una relazione di dipendenza causale, vengono presentate le definizioni di consistenza istantanea e consistenza quasi-istantanea. Una cattura soddisfa la coerenza istantanea se tutte le regioni di memoria sono state acquisite nello stesso momento: un esempio è raffigurato in Figura 3.7.

Si soddisfa la proprietà di consistenza quasi-istantanea, mostrata in Figura 3.8 se i risultati della cattura si rivelano pari a come sarebbero se catturati in modo atomico, raggiungibili garantendo l'impossibilità di alterazione del contenuto della memoria durante il processo di acquisizione, ad esempio sfruttando il metodo "copy-on-write", noto nell'ambito della programmazione di sistema e rappresentato in Figura 3.9.

Quanto alla definizione di integrità proposta da S. Vömel [17], soddisfatta unicamente se il contenuto della memoria non venga modificato durante il processo di acquisizione, J. Ottmann [35] propone una soluzione più "permissiva", permettendo modifiche della memoria, purchè il valore presente nelle celle all'inizio e alla fine della misura sia equivalente.

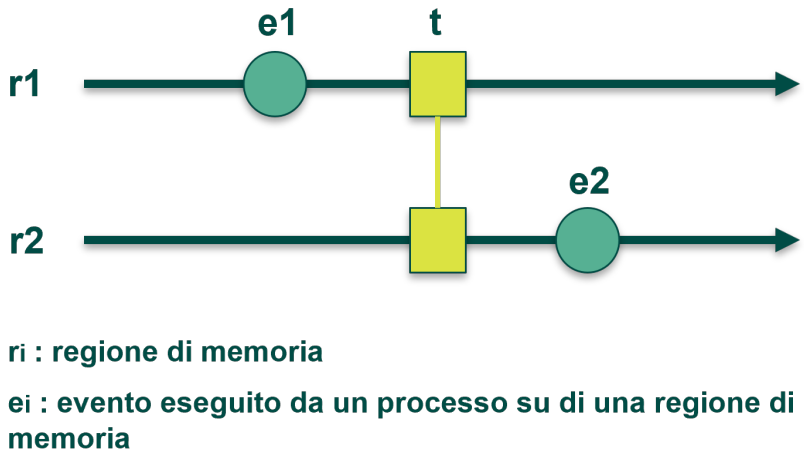


Figura 3.7. Quando la memoria viene catturata istantaneamente, tutte le regioni vengono acquisite allo stesso tempo

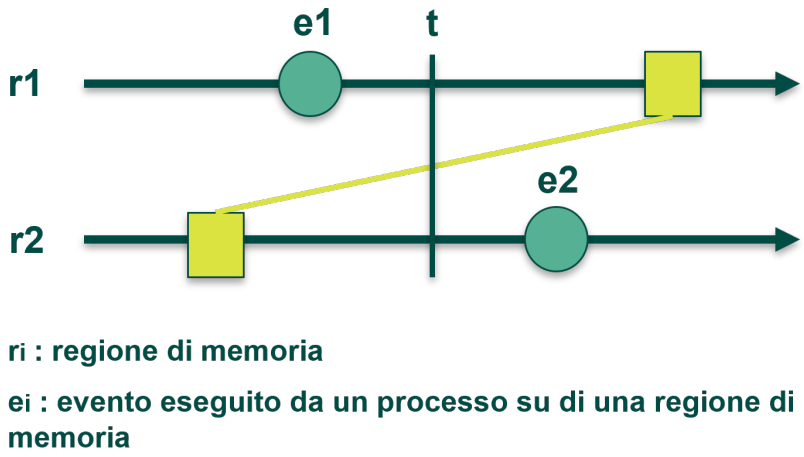


Figura 3.8. Uno snapshot di memoria quasi-istantaneo

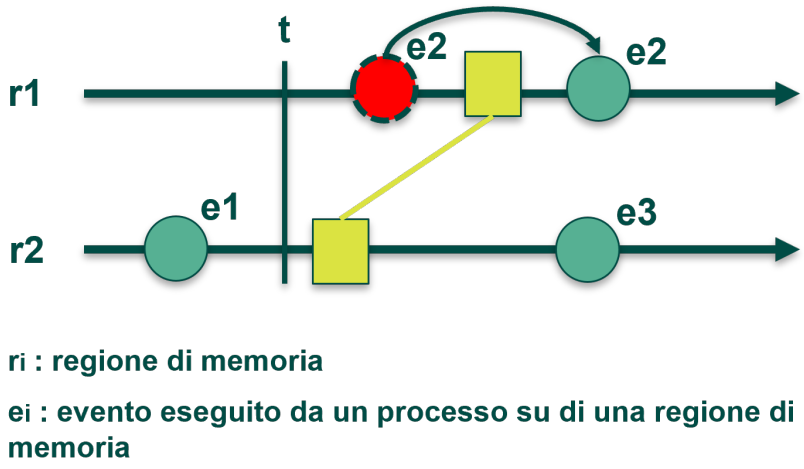


Figura 3.9. Un metodo per ottenere la consistenza, tramite la tecnica copy-on-write

Capitolo 4

Stealthy malware

L'era digitale è caratterizzata dall'uso sempre più frenetico e consapevole delle tecnologie dell'informazione e della comunicazione. E' ormai affermato quanto le infrastrutture pubbliche, private, governative, sanitarie e militari pongano le proprie basi su sistemi informatici. Allo stesso modo, le nostre esistenze individuali non possono fare più a meno di applicativi di condivisione di dati personali, professionali, sociali e finanziari. Questo ambiente ha portato allo sviluppo di minacce informatiche, inizialmente progettate al fine di ottenere una posizione di rilievo nell'Olimpo dell'hacking, successivamente mirate all'indirizzamento di bersagli più grandi e generatori di profitto. Se nel campo della crittografia nel corso della storia, fin dagli antichi Egizi, i crittografi e i crittoanalisti hanno giocato al gatto e al topo al fine di consegnare o decifrare un sistema di segni capace di nascondere un messaggio importante [36] , lo stesso accade nel settore della sicurezza informatica che, fin dagli albori dei computer, vede come attori gli autori di malware e l'industria della sicurezza. Quest'ultima si pone come obiettivo lo sviluppo di programmi in grado di rilevare e prevenire l'attacco intrusivo dei virus informatici. L'evoluzione delle contromisure all'industria dei malware hanno portato allo sviluppo di una tipologia di malware particolarmente sofisticati, gli "stealthy malware", in grado di condurre attacchi informatici dalle conseguenze catastrofiche senza essere rilevati da qualsiasi soluzione antivirus.

Il termine "stealth" può essere utilizzato come termine generale per tutti i tipi di codici maligni che sono in grado di nascondersi per non essere visibili. In generale, le azioni di camuffamento del malware stealth possono essere classificate in due aspetti: nascondere le tracce del malware o nascondere il proprio codice all'uomo o ad altri programmi, impedendo tanto l'analisi statica e il reverse engineering del virus, quanto la modifica del codice [37].

Se da un lato i malware tradizionali, o "file-based", una volta esplosi apportano modifiche al filesystem al fine di danneggiare un sistema o ottenere informazioni sensibili, gli stealthy malware, o "file-less", si immettono direttamente nella memoria volatile del sistema senza lasciare tracce dei loro movimenti sul disco e senza scaricare eseguibili malevoli. I malware appartenenti a questa categoria sono in grado di eludere le soluzioni di rilevamento degli antivirus, diffondendosi in modo dormiente e persistente nei sistemi informatici e conducendo attacchi ad impatto nullo. La particolarità dei malware file-less risiede nel non dipendere da programmi complessi installati su disco in fase di esplosione, il cui flusso di istruzioni potrebbe essere rilevato dalle soluzioni di rilevamento basate sulle firme, ma dallo sfruttamento di API comuni e legittime, pre-installate nel sistema. Non a caso, i vettori principali per gli attacchi condotti dai malware stealthy vengono forniti, nel Sistema Operativo Windows, dagli strumenti di amministrazione del sistema, in particolar modo Powershell e Windows Management Instrumentation (WMI). Una volta iniettato nel sistema, il virus potrebbe essere in grado, in base alle specifiche di design, di fornire il controllo del sistema (C&C) agli attaccanti, sfruttando le connessioni di rete. Un'ulteriore caratteristica fondamentale che permette alla citata tipologia di malware di identificarsi come "stealth" è rappresentata dalla capacità di persistere in RAM. Si è descritto diverse volte nei precedenti capitoli di come un processo in RAM abbia un ciclo di vita effimero, che termina allo spegnimento del dispositivo, derivante dalla caratteristica volatile della memoria principale. Ebbene, dopo aver infettato il sistema, un malware stealthy può essere in grado di rieseguirsi in memoria, utilizzando dei meccanismi come le "Registry entry", gli eventi attivati da WMI e le attività del "background

intelligent transfer service" (BITS): per questo motivo ci si riferisce ai virus stealthy anche con l'appellativo di memory-resident malware [38].

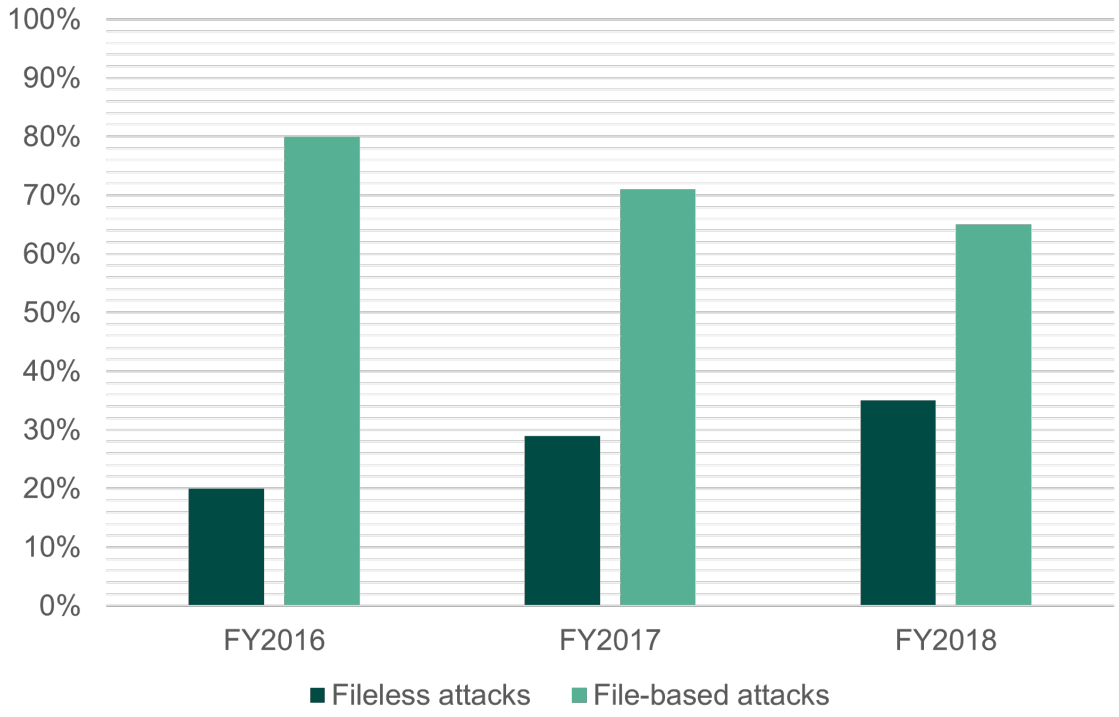


Figura 4.1. Confronto degli attacchi file-based e fileless per anno

Secondo uno studio del Ponemon Institute [39], il 77% degli attacchi file-less avvenuti nel 2017 ha riscontrato esito positivo, stimando una probabilità di successo dieci volte superiore rispetto alle corrispettive file-based. In dettaglio viene riportata in Figura 4.1 la statistica sull'evoluzione degli attacchi file-less in contrapposizione alle versioni file-based. Un caso esemplare è rappresentato dall'Equifax data breach, che nel 2017 tramite l'utilizzo di tecniche file-less ha portato alla violazione di dati privati di 147,9 milioni di cittadini statunitensi, 15,2 milioni cittadini inglesi e 19.000 cittadini canadesi.

Le motivazioni principali per lo sviluppo di malware stealthy tramite tecniche file-less sono identificate da Arista Networks [40] in:

- Rimanere inosservati per lunghi periodi di tempo, sfruttando l'inabilità dei moderni software antivirus nella rilevazione degli attacchi file-less
- Sfruttare vulnerabilità che consentano l'accesso come amministratore e il controllo completo del sistema
- Collezionare dati dell'obiettivo, rilevanti per attacchi futuri

4.1 L'evoluzione da file-based a file-less malware

I malware nel corso della loro evoluzione sono stati canonicamente classificati in categorie come virus, worm, trojan, rootkit, meramente legate alle tecniche di attacco con cui sono stati progettati. I malware avanzati spesso includono diversi componenti con funzionalità diverse che non permettono una classificazione unica del programma malevolo. A scopo dimostrativo un malware potrebbe essere in grado di mostrarsi in fase di diffusione come un virus, come un worm in fase di propagazione attraverso la rete, come una botnet durante una comunicazione Command&Control

(C&C) o come un rootkit nell'evitare di essere rilevato da un Intusion Detection System (IDS) Nella loro accezione più comune i malware sono dei programmi parassitari, capaci di infettare il sistema ospitante allo scopo di creare un danneggiamento, un disservizio o rubare informazioni sensibili, per poi trasferirle tramite connessioni di rete all'attaccante. D. Pattern [41] propone una linea temporale circa l'origine delle tipologie di malware, distinguendo cinque categorie:

- **Early Malware**: I malware appartenenti a questa categoria furono creati meramente come Proof Of Concept delle insicurezze all'interno dei sistemi dell'epoca. Venivano trasmessi principalmente tramite floppy disk e trovano il primo esempio completo in Brain, un virus sviluppato nel 1986 da due fratelli originari del Pakistan, in grado di infettare il boot sector di un sistema MS-DOS, causando rallentamenti significativi e mostrando le informazioni di contatto degli autori.

- **Windows Malware**: I malware appartenenti a questa categoria sono creati per attaccare i sistemi aventi Windows come Sistema Operativo. WinVir, che ne rappresenta il primo esempio, è un virus creato nel 1992 da M. Khafir in grado solo di diffondersi da file a file, disinfettandosi da un programma mentre ne infetta un altro. Era in grado di auto-replicarsi e auto-distruggersi. Il primo macro-virus fu WM.Concept: progettato per Microsoft Word nel 1995 era in grado di diffondersi tramite la condivisione di documenti Word, copiare un template dannoso sul master template in modo da infettare ogni nuovo documento che l'utente avrebbe creato. Il primo email-virus fu invece Happy99, contraddistinto dal non danneggiare il sistema ospitante, ma visualizzare a video un filmato con fuochi artificiali. Happy99 è stato una PoC di come un virus potesse sfruttare la connessione internet del sistema ospitante per propagarsi tramite email.

- **Network Worm**: I malware appartenenti a questa categoria hanno la capacità di danneggiare il sistema auto-replicandosi, per poi diffondersi sulla rete, sfruttando una vulnerabilità nel sistema infetto o tramite condivisione di file infetti e all'apparenza legittimi tramite e-mail [42]. Il primo worm di questa classe fu il Morris Worm nel 1988, creato come Proof Of Concept dallo studente della Cornell University Robert Trappan Morris e lanciato dai laboratori del Massachusetts Institute of Technology (MIT). A causa di un errore di programmazione, invece di inviare copie di se stesso ad altri computer, questo software continuava a replicarsi su ogni sistema infetto, riempiendo tutta la memoria disponibile del computer. Prima che venisse trovata una soluzione, il worm aveva infettato una quantità di computer al tempo pari ad un decimo di Internet. Un esempio di particolare rilievo è rappresentato, inoltre, dal worm ILOVEYOU, in grado di infettare un sistema e mandare copie di se stesso a tutti i contatti della rubrica mail presente nel sistema ospitante. ILOVEYOU si è diffuso in tutto il mondo dal 4 Maggio 2000 ed è stato in grado di infettare il 10% dei computer connessi ad internet in un solo giorno. Lato server, il Code Red worm si è distinto per aver infettato 359.000 server usando un attacco buffer overflow [43].

- **Ransomware**: I malware appartenenti a questa categoria hanno la capacità di infettare il sistema ospitante crittando il contenuto del disco e rendendo impossibile l'utilizzo del dispositivo previo inserimento della chiave di decrittazione, inviata tramite connessioni di rete anonime al Command&Control server (C&C). Questa tipologia di malware è diventata popolare per via degli alti margini di profitto ottenibili dall'attaccante, in quanto l'invio della chiave di decrittazione prevede il pagamento di un riscatto in una crypto-moneta, come il Bitcoin (BTC). Il primo malware di questa categoria, il Trojan.Randsom.C diffuso nel 2008, falsificava un messaggio del Windows Security Center, richiedendo una chiamata telefonica a tariffa maggiorata al fine di riattivare la licenza del software di sicurezza.

- **State Sponsored Malware**: I malware appartenenti a questa categoria sono sviluppati dalle infrastrutture militari degli Stati nazionali a scopo di spionaggio e sabotaggio. La concezione del malware passa dall'essere un problema finanziario e di disservizio per le aziende e gli utenti privati, all'essere una vera e propria arma sfruttata dalle agenzie militari, sfruttando il diritto di usare e difendersi dagli attacchi informatici. L'esempio più rilevante è rappresentato da Stuxnet, che fu diffuso nel 2010 dai servizi segreti statunitensi e israeliani al fine di colpire gli impianti fisici di arricchimento dell'uranio iraniani a Natanz e rallentare il relativo programma nucleare. Stuxnet viene definito come un "missile informatico di

livello militare" e rappresenta il primo esempio di valore di attacco informatico ai sistemi fisici. Un altro malware rilevante di questa categoria è Flame, diffuso nel 2012 dai servizi segreti israeliani e statunitensi. Flame è in grado di diffondersi tramite connessioni di rete o dispositivi USB e installa un rootkit che permette all'attaccante di spiare la vittima e registrare informazioni accedendo alla fotocamera e al microfono, per poi distruggere tutte le istanze presenti nel sistema infettato tramite comandi remoti Comman&Control (C&C).

Inizialmente la progettazione dei programmi intrusivi era riservata a utenti esperti in grado di stendere codici direttamente in linguaggio Assembly, riconosciuti come vere e proprie opere di ingegneria. Con il tempo l'industria dei malware, in grado di generare profitti monstre, si è specializzata verso linguaggi di alto livello, in modo da permettere anche ad utenti meno esperti l'entrata nel mercato dei malware. Un evento fondamentale in quest'ottica è stato il rilascio del framework .NET da parte di Microsoft, sfruttato dai creatori di malware per le funzionalità offerte al fine di interagire con il Sistema Operativo e coordinare attacchi tramite le API di Powershell, dando origine a malware capaci di evitare le tecniche di rilevazione dei moderni software antivirus, mantenendo la persistenza nel sistema e infiltrandosi nei dati. L'infrastruttura Windows Management Instrumentation (WMI) rappresenta un'ulteriore arsenale alla portata degli autori di codice malevolo. Il sistema WMI nasce per automatizzare le attività amministrative nei computer remoti e per fornire dati di gestione al sistema operativo Windows, ma diviene di particolare interesse per la comunità hacker per la capacità di eseguire la ricognizione del sistema, l'individuazione di macchine virtuali, l'esecuzione di script malevoli direttamente in memoria, il furto di dati e la generazione di backdoor senza rilasciare un singolo file sul fileystem [44]. Questa direzione ha portato allo sviluppo dei malware file-less, in grado di immettersi nella memoria volatile del sistema senza lasciare tracce dei loro movimenti e della loro persistenza sul disco, eludendo i controlli degli antivirus. Il software antivirus è progettato per rilevare, prevenire e intervenire per bloccare o rimuovere il software dannoso e i relativi file dal computer, tramite la scansione del dispositivo alla ricerca di "known pattern" o modelli comportamentali che possano segnalare la presenza di malware conosciuti e non. Ogni misura di riparazione ad un'intrusione è tuttavia irrilevante se il malware è in grado di sottrarsi agli strumenti di monitoraggio. In letteratura i malware file-less vengono spesso identificati con il termine "rootkit"sebbene, come descritto da E. Rudd in [45], esso "si riferisca propriamente ai moduli che reindirizzano l'esecuzione del codice e sovvertono le funzionalità previste del sistema operativo allo scopo di mantenere la segretezza". Infatti, per quanto possa essere seppur individuabile in diversi sample, la caratteristica di mutazione del codice propria dei malware stealthy non è una prerogativa dei rootkit. Il primo esempio di malware file-less risale al 2001, con l'emergenza lanciata dalla diffusione del Code Red Worm, che sfruttava una vulnerabilità di buffer overflow nel Microsoft Internet Information Services (IIS) per scrivere comandi nella working memory di un server. Altri esempi rilevanti di malware con caratteristiche stealthy sono rappresentati da SQL Slammer nel 2003, Stuxnet nel 2010, UIWIX nel 2017. L'evoluzione dei malware stealth procede parallelamente al progresso delle relative tecniche di camuffamento, che aumentano la possibilità del programma malevolo di sopravvivere al controllo delle soluzioni antivirus e, nel contempo, prolungarne la vita.

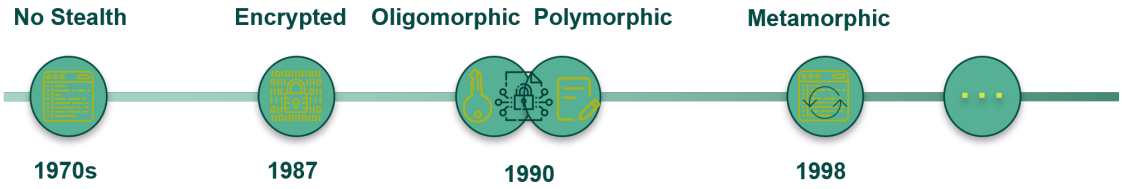


Figura 4.2. Evoluzione delle tecniche stealth nei malware

B. Bashari Rad in [37], descrive lo sviluppo delle tecniche di camuffamento presenti nei malware stealth in quattro passi, come indicato in Figura 4.2:

- **Encryption**: i virus crittati sono composti da due sezioni fondamentali, il ciclo di decrittazione e il corpo principale, come mostrato in Figura 4.3. Il corpo principale contiene il

codice macchina dell'eseguibile e necessita di essere decrittato dal "Decryptor" prima di essere eseguito, come dimostra la Figura 4.4. Il metodo di crittatura può basarsi su di una trasformazione del codice byte-per-byte o sfruttare tecniche di crittografia reversibili, come ADD o XOR con chiavi a valore costante o variabile. Il vantaggio per un malware nell'implementare questa tecnica risiede nel non essere immediatamente rilevato dalle soluzioni antivirus che utilizzano tecniche signature-based, fino al momento della decrittazione. Nel contempo, la sezione del decryptor, essendo in chiaro, è osservabile dagli scanner e sottomessa ai controlli di firma, in modo da rilevare indirettamente il virus. Il primo esempio di virus criptato è Cascade, diffusosi nel 1987.

Figura 4.3. Struttura di un malware cifrato

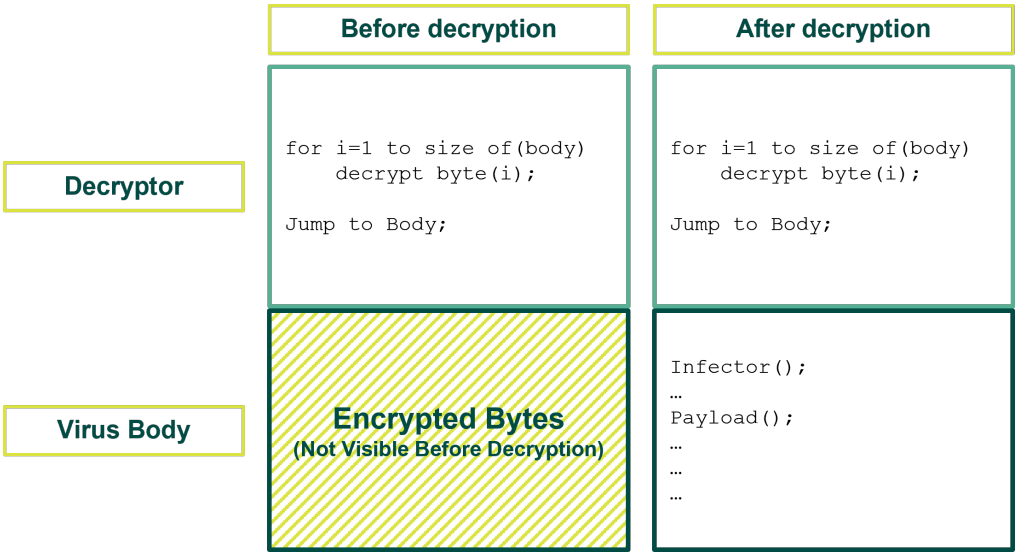


Figura 4.4. Decryptor e Virus body, prima e dopo la decifratura

- **Oligomorphism**: i virus che implementano questa tecnica di camuffamento, indicati in Figura 4.5, prevedono l'utilizzo di un modulo decryptor diverso ad ogni utilizzo, che si traduce in un "main body" con aspetto different ad ogni nuova infezione. Questo comportamento è reso possibile da una collezione di decrittatori diversi interna al virus, che vengono scelti casualmente per una nuova vittima, aumentando gli sforzi e il tempo per il programma antivirus che dovrà ispezionare la firma di ogni istanza della collezione, in modo da rilevare indirettamente il virus. Il primo esempio di virus oligomorfo, o semi-polimorfo, fu Whale nel 1990.
- **Polymorphism**: i virus che implementano le tecniche polimorfiche, indicati in Figura 4.6, cercano di aumentare la difficoltà di analisi del virus tramite la creazione di un numero illimitato di nuovi e differenti decryptor. L'idea alla base risiede nel modificare costantemente l'aspetto del codice, in modo che non rimangano stringhe comuni tra le varianti del virus, che potrebbero essere altrimenti sfruttate dal motore antivirus. A questo scopo si utilizzano tecniche di offuscamento del codice come:
  - **Sostituzione di istruzioni**
  - **Permutazione di istruzioni**
  - **Sostituzione di variabili/registri**

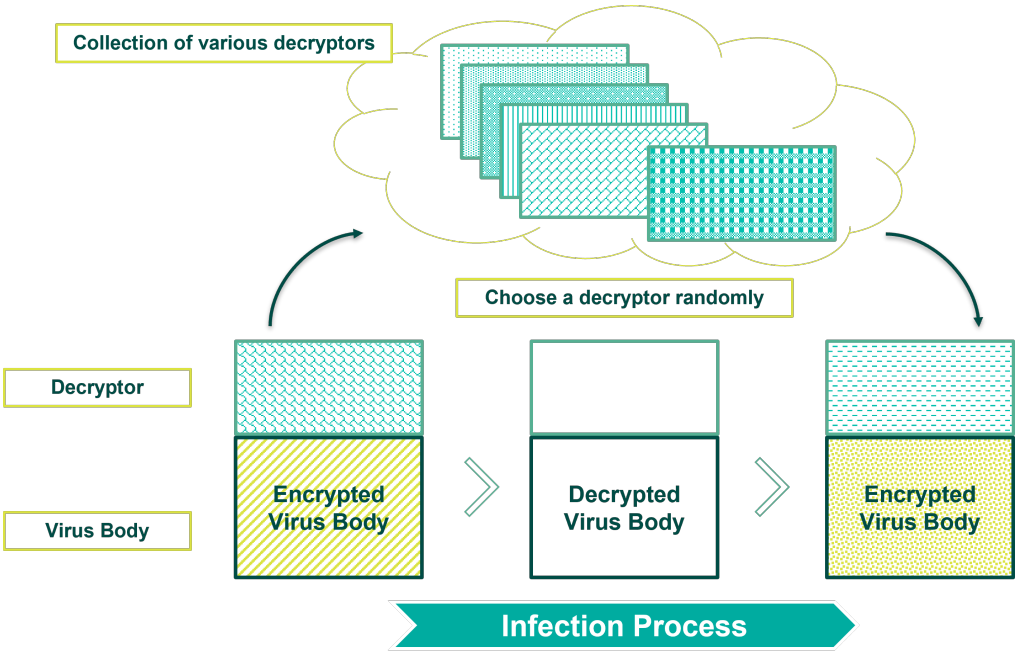


Figura 4.5. Struttura di un malware oligomorfo

- **Inserimento di Junk/Dead code**
- **Trasposizione del codice**

Il modulo responsabile di queste trasformazioni viene detto "Mutation Engine" o "Obfuscation Engine". In particolare la pubblicazione del motore di mutazione del creatore di virus "Dark Avenger" nel 1992 provocò una rapida ascesa per questa tipologia di malware. Il primo virus polimorfico, il 1260, è stato sviluppato nel 1990 da Mark Washburn ed era caratterizzato da una stringa di ricerca di 2 byte, che lo rendeva impossibile da rilevare dagli antivirus dell'epoca tramite le tecniche di signature-matching [46]. Il virus 1260 non si è diffuso tanto quanto il virus polimorfico Tequila, comparso nel 1991.

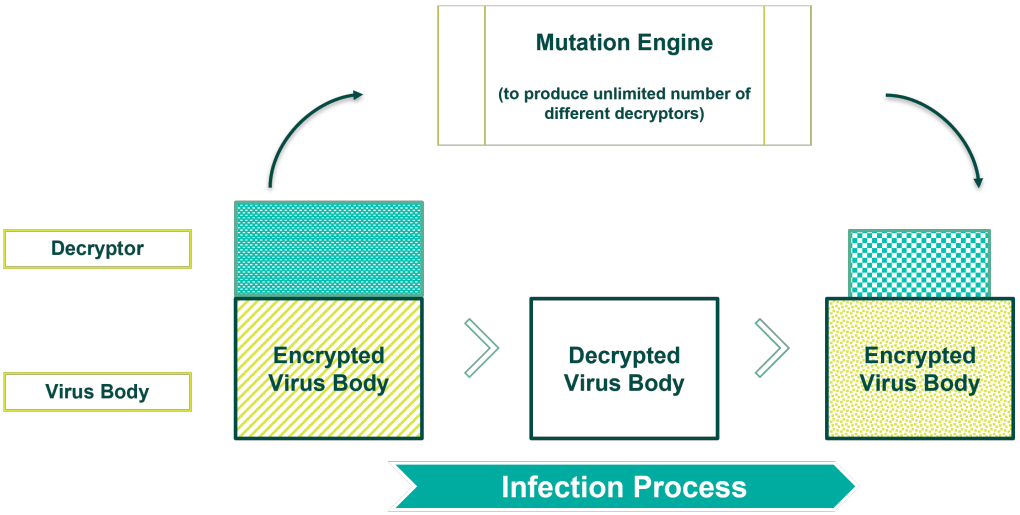


Figura 4.6. Struttura di un malware polimorfo

- **Metamorphism**: I virus che implementano tecniche metamorfiche, indicati in Figura 4.7,

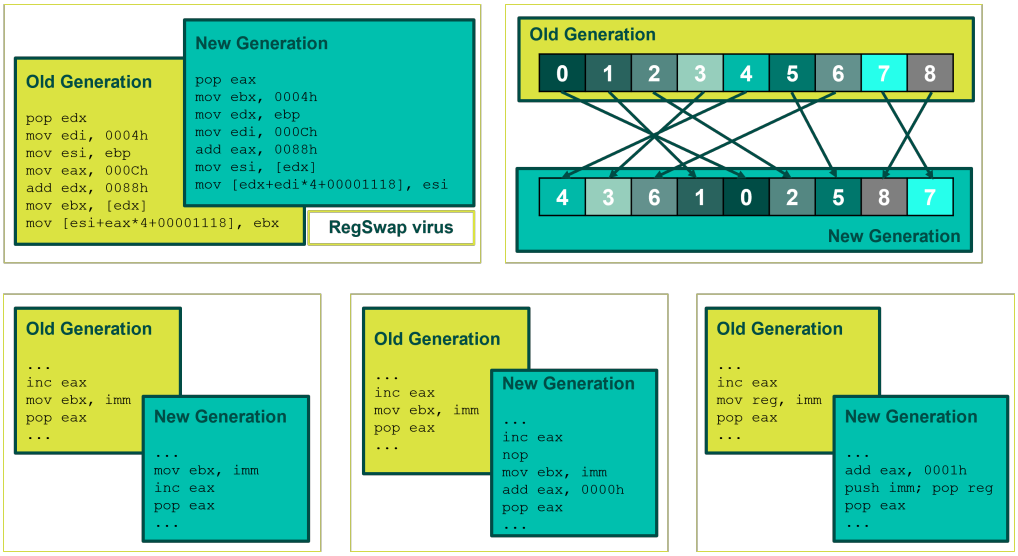


Figura 4.7. Struttura di un malware metamorfo

non hanno alcuna parte criptata e, di conseguenza, non necessitano della sezione di decrittazione. Essi basano il proprio offuscamento su di un motore di mutazione in grado di trasformare l'intero body, cosicché ogni nuova copia possa avere una diversa struttura, sequenza di codice, dimensioni e proprietà sintattiche, mantenendo lo stesso comportamento. La particolarità dei virus metamorfici risiede nel non mostrare firme di stringhe comuni tra le diverse istanze, generabili in numero illimitato. Un motore di mutazione metamorfico deve includere i seguenti componenti:

1. **Disassembler**: utile al virus per convertire il codice in istruzioni Assembly, successivamente alla localizzazione del proprio codice
2. **Code Analyzer**: responsabile di fornire al modulo Code Transformer informazioni, come la struttura e il diagramma di flusso del programma, le subroutine, il ciclo di vita delle variabili e dei registri.
3. **Code Transformer**: responsabile dell'offuscamento del codice e della modifica della sequenza binaria del virus tramite le tecniche menzionate per i virus polimorfi. I moduli precedenti sono preparativi per questa fase.
4. **Assembler**: responsabile della conversione del nuovo codice Assembly in codice binario.

A causa dell'impossibilità di rilevazione di questa tipologia di virus tramite le tecniche Signature-based, i moderni motori di ricerca si affidano a euristiche e modelli di analisi comportamentale altamente sviluppati. Il primo virus metamorfico, attore di numerosi attacchi DOS, è stato ACG nel 1998.

4.2 Morfologia e comportamento dei malware stealthy

La nuova classe di malware stealthy comprende i programmi malevoli che, in primo luogo, vengono eseguiti completamente in memoria, lasciando impronte minime o nulle sul sistema infettato in assenza di file, dannosi e non, scaricati sul disco al fine di compromettere il sistema. Questa caratteristica li rende noti anche con il nome di "DLL-injection" o "Memory-injection attack". V. Khushali distingue in [47] due categorie di malware file-less:

- **RAM-resident Fileless malware**: i malware di questa tipologia risiedono ed eseguono le proprie operazioni direttamente nella memoria volatile e vengono iniettati sfruttando

applicazioni vulnerabili [44]. . Essi riescono ad eludere la maggior parte dei controlli delle soluzioni antivirus, che vengono eseguite all'avvio di un nuovo processo. Essendo già in esecuzione sul sistema e non producendo file sul disco, vengono identificati come non sospetti dagli scanner.

- **Script-based Fileless malware**: i malware di questa tipologia ricavano il massimo rendimento da applicazioni affidabili, tra tutte quelle presenti nella suite Microsoft Office, e da strumenti di amministrazione nativi del sistema operativo Windows, come i citati Powershell e WMI, necessari al caricamento del codice dannoso direttamente in RAM.

Non a caso, in accordo con numerosi report pubblicati da fornitori di anti-malware e gruppi di ricerca, la quasi totalità di malware viene progettata al fine di attaccare le macchine con Sistema Operativo Windows [45]. Come espone Y. Keshet in [48], i ricercatori di malware C. Campbell e M. Greaber hanno coniato il termine "Living Off The Land" (LOL) per descrivere l'uso di strumenti di sistema preinstallati e affidabili per diffondere il malware. La peculiarità di questi strumenti risiede nell'impossibilità di distinzione da parte delle soluzioni antivirus standard tra una chiamata legittima e una con scopi malevoli, rappresentando un importante punto a favore per gli autori di malware. In particolare, i malware stealthy operanti sul sistema operativo Windows sono strettamente correlati all'utilizzo delle tecniche di LOL, tra tutte: i LOLBin, identificati dai file binari di Windows, i LOLLib, che sfruttano le librerie di Windows, e i LOLScript, che utilizzano script pre-installati sul sistema al momento dell'infezione.

Il ciclo di vita comune agli attacchi per mezzo di malware file-less si articola, come descritto da V. Khushali in [47] e Sudhakar in [44], e riportato in Figura 4.8, in tre fasi descrittive delle relative tecniche di evasione:

1. **Consegna del malware**: i malware vengono consegnati alle vittime tramite campagne di social engineering, in particolare phishing, per indurre gli utenti a cliccare su link o file allegati ad e-mail in entrata. Gli script presenti nei documenti sono progettati in modo da scaricare codice dannoso ed essere eseguiti direttamente nella memoria del dispositivo-vittima. A questo scopo sono utilizzate applicazioni affidabili, per evitare che i vettori vengano ispezionati dalle tecnologie di monitoraggio della sicurezza.

2. **Persistenza del malware**: i malware eseguono VBScript o Javascript dannosi, passati direttamente come riga di comando a Windows Powershell, in modo da essere memorizzati in posizioni non convenzionali associate al sistema operativo, come il registro di Windows o l'archivio WMI, ed essere successivamente eseguiti da OS Scheduler. I malware, memorizzati in punti ciechi alle soluzioni antivirus, verranno poi scompattati in memoria in fase di esecuzione, senza salvare artefatti digitali nel filesystem

3. **Esecuzione del malware**: la fase di esecuzione avviene, tramite Powershell ed altre risorse pre-installate dell'eseguibile di Windows (LOL), direttamente nella locazione di memoria RAM del processo legittimo infettato e senza rilasciare alcun file. Precisamente, gli script vengono eseguiti direttamente su Microsoft Windows, sfruttando istanze di powershell.exe, script.exe, cmd.exe e mshta.exe [47].

I virus tradizionali modificano le risorse di dati sul sistema, ad esempio aggiungendo il proprio codice malevolo alla fine di un file eseguibile, con la conseguenza di essere rilevati e catturati dalle soluzioni di monitoraggio della sicurezza [37]. I virus stealth operano diversamente in queste situazioni, sottraendo alla vista del programma di scansione le modifiche apportare ai file infetti. In genere, al momento dell'esecuzione dell'applicativo antivirus, il malware stealth entra in fase di allarme, nascondendosi nella memoria, monitorando l'attività del sistema e servendosi di diverse tecniche per occultare le modifiche effettuate su file o record di avvio. A seconda del bisogno il virus può ricorrere alla crittografia per rendere inaccessibile o illeggibile un determinato file alterato oppure mantenere una copia del file originale, cosicché il programma antivirus possa essere reindirizzato nella relativa area di memoria e non lanciare l'allarme [49]. Al fine di monitorare gli accessi ai file infetti e intercettare le richieste, i virus stealth si agganciano agli Int 13h e Int 21h del BIOS [46], come riportato in Figura 4.9.

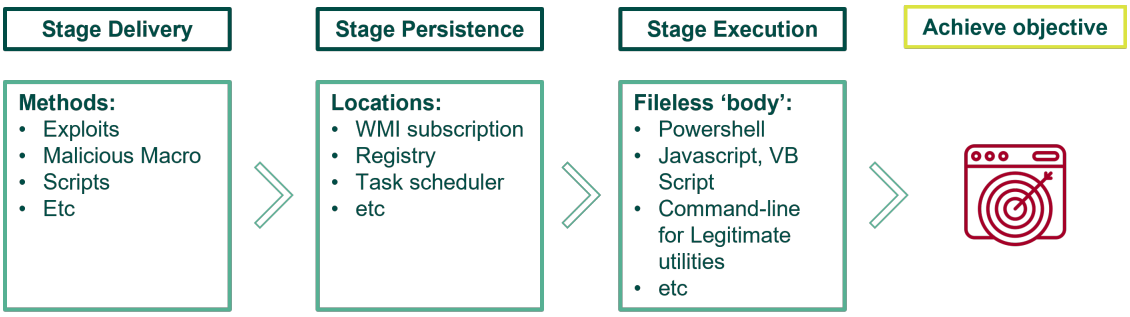


Figura 4.8. Ciclo di vita degli attacchi stealth

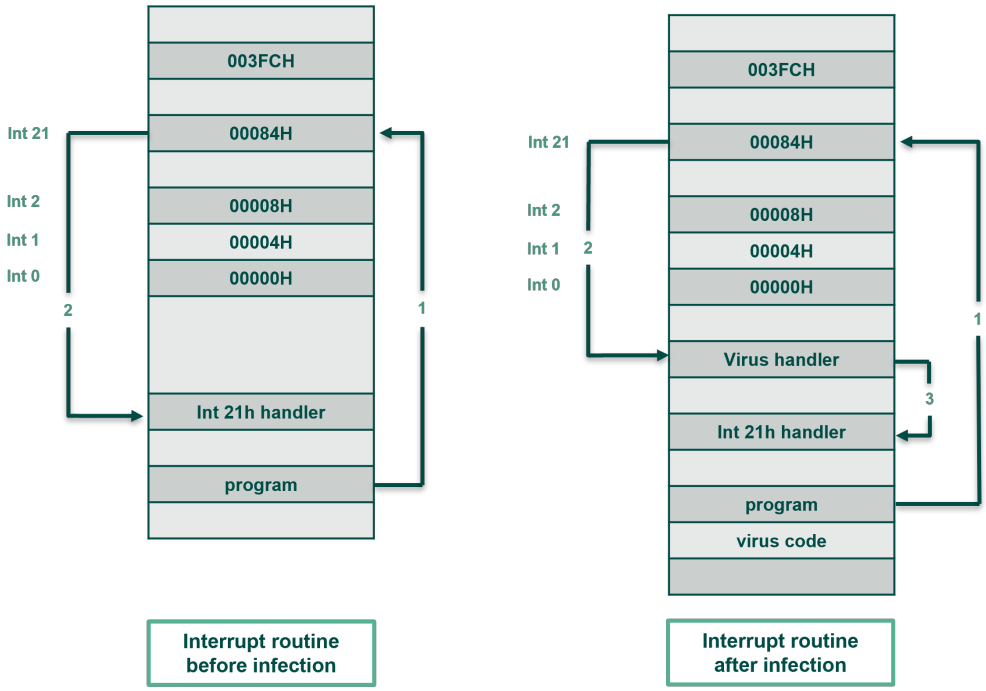


Figura 4.9. Interrupt routine prima e dopo un'infezione

L'Int 13h fornisce servizi di lettura e scrittura di dischi rigidi e floppy disk basati su settori, utilizzando l'indirzzamento CHS (Cylinder Head Sector), mentre l'Int 21h rende possibile l'implementazione della maggior parte delle funzioni e dei servizi offerti dai sistemi MSDOS. Gli interrupt software sono sfruttati dalle applicazioni per comunicare con il sistema operativo: al momento della chiamata dell'interrupt, il sistema operativo determina l'indirizzo del relativo interrupt handle, consultando una tabella interna, e procede con l'esecuzione di esso. Il virus stealth sfrutta questo protocollo per nascondere le proprie tracce: inizialmente si aggancia ad un interrupt, modificando l'indirizzo del relativo gestore nella tabella del sistema operativo con l'indirizzo della locazione di memoria contenente il codice malevolo. In particolare file e settori di avvio vengono comunemente infettati agganciandosi all'Int 21h. I virus stealth possono sottrarsi alle ispezioni dei programmi di monitoraggio rimuovendosi dal file o dal settore di avvio prima che le funzioni di interrupt vengano applicate, per poi reinserirsi ad ispezione completata. Gli autori di malware stealthy sono soliti implementare tecniche di offuscamento del codice per proteggere la proprietà intellettuale, impedire ad un'analista l'esecuzione di tecniche di reverse engineering e per discostare un'applicativo di monitoraggio di virus dalla rilevazione del codice malevolo. A questo scopo i programmi dannosi sono spesso compressi e impacchettati servendosi di vari metodi combinati con la crittografia, al fine di limitare i tentativi di analisi statica del codice. La versione "packed" del malware è contraddistinta da una strettamente minore quantità di stringhe osservabili rispetto

alla corrispettiva "unpacked", che al contempo è segno dell'utilizzo di un software di compressione, il "Packer, agli occhi degli applicativi di analisi. I programmi "Packer" non sono intrinsecamente negativi o dannosi, al contrario nascono come soluzione di sicurezza che a proteggere file, dati e applicazioni. Due tipologie di Packer comuni sono: UPX, basato su di un algoritmo open-source che non richiede memoria di sistema aggiuntiva per la decompressione, e MPRESS, originariamente progettato per comprimere i file e ridurre il tempo di avvio delle applicazioni. I malware stealth ereditano, inoltre, la capacità di occultarsi ai meccanismi di rilevazione dei programmi malevoli che sfruttano i metodi signature-based, andando a mutare la sintassi del codice presente nel main body del programma, pur mantenendo il medesimo comportamento. In particolare, i metodi Signature-based si basano sulla generazione della "firma", propriamente calcolando l'hash o collezionando i binari della sequenza di istruzioni che compongono il body del programma in analisi, per poi confrontarla interamente o parzialmente con i corrispettivi valori presenti in un database di malware conosciuti. B. Bashari Rad illustra con chiarezza in [37] una classificazione delle tecniche di "obfuscation" del codice, comunemente utilizzate dai creatori di malware al fine di modificare la firma del programma ad ogni nuova istanza del malware:

- **Inserimento di Junk/Dead code**: questa tecnica di offuscamento mediante l'inserimento di "dead code", o codice spazzatura, rappresenta la modalità più semplice per alterare la sequenza binaria di un programma, senza incorrere in una restrizione sulle relative funzionalità e conseguenze comportamentali. Nella pratica, questa tecnica di offuscamento si basa sull'aggiunta di istruzioni equivalgono ad istruzioni "no-operation" (NOP). Le istruzioni aggregate possono seguire due flussi che, in ogni modo, non hanno alcun effetto sui risultati finali dell'esecuzione: da una parte è possibile rendere complicata l'analisi statica del codice aggiungendo istruzioni che non variano lo stato dei registri, ad esempio l'aggiunta di uno 0 ad una variabile o l'assegnazione del valore di un registro a se stesso, dall'altra si inseriscono operazioni volte a mutare lo stato della macchina, il contenuto della memoria o dei registri della CPU, per poi annullare il conseguente risultato in una diversa porzione del codice, in modo da non influire nell'output finale del programma. La difficoltà di analisi dipende strettamente dalla quantità di dead code presente nel codice del programma, che influisce nel valore della relativa firma di stringa.

- **Sostituzione di variabili/registri**: questa tecnica di offuscamento permette di superare il rilevamento della firma delle stringhe, procedendo con lo scambio di registri o variabili di memoria in diverse istanze del virus. Ad esempio, una variabile memorizzata nel registro eax non modifica la funzionalità del codice se salvata, piuttosto, nel registro edx, risultando però in una firma differente. Lo svantaggio di questa tecnica risiede nell'essere vulnerabile alle scansioni wildcards-based, che costruiscono la firma pescando nelle sequenze di codice che non variano a seconda delle diverse istanze del programma.

- **Sostituzione di istruzioni**: questa tecnica di offuscamento si basa sulla commutazione di determinate istruzioni con un modello equivalente, allo stesso modo di come accade in linguistica con l'utilizzo dei sinonimi. La modalità con cui avviene questo processo pone le basi sulla sostituzione di operatori binari standard (come l'addizione, la sottrazione o gli operatori booleani) con sequenze di istruzioni funzionalmente equivalenti, ma che introducono uno strato di difficoltà ulteriore per l'analisi statica del codice. Se sono disponibili diverse sequenze di istruzioni in grado di fornire lo stesso output, ne viene scelta una casualmente [50]. Questa tecnica rappresenta una modalità di metamorfismo della sequenza binaria del codice di un programma. Tuttavia, anch'essa è soggetta alle scansioni wildcards-based, che sfruttano le porzioni di codice non mutato al fine di costruire la firma.

- **Permutazione di istruzioni**: questa tecnica di offuscamento si basa sulla permutazione di istruzioni del codice in modo da mostrare un aspetto differente nelle varie istanze generate dello stesso programma. Ad ogni modo, una sequenza di istruzioni per essere permutata e quindi riorganizzata in modo diverso, deve contenere istruzioni indipendenti e non deve risultare in una variazione delle finali funzionalità.

- **Trasposizione del codice**: questa tecnica di offuscamento commuta l'ordine del flusso di istruzioni, delle routine e dei rami del programma, senza mostrare effetti visibili sul comportamento originale. La trasformazione può essere eseguita a livello di singola istruzione o

blocco di codice, sfruttando branch di tipo "unconditional" o "conditional". Un altro metodo di trasposizione del codice sfrutta le tecniche di virtualizzazione per sottrarsi all'analisi delle soluzioni antivirus. In questo caso l'obfuscator include una macchina virtuale in grado di interpretare la logica del programma [37].

La diversità di tecniche di offuscamento implementate nei malware di tipo stealth moderni, passando dalla semplice crittografia alle avanzate tecniche di metamorfismo, ha reso totalmente inefficace il rilevamento dei virus tramite controlli meramente signature-based.

Una delle considerazioni principali su cui riflettono gli autori di malware di tipo stealth in fase di preparazione dell'attacco, consiste nel processo di persistenza in memoria che il programma deve implementare, allo scopo di mantenersi attivo anche in caso di riavvio della macchina. La persistenza in memoria consente, dunque, di infettare la macchina in un'unica operazione, evitando di ripetere la fase di accesso al sistema ad ogni boot, essendo essa lo stadio più complicato del ciclo di attacco. Sudhakar distingue in [44] tre categorie di malware fileless in base alle modalità con cui persistono all'interno della memoria volatile:

- **Memory-resident malware**: I malware appartenenti a questa classe risiedono interamente in memoria, sfruttando processi legittimi di Windows per l'esecuzione, occultandosi fino al momento dell'attivazione. Esempi di questo genere sono Code Red, SQL Slammer, Lurk e Poweliks. Tra questi, Poweliks è sviluppato in modo tale da installarsi nel Windows Registry [51], in particolare sfruttando la Run registry key, e persistere nel sistema senza rilasciare file ed essere rilevato dalle soluzioni antivirus.
- **Windows Registry malware**: I malware appartenenti a questa classe sono memorizzati nel Windows Registry. Il registro di sistema di Windows è la base di dati per la memorizzazione delle configurazioni di basso livello del sistema operativo Windows e di tutte le applicazioni installate. Gli autori di malware di questo tipo sono soliti sovrascrivere i valori delle chiavi di registro all'interno del Windows Registry in modo da puntare al proprio codice malevolo e autodistruggersi una volta aver raggiunto l'obiettivo [51]. Esempi di questa categoria sono Kovter e PowerWare.
- **Rootkits fileless malware**: i malware appartenenti a questa categoria sono installati nel kernel del sistema operativo Windows, dopo aver ottenuto i privilegi di amministratore. Un esempio di questa classe è rappresentato da Phase Bot.

I metodi più comuni, e well-known, usati dagli autori di codice malevolo per garantire la proprietà di persistenza al prodotto finale sono riassunti da A. Perlman in [52]:

- **Startup folder**: questo processo sfrutta la cartella di avvio di Windows, contenente i programmi che vengono automaticamente eseguiti all'avvio del dispositivo. Sono presenti in ogni macchina Windows due locazioni nel filesystems con lo scopo di collezionare i file da eseguire all'avvio. Una cartella è predisposta al singolo utente ed è situata in:

  C:\Users\USERNAME\AppData\Roaming \Microsoft\Windows\StartMenu \Programs\Startup

  Un'altra "startup folder" contiene i programmi che vengono automaticamente eseguiti per ogni utente della macchina ed è collocata in:

  C:\ProgramData\Microsoft\Windows\StartMenu\Programs\StartUp .

  Il malware stealth, copiato manualmente o in forma automatizzata in queste due directory, può così ottenere e mantenere la proprietà di persistenza.
- **Run and RunOnce Registry Keys**: questo processo sfrutta le registry key Run e RunOnce di Windows, che permettono l'esecuzione di programmi ogni qual volta un utente supera la fase di accesso. Questo metodo rappresenta lo standard Windows per l'esecuzione di programmi in fase di start-up: la Run key esegue tutti i programmi inclusi ad ogni avvio, mentre RunOnce key si limita esclusivamente all'esecuzione durante l'avvio successivo all'impostazione del programma. Il valore di entrambe le chiavi di registro ha una dimensione limitata a 260 caratteri. Il Windows Registry include 4 chiavi di registro Run e RunOnce,

in cui gli autori di malware stealth possono impostare un valore e forzare l'esecuzione del loro codice maligno ogni volta che un utente si collega alla macchina. Le 4 chiavi di registro sono situate propriamente in:

  - HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
  - HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
  - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce

- **Scheduled Task**: questo processo abusa del Windows Task Scheduler per pianificare l'esecuzione iniziale o ricorrente del codice malevolo all'interno del malware stealth, direttamente da riga di comando servendosi dell'eseguibile schtasks.exe, oppure accedendo da interfaccia grafica (GUI) nella sezione Strumenti di Amministrazione del pannello di controllo di Windows. A. Perlman mostra in [52] un interessante modalità con cui un APT3, un malware della famiglia dei Keylogger, ottiene la proprietà di persistenza sfruttando questa tecnica, invocando il comando:

  schtasks /create /tn "mysc" /tr C:\Users\Public\test.exe /sc ONLOGON /ru "System"

  - /tn: Specifica un nome per il task.
  - /tr: Specifica il programma o il comando da eseguire.
  - /sc: Specifica il tipo di pianificazione.
  - /ru: Esegue l'attività con i permessi dell'account utente specificato.

Queste tecniche rappresentano il punto di partenza e fungono da introduzione ai giovani hacker ai concetti di persistenza di un processo in memoria e nel tempo sono diventati facilmente rilevabili dalle soluzioni di monitoraggio della sicurezza. A. Perlman cita in [52], in aggiunta, una branca di tecniche meno comuni che abusano di processi legittimi del sistema operativo per rimanere "undetectable":

- **Image File Execution Options (IFEO)**: questo processo fornisce un meccanismo per lanciare sempre un eseguibile direttamente sotto il debugger. Gli autori di malware possono definire il loro malware come debugger principale allegando come argomento il percorso di un particolare eseguibile, in modo che il programma malevolo venga avviato ad ogni esecuzione dell'applicazione. A. Perlman cita in [52] come esempio di questa tecnica un malware keylogger in grado di registrare i tasti premuti dall'utente ogni qual volta venga avviato un particolare programma di gestione delle password. La registry key responsabile per IFEO è:

  HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ImageFileExecutionOptions\[progs_name]

- **Abuso delle Default File Associations**: questo processo sfrutta l'associazione tra una particolare estensione di un file e l'applicazione demandata alla relativa apertura. Un attaccante può sfruttare questo standard ed ottenere la proprietà di persistenza andando a modificare le citate associazioni di default di una specifica estensione, ad esempio ".txt", apponendo il percorso al proprio programma malevolo, ad esempio una Reverse Shell utile per la comunicazione con un Command&Control server. La relativa registry key è:

  HKEY_CLASSES_ROOT\[file name] \shell\open\command

- **Abuso dei programmi Screensaver**: questo processo sfrutta le caratteristiche di design dei programmi screensaver, ovvero dei Portable Executable (PE) file distinti da un'estensione ".scr" ed eseguiti in fase di inattività dell'utente. Un attaccante può ottenere la proprietà di persistenza per il suo prodotto malevolo manipolando le impostazioni dell'eseguibile di screensaver accessibili alla registry key:

  HKCU\Control Panel\Desktop

  In particolare i parametri di configurazione utili allo scopo sono:

- **ScreenSaveActive**: flag di abilitazione dello screensaver.
- **ScreenSaveTimeout**: parametro temporale in secondi, descrivente il tempo di inattività dell'utente dopo il quale eseguire il programma screensaver.
- **SCRNSAVE.exe**: contiene il percorso del PE file da eseguire.

Un intrigante esempio citato da A. Perlman in [52] circa questa tecnica innovativa per raggiungere la persistenza è riprodotta da un processo di crypto-mining configurato per minare moneta virtuale durante le sessioni di inattività dell'utente, per poi terminare automaticamente e non essere rilevato dal Task Manager alla ripresa della sessione

Una tecnologia largamente utilizzata dai malware stealth e altamente descrittiva delle loro modalità di comportamento è rappresentata dai rootkit, largamente analizzati da E. Rudd in [45]. Un rootkit è un malware che, comunemente, mantiene file dannosi sul disco sfruttando al contempo tecniche di "hooking" e di mutazione del codice al fine di occultare la visibilità dei file e l'esecuzione dei processi ai moderni sistemi di rilevazione anti-minaccia. I rootkit sono pensati in fase di design con il duplice scopo di consentire l'accesso continuo al sistema informatico infettato e mantenere la presenza in modalità stealth, mascherando l'utilizzo di processi, risorse, connessioni di rete e chiavi di registro. Gli attacchi basati sull'utilizzo della tecnologia rootkit possono essere pensati, su suggerimento di E. Rudd in [45], come "attacchi man-in-the-middle (MITM) tra i diversi componenti del sistema operativo" e sono classificabili in quattro generazioni, a seconda delle tecniche comportamentali implementate per evadere la rilevazione dagli strumenti di monitoraggio:

- **Tipo 1, File di sistema dannosi su disco**: i rootkit di prima generazione si occultavano come file di processo del sistema residenti sul disco ed avevano come obiettivo primario l'ottenimento dei permessi di amministratore tramite privilege escalation. Un vantaggio significativo consisteva nell'essere semplici da installare e sopravvivere ai reboot per la presenza di file sul disco, arma a doppio taglio che li rendeva al contempo semplici da rilevare e rimuovere tramite il confronto con gli hash o i checksum dei file di sistema originali.
- **Tipo 2, Hooking e reindirizzamento in memoria dell'esecuzione del codice**: i rootkit di seconda generazione si affidano a tecniche di hijacking della memoria di un processo al fine di modificare il relativo flusso di esecuzione e saltare al codice malevolo. Il termine "hooking" indica le tecniche di alterazione di puntatori a librerie e funzioni o di sovrascrittura di codice (inline function patching). In genere l'hooking non nasce come tecnica dannosa per un sistema, ma in supporto a processi legittimi come l'hot patching, il monitoraggio delle prestazioni e il debugging. Gli autori di malware sfruttano le potenzialità offerte dall'hooking per l'esecuzione del codice malevolo precedentemente o successivamente l'esecuzione di una chiamata legittima del sistema operativo, rimanendo in questo modo nascosti alle soluzioni antivirus per via della difficoltà di differenziare l'uso legittimo da quello malevolo. Ad ogni modo, la modifica del comportamento di una funzione incrementandone la quantità di codice comporta il rilascio di un'impronta, rilevabile dagli applicativi di monitoraggio delle minacce.
- **Tipo 3, Direct Kernel Object Manipulation (DKOM)**: I rootkit di terza generazione consistono nell'alterazione dell'integrità del kernel, modificando le strutture dinamiche impiegate nelle operazioni di contabilità delle risorse, e vengono indicati con l'acronimo DKOM. L'esempio classico è il "process hiding", possibile sulla maggioranza dei sistemi operativi e basato sullo sfruttamento del disaccoppiamento tra le strutture dati dinamiche utilizzate dallo scheduler, per tracciare i processi, e quelle impiegate per la contabilità delle risorse. Nel kernel Windows NTOS lo strato kernel1 è responsabile dello scheduling dei diversi thread, mantenendo una lista circolare doppiamente concatenata di strutture dati KTHREAD. Nel contempo lo strato esecutivo del kernel è responsabile della gestione delle risorse e mantiene una lista circolare doppiamente concatenata di strutture dati EPROCESS. Un rootkit, con controllo sulla memoria kernel, agisce su quest'ultima struttura dati, disaccoppiando il nodo EPROCESS desiderato, cosicché il processo non sarà più visibile nè a livello esecutivo, nè dalle API di Windows. Esso rimarrà a disposizione solamente dello

scheduler, che continuerà ad assegnare le risorse necessarie al funzionamento. Questa tipologia di rootkit è tanto complicata da implementare quanto estremamente difficile da rilevare, in quanto puntano a modificare strutture dati di per sé dinamiche. Inoltre essi, agendo nel kernel-space, sono immuni agli antimalware che agiscono in user-mode.

- **Tipo 4, Cross Platform Rootkit e Rootkit in Hardware**: i rootkit di quarta generazione sono attualmente una Proof of Concept (PoC) [45] e si distinguono per operare a basso livello e non essere rilevabili dagli applicativi di contromisura convenzionali, in quanto collocati in un livello inferiore al sistema operativo. In particolare essi eseguono a livello di virtualizzazione, nel BIOS e nell'hardware.

I malware stealth che utilizzano le tecniche della seconda generazione di rootkit [45] sono indubbiamente i più frequenti. I metodi legati allo sfruttamento dell'API Hooking, propriamente il processo con cui un'applicazione intercetta una chiamata API tra due altre applicazioni, sono parte delle fondamenta su cui si basa il funzionamento del Sistema Operativo, consentendo a sviluppatori e ricercatori di estendere le funzionalità del proprio codice e di eseguirne l'analisi dinamica [53] [54]. Il sistema di agganciamento, che si mostra alterando i componenti della memoria di un processo e consentendo la modifica di funzioni API, nasce quindi in funzione di usi legittimi. Il sistema operativo Windows supporta differenti tipologie di hook, in grado di fornire l'accesso a un diverso aspetto del meccanismo di gestione dei messaggi. Ad ogni tipo di hook è associata una "hook chain": una lista di puntatori a funzioni di callback, definite dall'applicazione e chiamate "hook procedure" [55]. Una procedura di hook viene installata e posta in cima ad una catena di hook mediante l'API SetWindowsHookEx. Ad ogni occorrenza di un messaggio associato ad uno specifico tipo di hook, il sistema rimbalza alla "hook procedure" referenziata nella relativa "hook chain". Lo scorrimento ordinato nella catena, terminata una specifica procedura di hook, avviene tramite l'API CallNextHookEx: in questo modo l'evento viene passato alla prossima procedura di hook [55], come mostrato in Figura 4.10.

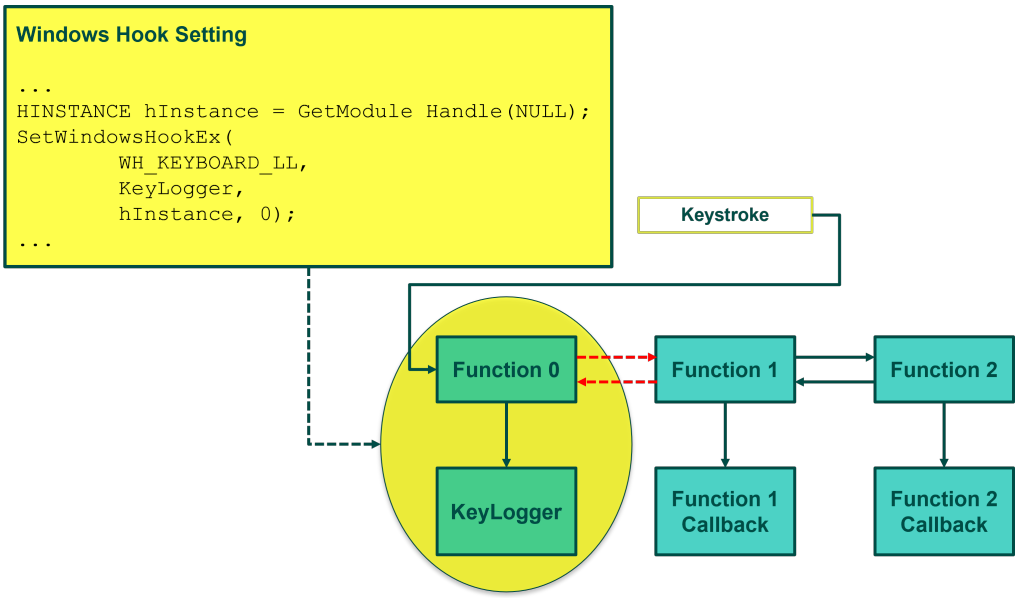


Figura 4.10. Hook chains

Si distinguono due tipologie di hook, sulla base dei thread monitorati: i "global hook", che monitorano i messaggi di tutti i thread con Windows Desktop comune al thread chiamante, i "thread-specific hook", è chiamata esclusivamente per monitorare il thread associato. La tecniche che sfruttano i Windows Hook, al contempo, si mostrano come bene prezioso agli occhi degli intrusori, permettendo il controllo del flusso di esecuzione di un programma e il relativo reindirizzamento verso il codice dannoso prodotto. Si apre al malware un vero e proprio mondo di possibilità, che in particolar modo si riflettono nel monitoraggio degli input utente (MITRE T1056.004, [56]), al fine di sottrarre alla vittima informazioni sensibili, quali password o credenziali

bancarie, a monte della trasmissione mediante connessioni di rete. Ad esempio, un attaccante può monitorare gli eventi di input della tastiera, inseriti nella coda di input di un thread, agganciandosi globalmente alla procedura di hook WH_KEYBOARD_LL o monitorare gli input da tastiera WM_KEYDOWN e WM_KEYUP, inviati a una coda di messaggi, agganciando l'hook WH_KEYBOARD_LL [55]: il trojan Trickbot [56], diffusosi nel 2016 nel settore bancario al fine di catturare le credenziali RDP agganciandosi all'API CredEnumerateA [54], è un valido esempio di questa tecnica. Altri casi limitano le proprie funzionalità al fine di rimanere in modalità "stealth" nel sistema, alterando i risultati restituiti dalle chiamate API del filesystem o del Windows Registry [53]. Difatti, l'hooking può essere usato per vari scopi, persino per occultare processi dannosi e porte di rete Command&Control (C&C), agganciando funzioni come NtQuerySystemInformation [54]: NetRipper si presta a questo scopo [57].

E. Rudd in [45] descrive in profondità le modalità con cui i malware di tipo stealth sfruttano i processi messi a disposizione dalle tecniche di hooking, distinguendo tre classi di hook, a seconda dei permessi che utilizzano:

- **User-Mode Hooking**: questa tipologia di attacco viene sfruttata dai malware di tipo stealth che operano in user-mode e consiste nel porre codice malevolo all'interno dello spazio di indirizzamento di un processo specifico. Queste tecniche sfruttano in particolar modo le API di Windows, progettate come "Dynamically linked library" (DLL) al fine di "migliorare l'utilizzo delle risorse e fornire un'interfaccia organizzata per richiedere le risorse del kernel dallo spazio utente" [45]. Le DLL sono PE file contenenti funzioni, accessibili tramite tabelle di puntatori a funzione, e permettono la condivisione di codice tra programmi diversi. Le tabelle di puntatori a funzione, messe a disposizione da ogni DLL, sono denominate Export Address Table (EAT): i programmi possono importare le funzioni in esse contenute, evitando di memorizzare codice aggiuntivo, associando la relativa entry inclusa nell' EAT all'interno della Import Address Table (IAT) specifica del programma. L'azione di caricare una DLL all'interno dello spazio di indirizzamento di un particolare processo viene definita "DLL injection" ed, essendo comunemente a disposizione delle API di Windows, è tanto semplice rilevare quanto ardua da distinguere dalle chiamate legittime. D. Lukan distingue tre tecniche principali per ottenere DLL Injection in [58], [59], [60]:

  - **AppInit**: viene sfruttata la registry key "AppInit_DLL" in modo da includere il percorso alla DLL malevola, contenente la funzione DLLMain e opzionalmente un payload da eseguire. Settando a "1" il valore della chiave di registro "LoadAppInit_DLLs", la DLL precedentemente definita verrà caricata, e la funzione DLLMain eseguita, ad ogni caricamento della "user32.dll" da parte di un processo. In particolare, la "user32.dll" è presente in diverse applicazioni, in quanto responsabile di funzionalità primarie per l'interfaccia utente [58].
  - **SetWindowsHookEx API**: viene sfruttato il meccanismo delle hook chain per iniettare una specifica DLL all'interno della catena, tramite l'API SetWindowsHookEx. Al verificarsi del primo evento scatenante la catena, la dll viene iniettata nello spazio di indirizzamento del processo, che automaticamente esegue la funzione DLLMain. Questa tecnica è comune nei malware di tipo stealth della famiglia dei keylogger ed è mostrata in Figura 4.11[59].
  - **CreateRemoteThread API**: viene sfruttata la funzione CreateRemoteThread, al fine allocare la DLL direttamente nello spazio di indirizzamento di un thread creato allo scopo e appartenente ad un processo arbitrario. Un programma di supporto chiama in sequenza la OpenProcess, che torna il gestore del processo vittima, e la GetProcAddress, per ottenere l'indirizzo della funzione LoadLibraryA interna alla kernel32.dll. Al fine di inserire il percorso della dll da iniettare nello spazio di indirizzamento del processo vengono chiamate in ordine le API VirtualAllocEx, utile ad allocare un intervallo di memoria virtuale all'interno dello spazio di indirizzamento processo vittima, e WriteProcessMemory, chiamata per porre il percorso della dll malevola all'interno dello spazio allocato. Queste azioni sono preparative alla chiamata della CreateRemoteThread, che rimbalza nella funzione LoadLibraryA, presente all'interno dello spazio di indirizzamento del processo vittima, con la quale la dll infetta viene iniettata. Il processo è mostrato in Figura **??**.

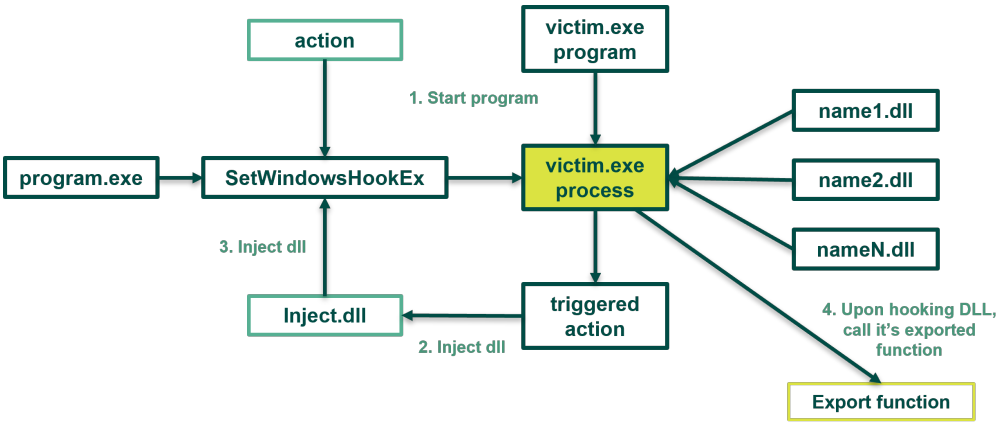


Figura 4.11. DLL injection mediante SetWindowsHookEx API

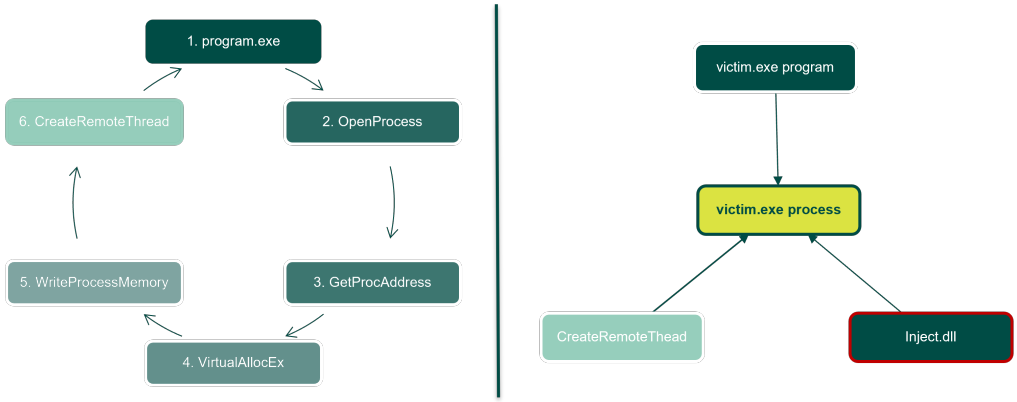


Figura 4.12. DLL Injection mediante CreateRemoteThread API

Due specifiche modalità di user-mode Hooking utilizzate dai malware di tipo stealth sono: l'IAT Hooking e l'inline function patching [45].

L'IAT hooking è una tecnica volta alla modifica del valore dei puntatori a funzione all'interno della IAT in modo da puntare al codice malevolo, come mostrato in Figura 4.13. Nella maggioranza dei casi il codice malevolo viene agganciato ad API del sistema operativo, che vengono copiate, cosicché l'esecuzione del malware sia precedente o successiva al completamento della funzione originaria. Il risultato della funzione originaria viene in questo modo modificato, ad esempio è possibile filtrare file con specifici identificatori Unicode mediante chiamate ad API come FindFirstFile o FindNextFile, che vengono introdotte nel codice e contribuiscono ad occultare i movimenti del malware stealth, essendo eseguite da processi legittimi infetti [45]. Una limitazione della tecnica dell'IAT Hooking è individuata nella difficoltà di agganciare le entry all'interno della IAT. La causa principale risiede nel fatto che le librerie dinamiche possono essere caricate in momenti differenti rispetto al lancio dell'eseguibile, propriamente al relativo caricamento (load time) o durante l'esecuzione (load time). L'IAT risulta così sprovvista delle entry necessarie all'hooking. Inoltre, il processo descritto da D. Lukan in [60], che si conclude con la chiamata alla funzione LoadLibraryA, non comporta la creazione di entry all'interno della IAT.

D'altra parte la tecnica dell'Inline Function Patching, anche detta "detouring", propone una diretta modifica del codice in memoria, mediante la sovrascrittura di un frammento di codice, memorizzato al termine del codice dannoso, con un'istruzione di unconditional jump che collega direttamente alla prima istruzione del malware. L'esecuzione della porzione di codice sovrascritta ("Stub") è successiva a quella del codice dannoso e può terminare con un'istruzione di jump back verso il precedente punto di interruzione del codice originale,

cosicché esso venga eseguito ("trampolining). La maggioranza dei malware stealth che utilizzano la tecnica del detouring introducono l'istruzione di salto nella prima porzione di byte della funzione agganciata, propriamente nei primi 5 bytes, che corrispondono a istruzioni NOP per via di uno standard dei compilatori Windows x86 volto a limitare strani comportamenti nelle funzioni originali, dovuti all'utilizzo immediato del detouring.

Riassumendo, il vantaggio delle tecniche di user-mode Hooking risiede nella difficoltà di essere classificati come malevoli dalle soluzioni di monitoraggio delle minacce, mentre la limitazione principale sta nell'essere facilmente rilevabili.

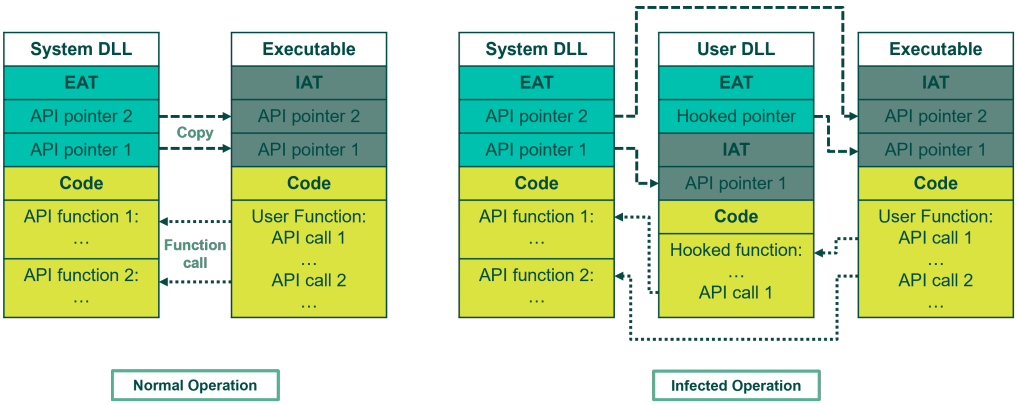


Figura 4.13. IAT Hooking

- **Kernel-Mode Hooking**: questa tipologia di attacco viene sfruttata dai malware di tipo stealth che operano in kernel-mode e consiste nell'iniezione di codice per mezzo di kernel hook, implementati mediante device driver. Questa classe risulta più complicata da rilevare, rispetto ai corrispettivi agenti in user-mode, per via della locazione dei kernel hook all'interno dello spazio di memoria riservato alla memoria kernel. Quest'ultima si pone in cima allo spazio di indirizzamento di un processo e non può essere acceduta in assenza di speciali permessi. G. Hoglund in [61] riconosce tre strutture che possono essere "agganciate" lato kernel: la System Service Descriptor Table (SSDT), la Interrupt Descriptor Table (IDT) e l'I/O Request Packet (IRP).
  - **SSDT hooking**: La SSDT è una struttura di dati della memoria del kernel di Windows che contiene i puntatori alle system call [45] ed è strettamente correlata alla System Service Parameter Table (SSPT), che detiene per ogni system call il numero di byte richiesti dai relativi argomenti. In particolare, SSDT e SSPT sono puntate dalla KeServiceDescriptorTable. Un esempio del processo di Hooking della SSDT è descritto da G. Hoglund in [61], nel quale viene agganciata la system call NTQuerySystemInformation per puntare ad una shellcode, che agisce filtrando le strutture ZwQuerySystemInformation corrispondenti ai processi in base ai loro identificatori di stringhe unicode e ottenendo la possibilità di occultare processi in esecuzione mediante la modifica dei relativi puntatori presenti in queste strutture.
  - **IDT hooking**: la IDT è una struttura di dati della memoria kernel di Windows che contiene i puntatori alle callback chiamate in risposta ad uno specifico interrupt hardware o software. Questa tipologia di hook non può filtrare e restituire dati, essendo gli interrupt forzati ad avere valore di ritorno "void". Gli IDT Hooking sono in grado esclusivamente di negare una richiesta di interrupt. I sistemi multiprocessore hanno complicato seriamente questa tipologia di hook poichè ogni CPU possiede una propria tabella IDT. Al fine di non limitare l'impatto un malware stealth che desidera sfruttare questa classe di agganciamenti kernel deve essere in grado di attaccare le tabelle IDT presenti in ogni CPU.
  - **IRP hooking**: la IRP è una struttura di dati della memoria kernel che contiene i riferimenti ai device driver, in Windows trattati come oggetti "device object". Questi ultimi, in rapporto 1:N con i device driver, sono in grado di rappresentare virtualmente

dispositivi hardware fisici, come bus, o dispositivi software come componenti di un antivirus atto a monitorare il kernel. Il modulo I/O Manager è responsabile della gestione della comunicazione tra il kernel e i device driver, resa possibile inviando al driver una richiesta I/O, all'interno della struttura dati IRP, e il puntatore al device object. I device object puntano ai driver object, ovvero particolari tipi di dll. Un malware di tipo stealth che agisce sfruttando questa tipologia di hook è in grado di modificare il puntatore a queste strutture in modo da puntare ad una shellcode [45].

- **Hybrid Hooking**: questa tipologia di attacco viene sfruttata dai malware di tipo stealth al fine di estremizzare la difficoltà di rilevazione da parte dei programmi anticirus, implementando user-mode hook all'interno della memoria kernel. Le motivazioni dietro a questa tecnica risiedono nella facilità di rilevare IAT hook per via dell'allocazione di memoria aggiuntiva nello spazio di indirizzamento per l'iniezione della dll infetta. G. Hoglund in [61] descrive una tecnica di agganciamento ibrido che non necessita dell'allocazione di memoria nello spazio utente per sfruttare gli agganciamenti della tabella IAT. La modifica dell'IAT avviene mediante l'exploiting di una vulnerabilità nell'API PsSetLoadImageNotifyRoutine, una "routine di supporto alla modalità kernel che registra le funzioni di callback del driver da richiamare ogni volta che un'immagine PE viene caricata in memoria" [45]. La callback viene chiamata tra il caricamento in memoria e l'esecuzione all'interno del contesto del processo dell'immagine PE: agendo sull'immagine PE in memoria l'attaccante è in grado di modificare l'IAT Per eseguire il codice dannoso senza allocazione di memoria nello spazio utente o iniezione di DLL, G. Hoglund suggerisce in [61] di sfruttare una porzione di memoria condivisa tra lo spazio kernel e lo spazio user. Questa porzione di memoria, presente al fine di gestire velocemente le chiamate SYSENTER e SYSEXIT, è scrivibile esclusivamente dal kernel ed è presente all'indirizzo 0xFFDF0000 nello spazio kernel e mappata in read-only nello spazio user, accessibile ai processi, all'indirizzo 0x7FFE0000. Il kernel utilizza circa 1KB di questa specifica pagina, lasciando i restanti 3KB a disposizione degli attori malevoli per la scrittura di codice dannoso. In questo modo si necessita nello spazio utente esclusivamente di un puntatore al codice dannoso sovrascritto nello spazio kernel, consentendo al malware stealth di agganciare la IAT senza allocare memoria nello spazio user o eseguire una dll Injection.

La minaccia dei virus polimorfi ha portato le società proprietarie delle soluzioni antivirus, Kaspersky in primo luogo, alla creazione di motori di emulazione interni a macchine virtuali [45]. Questi rendono possibile l'analisi del codice dannoso e i relativi controlli, successivamente all'esecuzione in memoria, dove il malware viene decrittato e il corpo del codice diviene accessibile. In risposta, i malware stealth possono essere implementati con moderne tecniche di anti-emulation in grado di rilevare l'ambiente virtualizzato e, nel caso, mutare il proprio comportamento al fine di eludere il rilevamento. In particolare i malware Kelihos e Nap sfruttano le API SleepEx e NtDelayExecution per ritardare l'esecuzione fino al termine delle analisi antivirus da parte dell'emulatore. Una tecnica più aggressiva viene implementata dal malware PushDo, in grado di annullare le routine di monitoraggio della sandbox, sfruttando l'API PspCreateProcessNotify.

Un'ultima caratteristica rilevante è rappresentata dalle tecniche di "targeting", che consistono nella diffusione ed esecuzione del malware solo in ambienti mirati, rimanendo inosservato o modificando il proprio comportamento in caso di mismatch delle configurazioni del sistema infettato con quelle impostare nel corpo del programma malevolo. Alcuni malware, come Gauss [62], ottengono questa caratteristica crittando dei moduli con cifrario RC4 mediante una chiave ricavata dalle specifiche del sistema che vogliono infettare, cosicché delle funzionalità rimangono sconosciute e inutilizzate in caso di intrusione in sistemi diversi dall'obiettivo. Stuxnet, Duqu e Flame sono esempi di malware famosi che sfruttano questa tecnica per prolungare il tempo con cui il virus può rimanere inosservato nel sistema infettato.

Rilevanti problematiche comuni associate agli attacchi stealth sono riportate da R. Awaati [49]:

- crash improvvisi del sistema e tempi di riavvio prolungati.
- rallentamento delle prestazioni del sistema.

- comparsa di icone non identificate sullo schermo del computer.
- il sistema può accendersi o spegnersi senza l'intervento dell'utente.
- problemi con i dispositivi di stampa.

4.3 Descrizione delle tecniche di rilevazione esistenti dei malware stealthy

La caratteristica principale dei malware di tipo stealth risiede nell'applicare procedure di occultamento, senza lasciare tracce della relativa esecuzione, al fine di eludere i controlli di rilevamento delle soluzioni antivirus.

Lo sviluppo delle tecniche di monitoraggio deve procedere di pari passo all'evoluzione delle tecniche stealth, in modo da prevenire che i dispositivi che le implementano vengano infettati. L'infezione, e la conseguente corruzione, di dispositivi che affidano la gestione della sicurezza ad un applicativo di monitoraggio anti-minaccia, a pagamento o in versione gratuita, comporta un serio danno reputazionale e finanziario alla la relativa software house.

Tecniche di rilevamento generiche dei malware stealth operano confrontando i primi byte di un possibile file infetto con i corrispettivi interni alla versione "clean", contenuta in un'istantanea verificata e non-infetta di un sistema informatico [46]. Sono generalmente i primi byte di un file, difatti, ad essere commutati con istruzioni volte a saltare direttamente all'indirizzo di partenza del codice dannoso, come descritto nella precedente sezione. Una modalità di controllo dell'integrità del sistema consiste nel calcolo del checksum del sistema pulito, tramite checksum CRC o crittografici, per poi procedere al confronto con il relativo valore calcolato sul file in analisi. Questo controllo si basa sull'osservazione che un programma eseguibile sia un file statico e che, quindi, la relativa modifica possa far pensare ad un'intrusione. Da questa riflessione è semplice dedurre l'ammontare di False Positive derivanti dall'analisi di eseguibili "self-modifying" o in caso di ricompilazione [46].

Al fine di semplificare l'analisi dei malware stealth che sfruttano la crittografia per rimanere inosservati, come i virus polimorfi, le soluzioni di monitoraggio sono solite implementare un "motore di decrittazione", in grado di rilevare dal codice del programma dannoso l'algoritmo di crittatura utilizzato e tentare la decifrazione del virus [46].

Il processo di rilevamento dei malware di tipo stealth, secondo V. Khushali [47], consiste principalmente in due fasi: analisi e rilevamento.

L'analisi del malware, come descrive K. Baker in [63], è "il processo di comprensione del comportamento e dello scopo di un file o di un URL sospetto" e supporta il rilevamento e la riduzione della potenziale minaccia. V. Khushali riassume in [47] le modalità di analisi più comuni interne al dominio dei malware stealth:

- **Static Analysis**: le tecniche di analisi statica prevedono l'indagine di un PE file scompattato e decompresso senza considerare la fase di esecuzione. La ricerca di evidenze malevole avviene per mezzo di programmi in grado di identificare gli attacchi mediante ricostruzione e analisi di modelli comuni. Gli strumenti di analisi statica permettono, ad esempio, di visualizzare istruzioni assembly, nomi e intestazioni di file, hash, indirizzi IP, che conducono l'analista ad etichettare uno specifico file come malevolo. Strumenti di questo tipo includono disassembler e analizzatori di rete. La limitazione di questa modalità consiste nell'impossibilità di rilevare i comportamenti malevoli attuati dal malware a runtime.
- **Dynamic Analysis**: le tecniche di analisi dinamica prevedono l'esecuzione e il monitoraggio del potenziale file infetto in una "sandbox", propriamente un ambiente controllato, come una macchina virtuale o un emulatore. Questa modalità permette di analizzare il comportamento di file sospetto senza il rischio di infettare i propri sistemi o contribuire alla diffusione nella rete. La limitazione di questa modalità risiede nelle tecniche di anti-emulation implementate dai malware, che dissociano il programma da comportamenti malevolo al riconoscimento di un ambiente emulato.

- **Hybrid Analysis**: le tecniche di analisi ibrida prevedono l'analisi del file sospetto con l'ausilio di entrambe le modalità statico e dinamica, al fine di combinare i vantaggi di una e dell'altra ed ottenere un completo set di evidenze digitali, descrittive del comportamento del malware in analisi. L'analisi ibrida è la modalità più vantaggiosa per il rilevamento di minacce sconosciute.
- **Memory Analysis**: le tecniche di analisi della memoria volatile di un sistema prevedono il rilevamento di malware mediante l'investigazione di un memory dump, in modo da ottenere informazioni circa i programmi in esecuzione, il sistema operativo, le connessioni di rete e lo stato attuale della macchina. Il vantaggio di queste tecniche consiste nella possibilità di monitorare il codice all'esterno dello scope della funzione, l'agganciamento di API, l'iniezione di dll e i processi nascosti.

Il processo di rilevamento dei malware di tipo stealth deve considerare aspetti comuni dei comportamenti ad essi associati: in primo luogo il processo dannoso utilizza tecniche di privilege escalation per ottenere i privilegi di amministratore ed essere eseguito in memoria, sfruttando le potenzialità offerte da Poweshell. Il sistema dovrebbe, quindi, monitorare gli eventi essenziali eseguiti per mezzo di Powershell, come l'esecuzione di comandi remoti, la modifica dei privilegi e l'esecuzione di programmi malevoli in memoria. Sudhakr in [44] descrive la possibilità di implementare delle tecniche di rilevamento "rule-based", sviluppate in base all'analisi della diffusione di malware che eseguono programmi come cmd.exe o powershell.exe e sono impacchettati da applicazioni della suite Microsoft Office, come winword.exe, excel.exe e powerpnt.exe. Comunemente questi malware si propagano nella rete al fine di raggiungere la vittima target. Un esempio di queste regole, che possono essere implementate su un browser al fine di limitare l'esecuzione di Powershell e CommandPrompt da parte di applicazioni malevole, è rappresentato dai seguenti casi:

- **Case 1**: ProcessName: cmd.exe AND (parentName: winword.exe OR parentName: excel.exe OR parentName: powerpnt.exe OR parentName: outlook.exe) AND chilprocessName: powershell.exe
- **Case 2**: ProcessName: cmd.exe AND (parentName: winword.exe OR parentName: excel.exe OR parentName: powerpnt.exe OR parentName: outlook.exe)
- **Case 3**: ProcessName: powershell.exe AND parentName: winword.exe
- **Case 4**: ProcessName: powershell.exe AND filemodCount: [1000 to *]

Il traffico e le connessioni di rete, unitamente alla sospetta modifica dei valori all'interno delle chiavi di registro di Windows, sono ulteriori caratteri primari da osservare per il monitoraggio delle minacce stealth. In particolare, Sudhakar in [44], suggerisce il monitoraggio di particolari eventi all'interno del Windows Security Event Log che potrebbero essere indicatori di attività malevole:

- **Event ID 4688**: indica l'evento di creazione di un nuovo processo. Si necessita il monitoraggio di ogni evento creato con Powershell identificato come processo padre.
- **Event ID 7040**: indica l'evento di modifica delle impostazioni di start-up per uno specifico servizio con valori "disable" e "demand". Si necessita il monitoraggio di questo evento, qualora registrasse la disattivazione di uno specifico servizio
- **Event ID 10148**: indica l'evento responsabile per l'impostazione di specifici indirizzi IP e Porte per l'ascolto delle richieste relative al Windows Remote Management.

La valutazione di un sistema di riconoscimento dei malware di tipo stealth deve tener presente del compromesso tra "precisione" e "richiamo" del sistema: per precisione si intende la percentuale di campioni che il sistema ha etichettato giustamente come dannosi, mentre il richiamo indica la percentuale di campioni dannosi di uno specifico tipo che sono rilevati correttamente come dannosi

di quel determinato tipo [45]. Nel contempo l'aumento del richiamo tende a diminuire la precisione e viceversa: il giusto compromesso dipende dalla modalità di applicazione.

E. Rudd in [45] distingue diverse tecniche component-based, che mirano a proteggere l'integrità delle aree del sistema conosciute per la loro vulnerabilità agli attacchi di tipo stealth:

- **Rilevamento degli hook**: questa classe di tecniche consiste nella rilevazione e prevenzione di malware di tipo stealth che fanno uso dei metodi di hooking. Un efficiente approccio in questa direzione consiste nell'agganciare preventivamente punti di attacco comuni, come funzioni API usate per iniettare DLL all'interno del contesto di un processo-vittima. Una limitazione di questo approccio risiede nella difficoltà di agganciare la giusta funzione. Per questo motivo queste tecniche vengono eseguite con l'ausilio di scansioni di firma, in grado di lanciare un allarme nel caso venga rilevata una firma sospetta su di un codice agganciato, ed euristiche. Un esempio è rappresentato da VICE, una tecnica di hook detection pubblicata da G. Hoglund [64] e basata sulla scansione della memoria orientata verso le strutture SSDT, IAT e IDT, alla ricerca di unconditional jump verso valori di memoria posti al di fuori di scope accettabili. VICE trova, a sua volta, delle difficoltà sulla rilevazione dei detouring hook. In particolare le funzioni SSDT che includono unconditional jumps nei primi byte sono indicatori di agganciamento, mentre i controlli sull'IAT necessitano un context switch all'interno dello specifico processo al fine di enumerare gli intervalli di indirizzi in cui sono caricate le dll e controllarne la validità. Un approccio differente mira a rilevare la presenza di hook malevoli, andando a comparare le sezioni di memoria di driver e dll con le corrispettive immagini PE sul disco. Supponendo che le sezioni siano in read-only, le due versioni nella maggior parte dei casi dovrebbero risultare equivalenti, ad esclusione delle aree di codice self-modifying. Se, inoltre, l'immagine sul disco risulta nascosta, questo approccio identifica prontamente la presenza di un rootkit nel sistema. Un esempio è rappresentato da System Virginity Verifier [65]. Un approccio indiretto per la rilevazione degli hook viene proposto da PatchFinder 2 [66], che sfrutta il conteggio del numero di istruzioni eseguite in seguito alla chiamata di una determinata API e lo confronta con il corrispettivo "Clean Value". Questo metodo si basa sull'osservazione che nell'ottica degli hook, un malware stealth aggiunge sempre delle istruzioni al codice originario. Le limitazioni delle tecniche di rilevazione degli hook risiedono nella facilità di segnalazione di hook benigni e lanciare falsi allarmi. Inoltre, queste tecniche non sono utili per la rilevazione di attacchi mediante tecnica DKOM.

- **Rilevamento cross-view**: questa classe di tecniche consiste nella rilevazione di malware di tipo stealth mediante l'osservazione e la comparazione dei valori di ritorno di chiamate a funzione equivalenti in diverse modalità. Un comune approccio è quello di confrontare l'output restituito da un API con la corrispondente chiamata basso livello, progettata per il medesimo scopo. Se si ottengono valori diversi, si attesta la presenza di un malware. Un esempio può essere la valutazione della coerenza tra i risultati tornati dalle API FindFirstFile e FindNextFile con i corrispettivi restituiti formulando un'apposita query al disk controller [45]. Questa tecnica risulta particolarmente efficiente, considerando il fatto che le applicazioni legittime di hooking raramente modificano l'output delle relative funzioni API. Questa classe è rappresentata da strumenti come Rootkitrevealer di Windows SysInternals, Klister e Microsoft Strider Ghostbuster.

- **Specifica invariante**: questa classe di tecniche consiste nella rilevazione di malware di tipo stealth mediante il monitoraggio delle specifiche invarianti del kernel e rappresenta una modalità affidabile per la rilevazione di attacchi DKOM. In particolare per "specifica invariante del kernel" si intende un aspetto del kernel di cui non ci si aspetta la modifica sotto le normali condizioni del Sistema Operativo. Un esempio di invariante kernel è rappresentato dalla dimensione delle liste concatenate dei processi esecutivi e del kernel, che in normali condizioni è equivalente e viene violata in caso di occultamento di processi (DKOM). Una limitazione di questa classe risiede nella caratteristica propria delle specifiche di un sistema operativo di essere platform-dependent, rendendo complicata la formulazione di una lista completa delle specifiche di interesse.

- **Soluzioni hardware**: questa classe di tecniche consiste nella rilevazione di malware di tipo stealth mediante delle interfacce hardware che restituiscono un'immagine fedele della

memoria volatile di un sistema. Questa categoria rende futili le azioni di sovvertimento della memoria kernel attuate dai malware avanzati. Un esempio è rappresentato da Copilot [67], uno strumento di monitoraggio in grado di sfruttare l'accesso diretto alla memoria (DMA) attraverso il bus PCI per accedere alla memoria del kernel dall'hardware della macchina host e visualizzarla su una macchina diversa, garantendo così l'ottenimento della corretta memory image e procedere all'analisi. La limitazione delle soluzioni hardware risiede nell'incapacità di intervenire nell'esecuzione della macchina host, al fine di prevenire o rispondere agli attacchi. Il motivo di questa risiede nella limitazione degli accessi alla CPU, che impossibilita l'ispezione dei relativi registri e cache. Ancora il DMA, operando ad un livello più basso del kernel, non può essere usato per l'acquisizione fedele di strutture dati, e relativi lock, nella memoria kernel. Di conseguenza, le letture del DMA non sono sincronizzare con le scritture del kernel.

- **Tecniche di virtualizzazione**: questa classe di tecniche consiste nella rilevazione di malware di tipo stealth mediante l'utilizzo di ambiente virtualizzato e sfruttando l'apposito modulo di monitoraggio e gestione risorse, l'hypervisor. L'hypervisor, o Virtual Machine Monitor, possiede un livello di privilegio maggiore rispetto al Guest OS ed è responsabile del controllo degli accessi del Guest OS alle risorse hardware. Diversamente dalle soluzioni hardware, l'hypervisor è, per natura, sincronizzato con il Guest OS e può accedere alle informazioni relative alla CPU. Questo comporta la possibilità di utilizzare le tecniche di virtualizzazione non solo per la rilevazione di malware di tipo stealth, ma anche per la relativa prevenzione e mitigazione. In particolare, è possibile utilizzare l'hypervisor per forzare politiche sull'utilizzo delle risorse hardware, in modo da monitorare la macchina virtuale e metterne in pausa lo stato in risposta ad eventi sospetti. Un vantaggio delle tecniche di virtualizzazione risiede quindi nel prevenire l'esecuzione di codice malevolo nella memoria kernel, mentre la principale limitazione consiste nelle tecniche di anti-emulation apportare in risposta dai malware più avanzati.

Diversamente, gli approcci pattern-based puntano ad ottenere un più riconoscimento di malware eterogenei, andando a fornire risultati che privilegiano il "richiamo" alla "precisione". L'analisi dei modelli, descritta nella Figura 4.14 può essere applicata sia a frammenti di codice statico che a porzioni di memory image e sfrutta al contempo metodi di analisi statica, mediante ispezione di raw-code, e dinamica, in ambiente emulato.

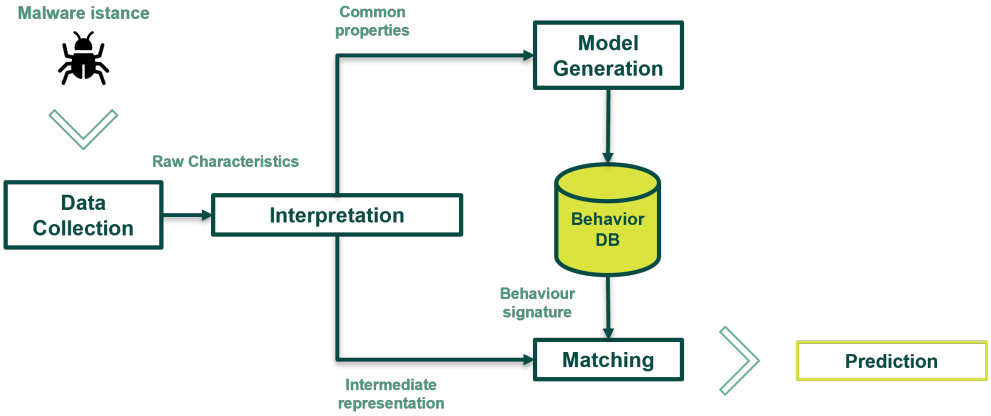


Figura 4.14. Flusso di rilevazione del malware

E. Rudd in [45] distingue tre categorie di analisi pattern-based:

- **Signature analysis**: questa categoria di analisi tenta la rilevazione di malware andando a comparare le firme di codici sospetti con le entry di una base di dati contenente valori etichettati come dannosi per il sistema. Una firma può essere ritornata direttamente come sequenze di codice binario in chiaro o mediante il calcolo del relativo hash. Nelle sezioni precedenti si è discusso delle tecniche attuate dai malware di tipo stealth al fine di produrre

un insieme di firme relative allo stesso binario malevolo. Le tecniche di analisi Signature-based tentano di risolvere questa soluzione di anti-detection andando a memorizzare anche porzioni di sequenze di byte, che si rilevano costanti nelle varie versioni del programma. In particolare, si è soliti classificare i rootkit residenti in memoria andando ad analizzare principalmente le impronte rilasciate sul disco, mentre si eseguono scansioni di memoria per rilevare impronte di offuscamento dei processi per mezzo di tecniche DKOM. Il vantaggio principale delle soluzioni Signature-based risiede nel rilevare con facilità i malware conosciuti, registrati nella base dati di interesse, trovando difficoltà nella rilevazione di malware o versioni di malware sconosciuti.

- **Behavioral/Heuristics analysis**: questa categoria di analisi tenta la rilevazione di malware andando a confrontare i modelli comportamentali di un programma sospetto con i corrispettivi identificati come dannosi per il sistema. J. Grégoire in [68] descrive il processo di analisi di questo genere distinguendo in quattro fasi principali: in primo luogo un rilevatore comportamentale identifica un file come sospetto monitorando le risorse richieste al sistema, disponibili nell'ambiente host; uno specifico componente si occupa dell'acquisizione, da localhost o mediante honeypot, dei dati relativi al programma in analisi, in modalità statica per ottenere le attività potenziali oppure in modalità dinamica per ottenere le azioni effettive; le informazioni collezionate verrano poi selezionate e formattate in una rappresentazione intermedia; l'ultimo passaggio prevede l'analisi delle informazioni selezionate tramite uno specifico algoritmo, in grado di comparare il modello estratto con i modelli conosciuti e collezionati in una determinata base dati e di etichettarlo come benigno o dannoso. La limitazione di queste tecniche risiede nella possibilità di analisi esclusivamente previa esecuzione del programma sospetto.

- **Machine learning model**: questa categoria di analisi tenta la rilevazione di malware andando ad applicare tecniche di intelligenza artificiale al fine di riconoscere un programma come dannoso mediante la classificazione di sequenze di codice. In letteratura è presente un'ampia quantità di pubblicazioni circa l'analisi malware per mezzo di questa classe, basate sulla crescente convinzione che gli strumenti di monitoraggio basati sull'intelligenza artificiale e l'apprendimento automatico saranno la soluzione ai moderni attacchi malware. In particolare, S. Saad in [69] descrive di come il modello perfetto di rilevamento delle minacce informatiche debba essere in grado di "rilevare tutti i tipi di software dannoso e non considerare un software benigno come software dannoso". Un metodo comune di machine learning adatto all'analisi di malware consiste nell'applicazione di tre fasi: cattura degli eventi generati dalla macchina host, etichettatura degli eventi in modo da scoprire i progressi dell'attaccante e apprendimento dagli eventi tramite appositi algoritmi [44]. E. Rudd in [45] distingue, inoltre, due tipologie di modelli di machine learning: "feature state model" e "state space model". I primi si basano sulla parametrizzazione dei comportamenti o delle firme per la classificazione di un programma sospetto come dannoso, mentre i secondi sfruttano la presenza di sequenza comuni di istruzioni interne ai codici sospetti per descrivere una specifica classe di malware. Una delle tecniche di apprendimento applicato all'analisi dei malware di tipo stealth che ha riscontrato maggiore accuratezza è stata proposta da S. Kilgallon in [70]. La tecnica colleziona informazioni sui valori dei registri e sulle chiamate API effettuate dai binari dei malware monitorati, per essere successivamente memorizzate in strutture dati e analizzate: il confronto tra malware sconosciuti e malware noti avviene tramite una metrica di somiglianza lineare ottenendo un'accuratezza del 98%. Lo svantaggio principale delle tecniche di rilevazione mediante machine learning risiede nell'impossilità di rilevare codice dannoso non visto in precedenza. Nello specifico, S. Saad in [70] analizza tre limitazioni delle tecniche di apprendimento applicato. Una prima difficoltà consiste nel costo della formazione e dell'aggiornamento dei rilevatori di malware nell'ambiente di produzione, a causa della dinamicità con cui i malware vengono modificati o sviluppati esibendo comportamenti diversi, che necessiterebbero di un apprendimento automatico addestrato frequentemente e in grado di adattarsi ai nuovi contesti per aumentare l'accuratezza di rilevazione. Ancora, le soluzioni di machine Learning dovrebbero essere ottimizzate in modo da attenuare e controllare l'effetto dei falsi positivi e dei falsi negativi, per fornire risultati interpretabili e comprensibili. Infine gli algoritmi di machine learning non sono progettati

per rispondere alle tecniche sviluppate dai malware per evitare la rilevazione e fuorviare le soluzioni di monitoraggio automatizzate.

Una delle principali priorità nella gestione di un evento di sicurezza informatica è rappresentata dal rilevamento e l'eliminazione del metodo di persistenza implementato dai malware di tipo stealth, che garantisce all'aggressore di rimanere sul sistema anche successivamente al riavvio, senza la necessità di infettarlo nuovamente. A. Perlman, infine, descrive in [52] delle regole da implementare per ottenere il sistema maggiormente resistente in termini di persistenza:

- **Ridurre i privilegi, cosicchè le tecniche di persistenza avanzate falliscano**: i privilegi di utente sul filesystem dovrebbero seguire il principio di sicurezza noto come "Least Privilege", che richiede la detenzione di un certo livello di privilegio per compiere specifiche operazioni sui file e directory. La finalità di questa misura sta nel prevenire che un utente possa inavvertitamente eseguire un'operazione non prevista. In [71] vengono delineati una serie di privilegi relativi al filesystem, utilizzati per modificare il normale comportamento del sistema Windows:
  - SeBackupPrivilege: consente di recuperare il contenuto del file
  - SeRestorePrivilege: consente la modifica del contenuto, del proprietario e della protezione relativa ad un file
  - SeChangeNotifyPrivilege: consente l'attraversamento di directory per accedere ai file o alle cartelle consentite
- **Limitare e monitorare i permessi del filesystem**: il sistema operativo Windows mette a disposizione una serie di controlli per la gestione dei privilegi all'interno del filesystem, descritti in [71]:
  - SePrivilegeCheck: questa routine controlla se uno specifico insieme di privilegi è abilitato nel token di accesso del soggetto.
  - SeSinglePrivilegeCheck: questa routine è una versione ottimizzata del SePrivilegeCheck, in grado di eseguire controlli per singoli prvilegi
  - SeAccessCheck: questa routine determina se i diritti di accesso richiesti possono essere concessi ad un oggetto protetto, normalmente un oggetto file per un filesystem
  - SeFreePrivileges: questa routine libera il blocco di privilegi restituito da una precedente chiamata a SeAccessCheck
- **Inserire modelli di controllo degli accessi (ACL) su chiavi di registro specifiche**: le liste per il controllo degli accessi (ACL) definiscono i permessi di accesso ad una specifica chiave di registro per un singolo account utente. Le ACL destinate ad un registro sono composte da un insieme di "access control entries" (ACE) che definiscono la singola regola. La procedura di inserimento o aggiornamento di una ACL viene riportata da A. Bertram in [72]. La definizione delle autorizzazioni per uno specifico registro avviene creando un oggetto System.Security.AccessControl.RegistryAccessRule, che permette di definire criteri come l'utente o il gruppo a cui si applica l'ACE e i diritti di accesso alla chiave di registro. I diritti di accesso possono essere selezionati in modo appropriato consultando una lista di 14 valori, disponibile in [73]. La regola creata viene successivamente aggiunta o sovrascritta all'ACL, che verrà definitivamente applicata alla chiave di registro solo successivamente ad una chiamata del comando Set-Acl impostando come parametro il relativo path.
- **Bloccare la scrittura di file in luoghi insoliti, come la cartella C:\Users\Public**: i permessi di scrittura di una specifica cartella possono essere modificati direttamente premendo il tasto destro del mouse e accedendo al menu Proprietà. Una modalità alternativa, specifica per la directory C:\Users\Public viene descritta da A. Bertram in [74], inviando i seguenti comandi su Powershell:
  - $ ACL = Get-ACL -Path "C:\Users\Public"

  - $ AccessRule = New-Object System.Security.AccessControl .FileSystemAccessRule("User","Write","Allow")
  - $ ACL.RemoveAccessRule($ AccessRule)
  - (Get-ACL -Path "C:\Users\Public").Access | Format-Table IdentityReference,FileSystemRights,AccessControlType, IsInherited,InheritanceFlags -AutoSize

- **Bloccare i file di configurazione in sola lettura**: i permessi di un generico file possono essere modificati direttamente premendo il tasto destro del mouse e accedendo al menu Proprietà. Una modalità alternativa, per impostare un file di configurazione in read-only è modellata secondo gli schemi descritti da A. Bertram in [74], inviando i seguenti comandi su Powershell:
  - $ ACL = Get-ACL -Path "ConfigFile"
  - $ AccessRuleRemove = New-Object System.Security.AccessControl .FileSystemAccessRule("User","FullControl","Allow")
  - $ AccessRuleRead = New-Object System.Security.AccessControl .FileSystemAccessRule("User","Read","Allow")
  - $ ACL.RemoveAccessRule($ AccessRuleRemove)
  - $ ACL.SetAccessRule($ AccessRuleRead)
  - (Get-ACL -Path "ConfigFile").Access | Format-Table IdentityReference,FileSystemRights,AccessControlType, IsInherited,InheritanceFlags -AutoSize

Capitolo 5

Framework esistenti di rilevazione di Stealthy Malware

Il tema del rilevamento e l'analisi dei malware di tipo stealth è stato affrontato nel corso degli anni da una vasta porzione di ricercatori e sviluppatori di soluzioni anti virus, nel tentativo di rimanere al passo con i progressi espressi dai produttori di codice malevolo. Questa sezione intende descrivere alcune delle più recenti metodologie di rilevamento e analisi degli stealthy malware, distinguendo i framework che sfruttano tecniche di Digital Forensics rispetto a quelli specifici nelle tecniche di Memory Forensics.

5.1 Framework di rilevazione di Stealthy Malware tramite Digital Forensics

Il campo della Digital Forensics propone un'ampia area di azione in termini di rilevamento e analisi dei malware di tipo stealth. Ogni branca interna alla forensica digitale presenta la propria modalità di investigazione in materia. In particolare, in questo elaborato, l'attenzione si è voluta focalizzare sulle ultime tecniche introdotte in letteratura, selezionate in base all'innovazione che hanno portato in materia.

A. Todd in [75] propone uno studio volto ad analizzare l'efficacia delle tecniche forensi di rilevazione online e offline dei rootikit. L'analsi è stata svolta su cinque diversi tipi di rootkit, comparando i risultati ottenuti da uno strumento di analisi live, cinque strumenti specializzati nella rilevazione di rootkit (RDT) e quattro strumenti di analisi offline. In particolare i rootkit, input dell'analisi, sono: AFXRootkit, Vanquish, Hacker Defender, FU e FUTo. Ogni malware è stato installato in macchine diverse con sistema operativo comune, Windows XP. AFXRootkit, Vanquish, Hacker Defender sono identificati come user-level rootkit, mentre Fu e FUTo come kernel-level rootkit.

- Live Response analysis: l'investigazione è stata eseguita su tutte le macchine in analisi con l'ausilio del tool Windows Forensic Toolchest, progettato nello specifico per la rilevazione di processi, servizi, driver, porte e file. WFT è stato in grado di rilevare diverse informazioni nascoste dai malware, come processi in esecuzione e driver di sistema. Lo strumento non è stato in grado di restituire informazioni circa file di sistema, porte aperte e servizi di sistema. In ausilio a WTF, sono stati utilizzati PSTAT, per informazioni generiche sui processi, PROCESS HANDLES, per dettagliate informazioni di sistema, NET START, per l'identificazione di servizi, e DRIVER, per l'identificazione dei driver di sistema. Riassumendo, WTF si è contraddistinto per un'alto detection rate nella rilevazione dei rootkit, non restituendo però alcuna informazione circa porte e file nascosti.

- RDT analysis: l'investigazione è stata eseguita su tutte le macchine in analisi con l'ausilio di quattro strumenti RDT: RootkitRevealer, RKDetector, BlackLight, IceSword. RootkitRevealer e RKDetector sono stati in grado di rilevare esclusivamente file nascosti, mancando nella rilevazione di tutti i rootkit; BlackLight e IceSword si sono rilevati efficienti nella rilevazione di tutti i rootkit. In particolare: RootkitRevealer è riuscito ad identificare file nascosti, configurazioni del Windows Registry e rootkit atti a nascondere eseguibili, mancando nella rilevazione dei rootkit atti a nascondere i processi e fornendo la minima quantità di informazione. RKDetector ha identificato le stesse informazioni del precedente strumento, presentandole in una modalità maggiormente comprensibile. BlackLight è stato in grado di identificare una lista di item nascosti, tra cui i file degli user-level rootkit, mancando nella rilevazione della cartella nascosta che li ospita. Per quanto concerne i rootkit kernel-level, BlackLight è riuscito a rilevare i relativi processi nascosti, senza però identificare il rootkit. Ad ogni modo BlackLight è riuscito ad identificare la presenza nei sistemi di tutti i rootkit in analisi. IceSword, uno degli strumenti più affidabili, si è distinto per aver restituito tutti i file, le directory, i servizi, le porte e i processi nascosti di tutti i rootkit user-level, mentre per quanto riguarda i corrispettivi kernel-level, sono stati rilevati tutti i relativi processi nascosti.

- Offline analysis: l'investigazione è stata eseguita su tutte le macchine in analisi con l'ausilio di due strumenti anti-virus signature based, Clam-AV e F-Prot, e due tool forensi, EnCase e Autopsy. alla ricerca di "cosa" è stato nascosto dai rootkit in analisi e "quando" è avvenuto il processo di occultamento. Clam-AV è riuscito a rilevare la presenza di 2 rootkit, Hacker Defender e Fu, mentre F-Prot ne ha identificati 3, AFXRootkit, Hacker Defender e FU. Successivamente, l'analisi si sposta verso gli strumenti forensi, ottenendo, mediante la ricerca di stringhe, qualche informazione temporali circa le DLL installate da AFXRootkit e Vanquish.

Gli esperimenti pubblicati da A. Todd in [75] rilevano come gli strumenti RDT siano in grado di fornire una quantità di informazioni affidabili maggiore rispetto ai corrispettivi live e offline.

W. Han in [76] introduce MalInsight, un framework per l'analisi di malware di tipo stealth mediante tre tipologie di profilazione mostrate in Figura 5.1: struttura di base, comportamento basso-livello e comportamento alto-livello. Mallnsight prevede la costruzione di un profilo completo per il malware estraendo le varie caratteristiche strutturali, le interazioni dannose di basso livello con il sistema operativo e le operazioni di alto livello su file, registri e sulla rete. Gli aspetti citati, coprendo interamente le caratteristiche del malware, rendono possibile risolvere efficacemente le limitazioni dell'individuazione di malware offuscati a causa della mancanza delle relative caratteristiche necessarie. L'esperimento si è svolto su di un data-set di 4250 sample, di cui 760 programmi benigni e 3490 malware di cinque classi differenti: backdoor, constructor, email-worm, net-worm e trojan-downloader. Il processo di rilevamento attuato da MalInsight consiste di 4 fasi:

1. Data Collection: questa fase, base dell'intero processo di analisi, prevede il collezionamento dei dati caratteristici dei programmi partendo dai report di analisi dinamica generati in un ambiente di esecuzione simulato. In particolare, i report sono generati mediante Cuckoo Sandbox [77].

2. Malware profiling: questa fase, partendo dai dati raccolti, consiste nella profilazione del malware sotto i relativi aspetti strutturali, basso livello e alto livello. La profilazione della struttura di base avviene selezionando sezioni-chiave rappresentative dal PE file, come il codice, i dati leggibili, le risorse e le sezioni della tabella di importazione ed esportazione. Il comportamento basso livello viene profilato basandosi sulle informazioni relative alle API e alle DLL, utili a descrivere le interazioni tra il programma e il sistema operativo. Il comportamento alto livello viene profilato selezionando le operazioni del programma su file, registri e rete, al fine di determinarne il livello di intrusione.

3. Features generation: questa fase, partendo dalle evidenze tornate dalla profilazione del malware, trasforma il modello di profilazione in un "feature space", che include i "feature vector" che collezionano i dati dei tre aspetti.

4. Malware detection: questa fase consiste nell'addestramento dei classificatori di machine-learning sulla base dei "feature-set" generati, al fine di predire nuovi sample di malware. Gli algoritmi di data mining implementano 4 classificatori al fine di classificare e rilevare il malware, tra cui Decision Tree, Random Forest, K-Nearest Neighbor e Extreme Gradient Boosting.

MalInsight è stato testato sull'effettiva rilevazione di malware conosciuti e malware non conosciuti, in modo da valutare quale metodo o combinazione di metodi di profilazione ritorni il valore di accuratezza più elevato. Per quanto concerne l'analisi dei malware conosciuti, il feature-space generato dalla profilazione sistematica ha raggiunto un'accuratezza del 99.76%, seguita dallo spazio generato mediante le caratteristiche strutturali, propriamente con valore di accuratezza pari al 98.82%. In questo caso la combinazione dei metodi strutturali e basso-livello non ha riscontrato vantaggi. Per quanto concerne l'analisi di malware sconosciuti gli esperimenti sono stati condotti al fine di rilevare i sample relativi all'APT Hangover. MalInsight è stato in grado di rilevare il malware sconosciuto con un'accuratezza del 97.21%. W. Han in [76] riconosce le limitazioni principali del suo progetto: nell'instabilità del modello di rilevazione, dovuta ad una assegnazione randomica del training set e test set nel processo di apprendimento automatico; nell'assenza di correlazione tra diverse caratteristiche dei malware; nell'assenza di predizione delle intenzioni relative agli attacchi condotti dai malware. Ad ogni modo i risultati riportati da MalInsight hanno mostrato come esso sia in grado di rilevare malware noti e non noti, ritornando un'accuratezza maggiore rispetto ai relativi progetti in letteratura [76].

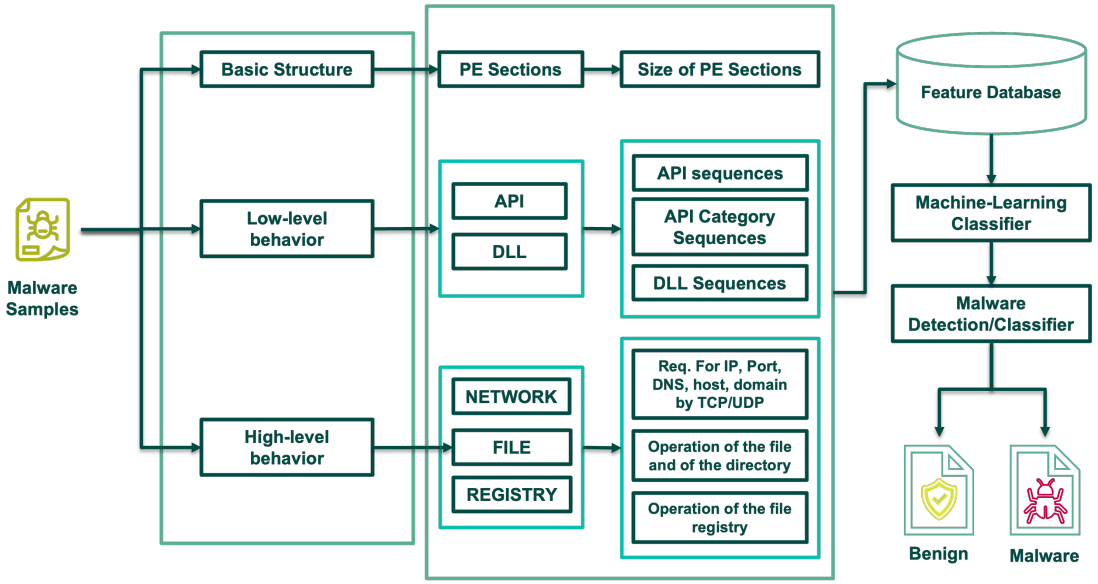


Figura 5.1. Design del processo di rilevazione di MalInsight

Q. Wang in [78] introduce il framework ProvDetector, basato sulla rilevazione di malware stealthy mediante un approccio provenance-based. ProvDetector impiega un modello di machine-learning basato sul modello delle Neural Network Embedding, utili a ridurre la dimensionalità delle variabili categoriali e rappresentare in modo significativo le categorie nello spazio trasformato [79]. La pipeline di embedding neurale viene applicata al fine di rilevare automaticamente qualsiasi comportamento che si discosti significativamente dai comportamenti normali dei programmi benigni. La particolarità di questo approccio consiste proprio nel training dell'algoritmo di machine-learning unicamente mediante dati benigni. L'intuizione alla base di questo approccio risiede nella comprensione che, sebbene i malware possano sfruttare tecniche di anti-emulation al fine di mostrare un comportamento benigno in ambiente virtualizzato e nascondersi durante le scansioni degli strumenti di monitoraggio, i comportamenti malevoli interagiscono inevitabilmente con il sistema operativo sottostante. L'ispezione sui malware stealthy avviene, quindi, analizzando il loro comportamento, in particolar modo i "provenance data" del sistema. Il termine "data

provenance", viene descritto da A. Gupta in [80] come: "una traccia di record che spiega l'origine di un dato (in un database, documento o repository) insieme a una spiegazione di come e perché è arrivato al luogo attuale". Il comportamento dei malware viene analizzato sulla base di percorsi causali nei grafi di provenienza, come caratteristiche per il rilevamento, ovvero tramite sequenze ordinate di eventi di sistema con dipendenza causale. ProvDetector comprende quattro fasi, raffigurate nella Figura 5.2:

1. Graph building: vengono collezionati in un database centralizzato i dati di provenienza del sistema, raccolti da un agente di monitoraggio installato sull'host. Il database viene esaminato periodicamente alla ricerca di eventuali dirottamenti su processi. Per ogni processo si costruisce il relativo grafo di provenienza.

2. Representation Extraction: vengono selezionati un sottoinsieme di percorsi del grafo di provenienza, sulla base di "percorsi causali".

3. Embedding: vengono convertiti i percorsi in vettori numerici, in grado di etichettare adeguatamente uno specifico percorso causale

4. Anomaly detection: viene utilizzato un metodo in grado di rilevare le "novità" nei percorsi anomali al fine di predire il programma dannoso. A questo scopo viene implementato il Local Outlier Factor (LOF) [81], un metodo di rilevamento delle "novità" basata sulla densità. Un punto è considerato un outlier se ha una densità locale inferiore a quella dei suoi vicini [80].

Per confermare l'efficacia dell'approccio, i ricercatori hanno condotto una valutazione sistematica di ProvDetector in un ambiente aziendale con 306 host per una durata di tre mesi, raccogliendo dati benigni di 23 programmi e utilizzandoli per la costruzione dei relativi modelli di rilevamento. ProvDetector ha così ottenuto prestazioni di rilevamento elevate, pero ad un valore medio di F1 score pari al 97.4%.

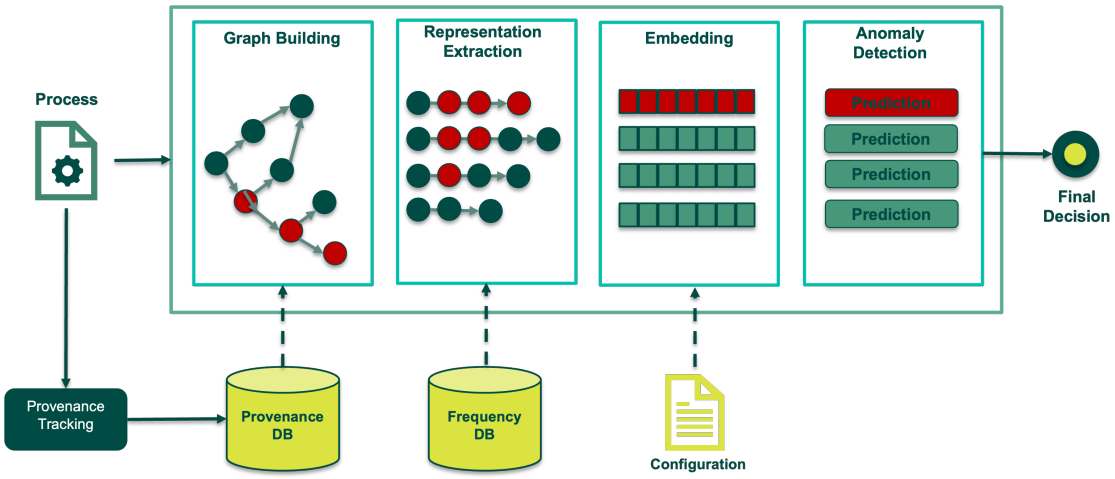


Figura 5.2. Design del processo di rilevazione di ProvDetector

M. Agarwal [82] introduce un metodo di monitoraggio a più livelli per la rilevazione di malware di tipo stealth capaci di nascondere la propria comunicazione di rete ad un sistema HIDS, comunemente usato per monitorare i diversi componenti dell'host, determinando la tecnica utilizzata dall'attaccante. Non sono state analizzate nei relativi esperimenti attività malevole del malware che non concernono le comunicazioni di rete, che se nascoste di per sé rappresentano evidenze di un'intrusione. Il metodo proposto, la cui architettura è mostrata in Figura **??**, combina un monitor di rete resiliente e "trusted" con più sensori "untrusted" e host-based. Per "untrusted" viene indicata la possibilità di essere attaccati in modalità diverse. L'architettura su cui viene implementato il framework consiste di un "host sensor" (HS) e un "trusted network monitor"

(TNM). L'HS, considerato untrusted, è installato sulla macchina host, mentre il TNM è distribuito su una macchina dedicata, con processore personalizzato per garantire l'affidabilità, ed è connesso alla rete mediante un'interfaccia di "Port mirroring", in modo tale da ricevere una copia di tutti i pacchetti inviati dall'host sulla rete e che attraversano lo switch. L'HS e il TNM hanno i seguenti componenti comuni o dedicati:

- Packet Hashing sensor: è parte sia dell'HS, rendendo possibile catturare il traffico relativo all'host su cui è distribuito, sia del TNM, garantendo la possibilità di catturare il traffico relativo a tutti gli host collegati, ed ha la funzione di network packet sniffer, impostato in modalità promiscua. In particolare esso è configurato per ignorare i pacchetti multicast e broadcast, che vengono evitati dai malware di tipo stealth per limitare le proprie tracce sulla rete, e accettare solo il traffico in uscita dall'HS, che potrebbe essere usato per contattare un server Command&Control (C&C).
- Netstat sensor: è parte unicamente dell'HS e memorizza le evidenze sulle connessioni di rete viste sull'host e ritornate dalla utility "netstat".
- Analysis Module: è parte unicamente del TNM e riceve le informazioni dai Packet Hashing Sensor e dal netstat sensor. Questo componente è responsabile dell'analisi sull'esistenza di stealthy malware nella macchina host.
- Heartbeat Generator e Heartbeat sensor: l'heartbeat generator è parte dell'HS, mentre l'heartbeat sensor è parte del TNM. La funzione di questi due moduli consiste nel mantenere attivi i sensori distribuiti nell'host, garantendo che non siano stati terminati o disabilitati da un particolare malware stealthy. A questo fine, l'heartbeat generator invia periodicamente messaggi di "keep-alive" all'heartbeat sensor.

In assenza di malware non dovrebbero mostrarsi discrepanze tra le comunicazioni di rete misurate al contempo nell'HS e nel TNM. Una differenza tra le due fornisce un'evidenza digitale di un'attività sospetta sull'host. Gli esperimenti sono stati svolti su cinque macchine, distinte da diversi sistemi operativi e diversi malware in esecuzione. In ogni macchina è stato installato un HS e la porta switch su cui viene connesso il TNM è stat configurata in "mirror mode". I risultati della valutazione hanno riportato la rilevazione di tutti i malware di tipo stealth, conosciuti e non, con il solo 0.01% di segnalazioni False Positive (FP). Il metodo introdotto non si è limitato alla rilevazione del traffico nascosto, ma in aggiunta è riuscito ad identificare la tecnica di occultamento usata dall'attaccante.

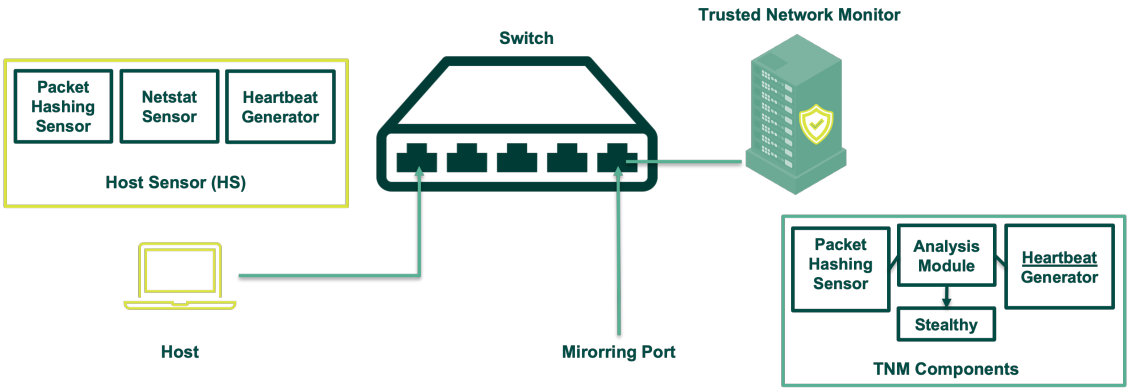


Figura 5.3. Architettura del metodo di multi-level monitoring proposto da M. Agarwal

P. Luckett in [83] propone una soluzione per il rilevamento di malware di tipo stealth sfruttando il consumo di energia della cpu a basso rendimento come base per gli algoritmi di machine learning. Il metodo è stato valutato su 4 sistemi operativi differenti (Windows 7, Windows 10, Ubuntu Desktop, and Ubuntu Server), impiegando 4 tipologie differenti di malware (KBeast, FU-To, Azazel, Windows NT rootkit). I dati sulla potenza sono stati raccolti e registrati utilizzando

un multimetro Fluke 289 con una pinza amperometrica Fluke i30 AC/DC. Il multimetro registra le variazioni delle letture di potenza in base a una determinata soglia. La soglia predefinita del multimetro è del 4% e non è stata modificata [83]. Il processo di "Data Collection" prevede la creazione dei data set, propriamente di 8 collezioni ciascuno: 2 collezioni eseguite sul sistema pulito, successivamente si infetta il sistema e sono eseguite altre due collezioni; il sistema viene poi passato ad un processo di "disk wiping" e, dopo aver ri-installato il sistema operativo, si ripetono le 4 collezioni precedentemente descritte. Ad ogni collezione vengono riempiti i seguenti valori, che verranno utilizzati per i vettori di machine learning: lettura iniziale sopra/sotto la soglia, durata della lettura, lettura minima nell'intervallo, lettura massima nell'intervallo, lettura media nell'intervallo. I data set differiscono per la modalità con cui vengono raccolti i dati:

- Data Collection 1: avviene seguendo azioni standard, come la creazione e cancellazione di file e cartelle. Essa è stata eseguita su Windows 10, Ubuntu Desktop e Ubuntu Server. Questa collezione ha prodotto 3 data set (type 1).
- Data Collection 2: avviene eseguendo stress test, forniti dalle risorse HeavyLoad su Windows 10 e Stress-ng su Ubuntu. Essa è stata eseguita su Windows 10 e Ubuntu Desktop. Questa collezione ha prodotto 2 data set (type 2)
- Data Collection 3: avviene al fine di fornire un out-of-sample test, eseguendo una sequenza di azioni mediante i prodotti della suite Microsoft (Notepad, Word, Calculator, Paint, Control Panel). Essa è stata eseguita su Windows 7.

Viene dimostrato come il consumo di potenza medio del sistema infettato, durante la collezione in stress test, sia maggiore rispetto al corrispettivo a sistema non-infettato. Lo stesso vale per la durata di lettura, più alta a sistema infetto e con massimi presenti in fase di boot e in fase di spegnimento. L'analisi è stata effettuata su un vario insieme di algoritmi di machine learning al fine di decretare quale ritornasse l'accuratezza maggiore: tre cui Nearest Neighbor, Decision Trees, Neural Networks, and Support Vector Machines. Ogni test, al fine di identificare l'algoritmo più performante, riporta i valori di accuratezza, AUC e tempo di addestramento (train time). Dalla valutazione dei risultati, originati dai valori collezionati su differenti sistemi operativi e su differenti hardware, si è compreso come un algoritmo basato su un'insieme di reti neurali (Nested Neural Network) sia stato il più accurato, seguito dagli algoritmi di tipo "Tree" (Complex Tree, Medium Tree, Boosted Tree, Bagged Tree). Il metodo proposto vanta la corretta ricognizione e classificazione di dati anomali e rootkit sconosciuti, con un'accuratezza platform-independent, dimostrando come la potenza della CPU possa essere una sfruttata per il rilevamento affidabile dei malware di tipo stealth, sia user-level che kernel-level. Questo framework è però limitato dai dati con cui sono configurati gli algoritmi e potrebbe non funzionare correttamente sui sistemi dinamici moderni. Una possibile applicazione potrebbe essere, al contempo, all'interno dei sistemi SCADA, caratterizzati da una quantità minima di applicazioni di terze parti e rari aggiornamenti del software.

5.2 Framework di rilevazione di Stealthy Malware tramite Memory Forensics

Il campo della Memory Forensics risulta più scarno in termini di rilevamento e analisi dei malware di tipo stealth, ma in rapida evoluzione. I framework di questa classe lavorano a stretto contatto con gli algoritmi di intelligenza artificiale, in modo da automatizzare un processo che manualmente può rilevarsi complesso. In particolare, in questo elaborato, l'attenzione si è voluta focalizzare sulle ultime tecniche introdotte in letteratura, selezionate in base all'innovazione che hanno portato in materia.

C. Spensky in [84] introduce LO-PHI, un framework per l'analisi di stealthy malware altamente sofisticati in grado di effettuare un'introspezione della macchina fisica sulla memoria volatile e sul disco, con la particolare introduzione di strumenti hardware-based. LO-PHI, acronimo di Low- Observable Physical Host Instrumentation, è un sistema in grado di analizzare il software in

esecuzione su macchine bare-metal Common-Off-The-Shelf (COTS), monitorando accuratamente host fisici in esecuzione in tempo reale. In particolare LO-PHI non richiede software aggiuntivo, ma esclusivamente una minima aggiunta hardware "plug-and-play". LO-PHI presenta un'architettura composta da "sensori fisici hardware e software per monitorare la memoria, la rete e l'attività del disco di un System Under Test (SUT), nonché per controllare la tastiera, il mouse e l'alimentazione" [84]. I dati collezionati vengono poi analizzati mediante strumenti di memory forensics e disk forensics, quali Volatility e SleuthKit, al fine di convertire i dati grezzi in output comprensibili. Quanto al modello di minaccia sulla cui analisi si basa il framework di "binary analysis" raffigurato in Figura 5.4, si suppone che il malware non abbia restrizioni e sia i grado di apportare qualunque modifica dannosa al sistema, lasciando tracce sulla memoria o sul disco. Si necessita, però, la presenza della strumentazione LO-PHI sul sistema precedentemente all'infezione del programma intrusivo. In particolare il framework comprende sensori, per collezionare la collezione dei dati (memoria, disco, rete), attuatori, per fornire input al sistema (tastiera, mouse) e software di analisi. Le funzionalità sono state implementare in ambienti sia fisici che virtuali. E' stato utilizzato un modulo Arduino al fine di emulare con precisione i movimenti di un mouse e cliccare i bottoni messi a disposizione dal programma sospetto. Quanto al reverting e il disaster recovery è stato utilizzato il programma Clonezilla, al fine di ripristinare il disco della macchina fisica allo stato precedente all'infezione. LO-PHI inoltre, mira alla riduzione degli artefatti prodotti sul disco e sulla memoria fisica, anche se sarebbero probabilmente inutilizzabili dal malware in mancanza di basi per il confronto. L'impatto del framework sulla memoria è stato calcolato, in condizioni di stress del sistema, con il tool RAMSpeed, mostrando una deviazione del memory throughput dello 0.4% rispetto ai valori raccolti in assenza degli strumenti caratteristici di LO-PHI. Gli artefatti sul disco sono stati calcolati con IOZone, riscontrando nelle operazioni di lettura una discrepanza nel throughput del 3.7%, mentre nelle operazioni di scrittura del 14.5%. L'infrastruttura software comprende:

- Master: responsabile per l'accettazione delle richieste e per la relativa delega ad un controllore
- Controllore: inizializzato con un insieme di macchine (pool), virtuali o fisiche, è responsabile di scaricare lo script per l'esecuzione dei malware e la collezione dei dati e sottoporlo allo scheduler
- Scheduler: attende che nel pool si renda disponibile una macchina del tipo appropriato, la alloca per l'analisi ed esegue le routine richieste
- Database MongoDB: su cui vengono memorizzati malware e risultati delle analisi

Lo script per l'esecuzione dei malware, in particolare, comprende le seguenti azioni:

- Reset della macchina ad uno stato pulito (Clonezilla o System snapshot)
- Produzione di un memory dump prima e dopo l'esecuzione del malware (clean e dirty state)
- Tentativo di cliccare eventuali bottoni grafici
- Produzione di screenshot
- Collezione delle attività sul disco e la memoria durante l'esecuzione del malware

Dai dati collezionati vengono ricostruite le modifiche apportate sulla memoria del sistema e sul disco, eseguendo Volatility e SleuthKit. In particolare, vengono usati i seguenti moduli di Volatility: psscan, ldrmodules, modscan, sockets, idt, gdt, ssdt, svcscan, e callbacks. Successivamente vengono filtrati i soli eventi attribuiti al binario in analisi. Il framework si è rilevato perfettamente in grado di rilevare la presenza e di analizzare malware conosciuti, sconosciuti ed "evasivi".

M. Botacin in [85] introduce MINI-ME, acronimo di Malware Identification based on Near- and In-Memory Evaluation, come un "hardware-based AV accelerator" in grado di supportare ogni sistema operativo ed interrompere l'esecuzione di un programma in presenza di pattern malevoli in memoria. Il rilevamento di malware di tipo stealth si basa principalmente sul matching

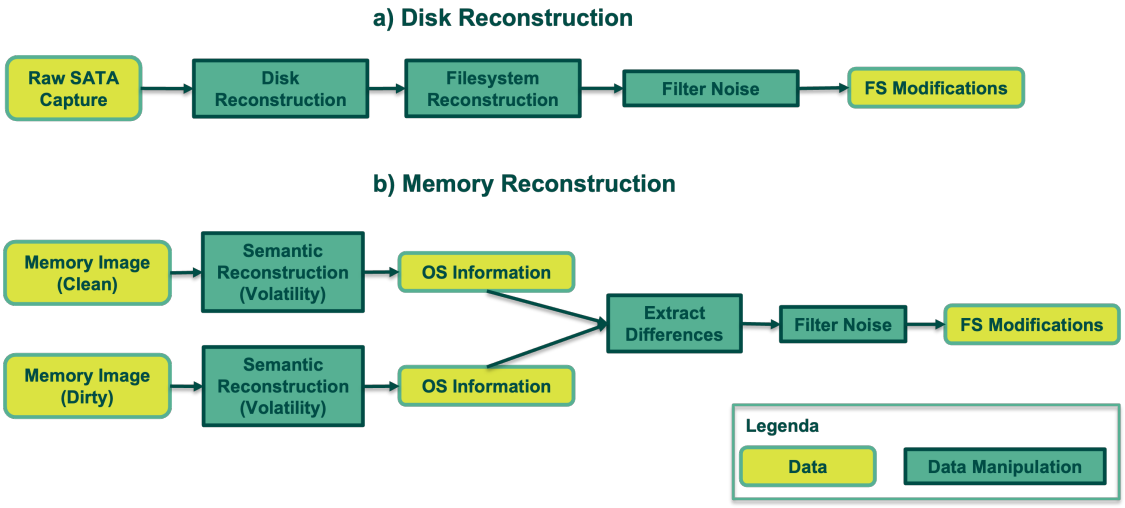


Figura 5.4. Binary analysis workflow implementato da Lo-Phi

delle firme del malware, fornite dalle aziende delle soluzioni antivirus e distribuite via Internet. Questo comporta l'utilizzo ideale di MINI-ME come strumento di contrasto a "1-day", in caso di minacce scoperte recentemente. La particolarità di MINI-ME risiede nell'accelerare il rilevamento di malware fileless in modo che l'operazione di pattern-matching dell'antivirus venga affidata all'hardware, aggiungendola al memory controller, il componente responsabile del "recupero automatico e continuo dei dati modificati e del loro confronto con un database di firme note di malware" [85]. I dati vengono gestiti in prossimità o all'interno della memoria volatile, evitando di passare direttamente per il processore, in modo da ridurre l'overhead. A questo scopo, vengono scansionate solo le regioni di memoria modificate. In presenza di pattern dannosi, il framework richiede il supporto di un componente antivirus software-based, per verificare la reale minaccia del relativo processo in esecuzione sul sistema, e in caso di falsi positivi (FP) si occupa dell'inserimento all'interno di un'apposita whitelist. L'architettura di MINI-ME, illustrata in Figura 5.5 si basa su tre moduli:

- userland AV: responsabile degli aggiornamenti delle firme sulle minacce del sistema e dell'applicazione di policy sulla sicurezza.
- Kernel driver: responsabile della comunicazione tra l'userland AV e l'hardware. In particolare il kernel driver accetta le richieste dallo spazio user e le inoltra al memory controller, scrivendo nei relativi registri di controllo.
- Memory-based AV (hardware): responsabile del rilevamento di pattern dannosi sulla memoria ed è composto da una Matching Engine, un Signature Database, un Malicious bit all'interno dei pacchetti o comandi di lettura, un Malicious bit Database, una Matching Signature Area (MSA), un Whitelisted Database, un Malicious bit per ogni entry della Page Table. In particolare la Matching Engine è implementata all'interno dei memory controller ed è composta da una o più basi di dati per le firme

MINI-ME funziona secondo un approccio Scan-On-Write (SoW), in modo che ad ogni ricezione di un pacchetto di scrittura dei dati al memory controller, il framework memorizza i dati nella DRAM e il componente Matching Engine confronta in parallelo i dati memorizzati nella sua base dati, aggiungendo un flag in caso di sospetto per la riga corrispondente. Il flag viene rimosso successivamente alla notifica da parte dello userland AV di aver ricevuto la segnalazione. In particolare, ad ogni traduzione di pagina anche il flag di sospetto viene mappato nella MMU e inoltrato dal sistema operativo come un semplice Page Fault. Questo avviene per colmare il gap semantico, che nasce dal rilevamento dei pattern sospetti nella memoria fisica e l'esclusiva operatività dello userland AV sulla memoria virtuale. Alla ricezione della notifica di Page Fault,

lo userland AV è responsabile per l'implementazione dell'apposita policy. La valutazione del framework è avvenuta misurandolo con 21 mila campioni di malware su un database di 100 mila firme e 3 meccanismi di matching (Direct Mapped Table, Signature Tree, Bloom Filter). MINIME è stato in grado di rilevare ogni malware a cui è stato sottoposto, senza produrre False Positive (FP) e con overhead trascurabile, mostrando la sua affidabilità in scenari reali.

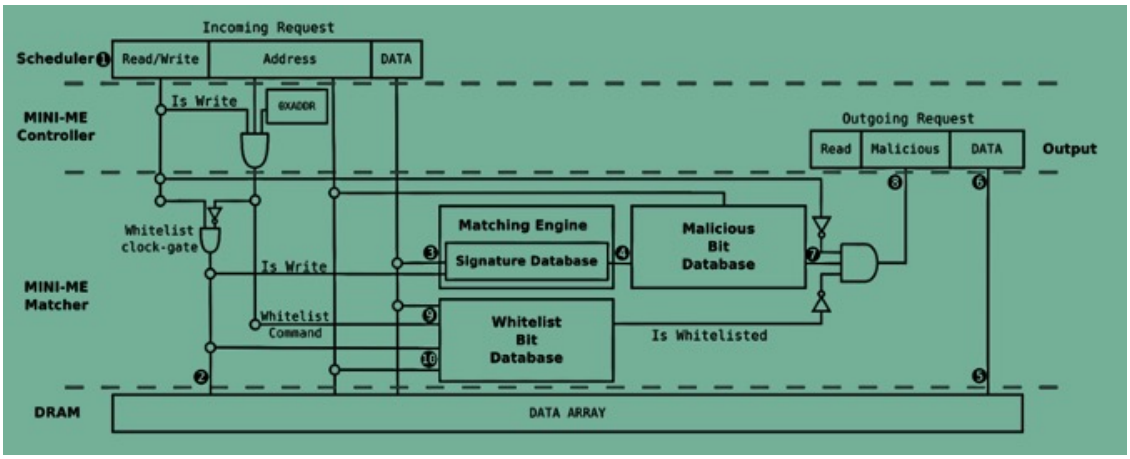


Figura 5.5. Architettura di MINIME

E. Mysliwietz propone in [86] una strategia di identificazione di malware di tipo stealth esclusivamente tramite strumenti di memory-forensics. Allo scopo di collezionare dati dalla memoria di una macchina fisica e osservare le modifiche sull'IRP in modalità stealth, sono stati creati due kernel driver. La lettura della memoria avviene sfruttando la classe Win32_PhysicalMemory fornita da Windows, per poi essere salvata sul disco. Il framework proposto analizza la memoria alla ricerca di evidenze digitali, che identificherebbero un rootkit stealth, impiegando diversi plugin del tool Volatility e indagando sulle strutture comunemente usate come target:

- Processi: la collezione dei processi è stata effettuata con l'ausilio del plugin "psxview", che compara la lista di processi tornati da altri moduli, come ad esempio "psscan" che identifica i processi in base al tag del pool di memoria (POOL_HEADER). Un'altra versione messa a paragone è fornita dal modulo "pslist", in grado di iterare sulla lista concatenata di EPROCESS, puntata da PsActiveProcessHead. Questo metodo serve a confrontare lo spazio di memoria di tutti i processi con lo stato conosciuto come buono.

- Thread: la collezione dei thread è stata collezionata con l'ausilio del plugin "threads", in grado di enumerare le istanze nella lista concatenata ETHREAD. Il plugin offre utili evidenze con l'ausilio dei tag ScannerOnly, AttachedProcess, OrphanThread, DKOMExit e HideFromDebug.

- DLL: la collezione delle dll incluse nella struttura LDR_DATA_TABLE_ENTRY sono estratte dal plugin dlllist. E' possibile evidenziare le dll relative ad un dato processo, estratto con psxview, specificandone il pid con l'apposito comando. "Malfind"e "ldrmodules" sono stati usati per cercare ulteriori dll nascoste. La ricerca degli hook è stata effettuata in due fasi. Principalmente è stato effettuato un dump delle dll tramite il modulo "dlldump" ed è stato confrontato con una copia pulita da infezioni. In caso di alterazioni delle dll, l'analisi è stata effettuata con "impscan" e "hooktracer".

- Funzionalità del Sistema: per la rilevazione degli attacchi DKOM è stato effettuato un dump della Global Descriptor Table, contenente le evidenze sui segmenti di memoria, tramite il plugin "gdt". Quanto all'analisi della SSDT è stato utilizzato l'omonimo plugin "ssdt". Per entrambe le strutture l'analisi si è effettuata sulla ricerca di hook. La collezione delle Callback è stata effettuata tramite il plugin "callbacks" e messa a paragone con valori conosciuti.

- Eccezioni ed Interrupt: la Interrupt Descriptor Table (IDT) è stata collezionata tramite il plugin"idt"
- IRP: la collezione è state effettuata tramite un processo di 4 fasi che comprende i plugin "driverlist", "devicetree" e "driverirp".

Questo framework è stato valutato dall'analisi di 62 sample di malware di tipo stealth di diverse famiglie, alla ricerca di una strategia comune sfruttata dai malware, portando alla luce le relative evidenze digitali.

R. Sihwailn presenta in [87] un framework in grado di rilevare e classificare malware di tipo stealth mediante tecniche di memory forensics, utili per l'estrazione delle evidenze dalla memoria volatile, e di machine learning, per la classificazione. Il framework è diviso in 5 passi principali:

1. Esecuzione del programma: viene utilizzata una Cuckoo sandbox al fine di eseguire in ambiente controllato i campioni dannosi e benigni. Al termine dell'esecuzione viene generato un memory dump.
2. Estrazione: viene utilizzato il tool di Volatility per l'estrazione di sei "memory-based feature" dall'immagine di memoria collezionata. In particolare, vengono estratte le chiamate API, le DLL, i Process Handle, i privilegi legati ai processi, le connessioni di rete e gli artefatti di code injection.
3. Conversione: le "feature" estratte dalla memoria sono convertite in vettori binari
4. Rimozione ridondanza: vengono utilizzate due tecniche di selezione, IG e correlazione, per rimuovere dal dataset le feature ridondanti o irrilevanti.
5. Classificazione: viene effettuata la classificazione dei malware mediante tecniche di machine learning

Gli esperimenti per la valutazione di questo framework hanno riguardato una collezione di 2502 malware, di differenti famiglie (Backdoor, Keylogger, Downloader, Adware, and Ransomware) e 966 file benigni, collezionati manualmente. Per la classificazione si è valutata l'accuratezza e il False Positive Rate (FPR) di cinque classificatori: Naïve Bayes, SVM, KNN, Decision Tree, Random Forest. I risultati degli esperimenti hanno dimostrato come l'algoritmo SVM tornasse la maggiore accuratezza pari a 98.5% e il minore FPR pari a 1.24% , seguito da Naïve Bayes (96.86%, 1.72%) e KNN (96.65%, 2.04%). Un calcolo del peso ha mostrato come la maggior parte delle feature appartengano alla classe delle DLL. L'esperimento è stato ripetuto analizzando ognuna delle cinque classi individualmente, confermando come la feature delle DLL riportasse l'accuratezza maggiore pari a 95.8%, comunque minore al calcolo dell'accuratezza basato sulla classificazione complessiva delle classi (98.5%).

M. Manna in [88] introduce un framework innovativo per l'analisi, tramite tecniche di memory forensic, dei malware di tipo stealth che sfruttano il .NET Framework e il .NET Core per la propria intrusione. Il framework si compone di una serie di Volatility plugin sviluppati ad hoc, al fine di testare gli esperimenti condotti e mettere a pubblica disposizione degli analisti forensi gli strumenti prodotti. In particolare si definiscono 5 informazioni utili all'analisi di applicazioni .NET, il cui riscontro è possibile con i plugin introdotti nel framework proposto:

- L'insieme delle classi caricate: la ricerca delle classi avviene successivamente alla comprensione della struttura dati del relativo Application Domain. Il plugin dotnet_memory_only permette di enumerare tutti gli Assembly caricati e determinare se siano memory-only, ottenendone l'indirizzo di base e passandolo all'API dump_pe di Volatility per la corretta memorizzazione sul disco.
- Per ogni classe, i suoi campi e metodi: la ricerca dei metodi associati ad una classe è possibile tramite diversi plugin, non citati dall'autore, in grado di interrogare il "metadata in-memory database" relativo ad un dato modulo, accedendo alla relativa struttura "PEAssembly"

- Per ogni campo, il nome e il tipo: queste informazioni sono tornate dal plugin dotnet_fields, in grado di identificare tipi di campi comunemente abusati dai malware.
- Per ogni metodo, la sua definizione e la sua posizione in memoria: ogni metodo è localizzato all'interno di una "MethodTable" relativa ad una specifica classe. Appositi plugin, non citati dall'autore, sono in grado di riportarne l'offset
- Istanze della classe (oggetti): vengono rilevate tramite il plugin dotnet_fields_value, in grado di elencare il valore di ogni campo relativo ad ogni istanza di una classe. Inoltre, le classi puntate per riferimento vengono identificare tramite il plugin dotnet_class_referenced, in grado di allertare in caso si presentino classi comunemente abusate dai malware

Il framework e i relativi plugin sono stati in grado di rilevare ed analizzare malware di tipo stealth, implementanti Covenant, il .NET framework per il Command&Control, e vengono proposti in ausilio ai moderni strumenti di analisi forense.

Capitolo 6

Framework di rilevazione ed analisi di Stealthy Malware

Questa sezione intende presentare una metodologia per il rilevamento e l'analisi dei malware di tipo stealth, mediante l'utilizzo di strumenti digitali disponibili con licenza open-source e, quindi, pubblicamente accessibili da chiunque abbia l'interesse ad approfondire l'ambito dell'analisi forense. Il framework pone prettamente l'attenzione sul rilevamento e l'analisi comportamentale di malware di tipo stealth, o file-less, per mezzo di applicativi appartenenti alla classe della Memory Forensics, volti quindi alla cattura di evidenze digitali presenti nella memoria volatile del dispositivo infetto sotto analisi forense. Si intende comunque notificare la possibilità di analizzare nel profondo, in aggiunta, malware che presentano caratteristiche completamente o parzialmente file-based.Il framework intende inoltre seguire con precisione le fasi del processo forense riportate dal National Institute Of Standards And Technology in [7], propriamente: Collection, Examination, Analysis, Reporting. A questo scopo, ogni tool presente nel framework è stato adeguatamente categorizzato nella corrispettiva area di analisi di interesse: il flusso è raffigurato nella Figura 6.1. L'analisi sui malware di tipo stealth, successivamente alla detonazione in ambiente controllato, pone le proprie basi sull'operazione di acquisizione della memoria RAM svolta sul sistema infetto. In seguito, secondo le politiche riportate dal NIST in [7], il memory dump viene passato sotto esamina in un differente sistema controllato, o sandbox, avente il medesimo Sistema Operativo e stesse configurazioni hardware, fisiche o virtuali. In questo si procede all'analisi della copia della memoria volatile catturata mediante appositi tool di Memory Forensics. All'occorrenza, è possibile analizzare le impronte digitali rilasciate sul sistema, in seguito all'analisi della memoria volatile, mediante tool di analisi statica. In presenza del file eseguibile, o una volta ritrovato durante le varie fasi del processo di analisi, è possibile procedere all'analisi comportamentale del malware mediante appositi applicativi di analisi dinamica. Durante la fase di detonazione del malware vengono, inoltre, attivati strumenti di analisi di rete, al fine di monitorare possibili canali di comunicazioni verso l'esterno instaurati dal malware, nel tentativo di inoltrare le informazioni ottenute sul sistema al proprio autore o ricevere eventuali comandi da server Comman And Control (C&C). Un'ulteriore peculiarità del framework in presentazione risiede nel supporto di strumenti di Open-Source Intelligence (OSINT), opportunamente selezionati nel relativo vasto panorama della "malware analysis". I tool di OSINT possono essere sfruttati in duplice maniera: a supporto delle varie fasi di analisi, mettendo a disposizione un'ampia gamma di informazioni relative al malware sotto esamina che possono essere utilizzate per un'analisi più profonda, o a conferma, nello step finale del framework, delle evidenze digitali riscontrate successivamente alla detonazione. Si intende quindi descrivere opportunamente ogni fase di analisi interna al framework in introduzione, seguita dall'esposizione dei relativi strumenti forensi. Per ogni strumento si intende dettagliare come è possibile installarlo sul sistema, quali sono le funzionalità ad esso connesse, in che modo interagisce con il sistema infetto e quali sono le azioni per riprodurre i risultati desiderati.

Il processo di **"memory dumping"**, corrispondente alla fase di "Collection" nel processo di Digital Forensics delineato in [6] [7], consiste nell'acquisizione di uno snapshot della memoria volatile del sistema infetto su cui si pone l'investigazione, successivamente alla detonazione del

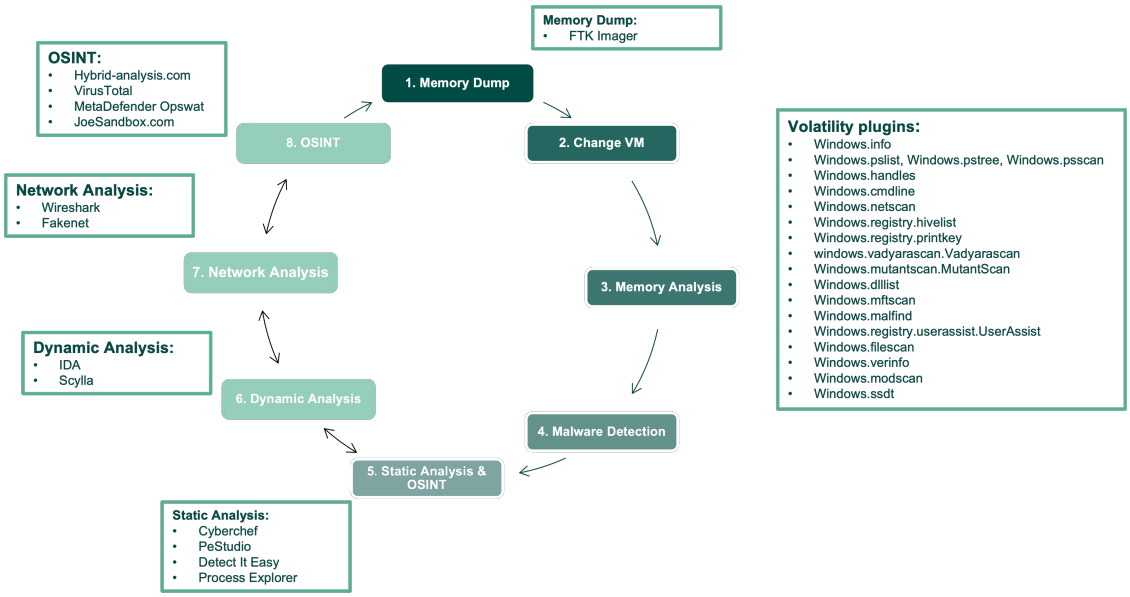


Figura 6.1. Flusso del framework di rilevamento e analisi

malware, al fine di condurre l'analisi forense post-mortem, sita in un differente sistema fisico o virtuale. **FTK Imager** è uno dei diversi tool presenti sul mercato che permette l'acquisizione della memoria RAM di un sistema in modo rapido e semplice, tornando un file con estensione ".mem". FTK Imager è gratuito e può essere scaricato ed installato gratuitamente accedendo al dominio AccessData.com [89]. Successivamente all'installazione sarà possibile catturare la memoria volatile del sistema seguendo i seguenti passi, descritti in [90] [91]:

1. Si esegue FTK Imager come amministratore mentre la macchina è in funzione.
2. Si seleziona "Capture Memory" accedendo al menu "File" in alto a sinistra.
3. Si imposta il path di destinazione e il nome del file con estensione ".mem" che verrà prodotto.
4. Si clicca su "Capture Memory" per dare inizio al processo di memory dumping, come raffigurato in 6.2.
5. Terminato il processo di memory dumping il file ".mem" sarà disponibile nella locazione indicata.
6. Si calcola l'hash del file ".mem" generato che, al fine di mantenere l'integrità del processo investigativo, dovrà rimanere il medesimo per l'intera indagine.
7. Sarebbe buona pratica, nel processo forense, generare due copie di backup dei dati acquisiti, tipicamente una "master copy", da conservare in luogo sicuro, e una "working copy" utile per l'analisi.

Durante il **passaggio al sistema di analisi** il file contenente la copia del memory dump ottenuta nel sistema sotto indagine viene spostato in una differente macchina fisica o virtuale avente le medesime caratteristiche hardware. Questa fase è necessaria al fine di prevenire l'alterazione della macchina sorgente durante i processi investigativi.

Successivamente si procede alla fase forense di **Memory Analysis**, corrispondente alla fase di "Examination" nel processo di Digital Forensics delineato in [6] [7]. In questa si prevede l'analisi di un'immagine di memoria volatile (RAM) al fine di filtrare l'insieme di dati raccolti e minimizzarne l'insieme allo stretto necessario rilevante per l'investigazione forense. Ogni funzione eseguita da un sistema operativo o da un'applicazione comporta modifiche nella memoria RAM, che di conseguenza conterrà i dati critici relativi alle ultime azioni svolte, dall'utente o da un

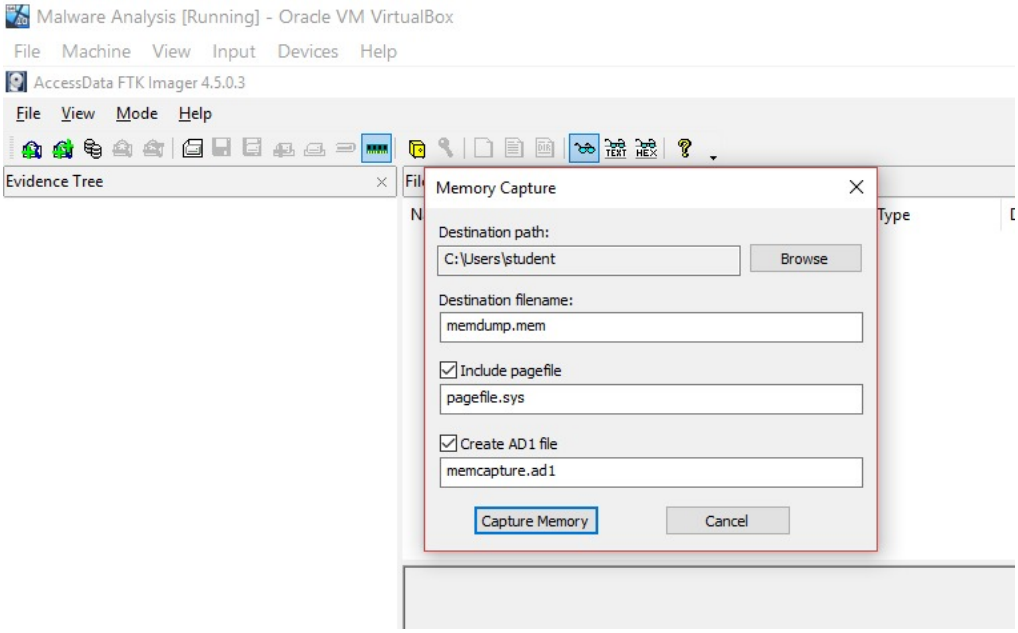


Figura 6.2. Processo di memory dumping

processo, sul sistema. Si comprende, dunque, la particolarità e l'importanza di questa fase nel metodo forense presentato. Nello scenario della Memory Forensics sono presenti diversi strumenti utili a ritornare artefatti digitali prendendo come input un'immagine di memoria. Il **Volatility Framework**, sicuramente il più completo nel panorama della Memory Forensics, si propone nel 2007 al Black Hat DC [92] come alternativa alle indagini basate unicamente su dump del disco rigido [93]. Volatility ha, quindi, introdotto la possibilità di analizzare lo stato di esecuzione di un sistema fisico o virtuale utilizzando i dati presenti in memoria RAM, fornendo una soluzione cross-platform, modulare ed estendibile sotto forma di "plugin". Uno degli obiettivi principali del progetto risiede nel proporre il Volatility Framework in modalità community-based, rendendo il codice sorgente, sviluppato in Python, disponibile pubblicamente in un apposita repository Github [94], allo scopo di incoraggiare ulteriori lavori di ricerca e condivisione delle conoscenze. Attualmente il progetto è sostenuto dalla Volatility Foundation, un'organizzazione indipendente, senza scopo di lucro, affiancata nell'innovazione da una delle comunità più ampie e attive del settore forense [95]. Il Volatility Framework si è sviluppato tanto da divenire uno strumento indispensabile per le indagini digitali, a cui si affidano le forze dell'ordine, i militari, il mondo accademico e gli investigatori commerciali di tutto il mondo. E' possibile scaricare ed installare le varie versioni prodotte in [96], messe a disposizione per un ampio insieme di sistemi operativi e architetture. In particolare, nel framework di analisi della memoria volatile in presentazione è stata scelta la versione "Volatility 3", scaricabile e installabile in [97]. Il Volatility Framework supporta i seguenti formati di input, essendo ulteriormente in grado di convertire un formato sorgente nel desiderato formato di destinazione:

- Raw linear sample (dd)
- Hibernation file
- Crash dump file
- VirtualBox ELF64 core dump
- VMware saved state and snapshot files
- EWF format (E01)
- LiME format

- Mach-O file format
- QEMU virtual machine dumps
- Firewire
- HPAK (FDPro)

Volatility presenta un approccio basato sui **Plugin**, ovvero delle API sviluppate in Python volte a rappresentare specifiche operazioni sulla memoria RAM e ritornare i relativi artefatti digitali. Il Volatility-Cheatsheet [98] distingue l'insieme di plugin in due classi, spesso riflesse nei loro nomi: i plugin "list" e i plugin "scan". I plugin "list" navigano nelle strutture Kernel per recuperare le informazioni e i footprint legati ai processi in memoria, comportandosi come farebbe un API di Windows ed ereditandone quindi la velocità e le vulnerabilità. Per esempio, se un malware usa DKOM per scollegare un processo dalla lista concatenata _EPROCESS, questo non apparirà nè nel Task Manager nè nella pslist. I plugin "scan", differentemente, adottano un approccio di ricerca nella memoria di elementi che potrebbero avere senso se deferenziati come strutture specifiche, comportando una maggiore lentezza rispetto ai precedenti, ma risultando in valori diversi Volatility, per una corretta esecuzione, necessita di sapere da quale tipo di sistema proviene il memory dump passato in input, in modo da selezionare le proprie strutture dati, algoritmi e simboli. Nel caso non si sia consapevoli del "profilo" di input, Volatility mette a disposizione plugin come "imageinfo" o "kdbgscan", disponibili unicamente su Windows, per avere dei suggerimenti e selezionare il sistema appropriato. Il comando comune per avviare l'esecuzione di un plugin su Volatility è il seguente, raffigurato ad esempio in Figura 6.3:

$ python vol.py [plugin] -f [image] –profile=[profile]

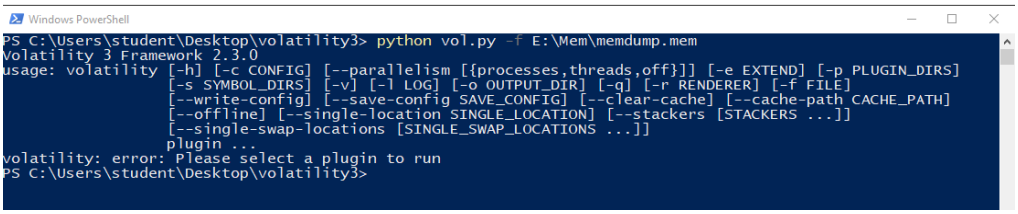


Figura 6.3. Comando di avvio di Volatility

Il framework in presentazione svolge analisi sull'immagine della memoria volatile di un sistema, passata come parametro in input, facendo uso dei seguenti plugin, perlopiù dettagliati in [99] [98]:

- **Windows.info**: ritorna i dettagli sul Sistema Operativo e sul kernel relativi al sample di memoria in analisi [99]. Tra questi è possibile riscontrare l'architettura del Sistema Operativo, l'indirizzo base del kernel in memoria, la quantità di processori nel sistema in analisi, il path della root di sistema, il timestamp di creazione della macchina fisica o virtuale. E' possibile eseguire il plugin windows.info tramite:

  $ python vol.py -f [image] windows.info

- **Windows.pslist**: ritorna un elenco dei processi presenti nell'immagine di memoria passata in input, accedendo alla stuttura PsActiveProcessHead [99]. Per ogni processo vengono ritornati, in particolare, i relativi PID, PPID, nome del processo (ImageFileName), indirizzo virtuale, quantità di threads e timestamp di creazione. Dall'analisi della lista di processi è possibile seguire il flusso di esecuzione del malware, mettendo sotto nota il relativo PID e la data di creazione, utili per correlazioni future. E' possibile eseguire il plugin windows.pslist tramite:

  $ python vol.py -f [image] windows.pslist

- **Windows.pstree**: ritorna i medesimi valori del plugin windows.pslist in un formato ad albero [99]. E' possibile eseguire il plugin windows.pstree tramite:

  $ python vol.py -f [image] windows.pstree

- **Windows.psscan**: ritorna un elenco dei processi presenti nell'immagine di memoria passata in input,scansionando i tag del pool (_POOL_HEADER). A differenza di windows.pslist, questo plugin rende possibile trovare processi precedentemente terminati o inattivi e processi che sono stati nascosti o scollegati da un rootkit [99]. E' possibile eseguire il plugin windows.psscan tramite:

  $ python vol.py -f [image] windows.psscan

- **Windows.handles**: ritorna un elenco dei gestori attivi relativi ad uno o più processi. Gli handle legati ad un processo possono essere ad esempio: file, chiavi di registro, mutex, pipe, eventi, thread [99]. E' possibile eseguire il plugin windows.handles tramite:

  $ python vol.py -f [image] windows.handles - -pid [PID]

- **Windows.cmdline**: ritorna i comandi eseguiti recentemente da un processo, ovvero la "User Command History", al fine di determinare la presenza di esecuzioni sospette. In particolare, i comandi passati a cmd.exe vengono processati da conhost.exe. Quindi in caso di compromissione del processo cmd.exe è possibile avere una 'recovering history' tramite un dump di conhost.exe [98]. E' possibile eseguire il plugin windows.cmdline tramite:

  $ python vol.py -f [image] windows.cmdline

- **Windows.netscan**: ritorna il dump delle connessioni di rete all'interno dell'immagine di memoria in analisi [99]. Netscan è in grado di trovare endpoint e listener TCP e UDP, visualizzare a video indirizzi IP e porte locali e remoti relativi ad una connessioni di rete, il timestamp in cui è stato collegato il socket o in cui è stata stabilita la connessione e il relativo stato attuale. Netscan si rileva particolarmente importante per rilevare i processi che stabiliscono connessioni in uscita sospette, con la possibile evidenza di backdoor e C&C [99]. E' possibile eseguire il plugin windows.netscan tramite:

  $ python vol.py -f [image] windows.netscan

- **Windows.registry.hivelist**: ritorna l'indirizzo virtuale dei registri hive in memoria il relativo percorso sul disco [99]. In particolare, un "hive" è un gruppo logico di chiavi, sottochiavi e valori nel registry che ha una serie di file di supporto caricati in memoria durante il boot. Tra questi si necessita valutare i file con estensione ".dat", in quanto vettori comuni per ottenere la caratteristica di persistenza, propria dei malware stealthy. E' possibile eseguire il plugin windows.registry.hivelist tramite:

  $ python vol.py -f [image] windows.registry.hivelist.Hivelist

- **Windows.registry.printkey**: ritorna le sottochiavi, i valori, i dati e tipi di dato contenuti all'interno di una specifica chiave di registro passata come parametro [99]. E' possibile eseguire il plugin windows.registry.printkey tramite:

  $ python vol.py -f [image] windows.registry.printkey -K [registry_key]

- **Windows.vadyarascan.Vadyarascan**: ritorna i risultati della scansione sulle "Yara rule", ovvero la presenza di una serie di stringhe, riconosciute come malevole, all'interno di un processo in esecuzione. Le Yara rule funzionano definendo una serie di variabili contenenti modelli di malware conosciuti [100] e nel framework in presentazione sono scaricate a parte

[101], per poi essere passate come parametro al plugin. E' possibile eseguire il plugin windows.vadyarascan tramite:

```
$ python vol.py -f [image] windows.vadyarascan —yara [yara_rule]
```

- **Windows.mutantscan.MutantScan**: ritorna i i risultati sulla scansione degli oggetti KMUTANT analizzando i "pool tag" [99]. Un "mutant" in Windows è l'oggetto kernel corrispettivo dei mutex in Linux, responsabile della sincronizzazione dei processi. I malware di tipo stealthy sono soliti utilizzare un oggetto mutant per assicurarsi di non reinfettare la stessa macchina e, quindi, limitarsi all'esecuzione di un'unica copia del malware. E' possibile eseguire il plugin windows.mutantscan.MutantScan tramite:

  ```
  $ python vol.py -f [image] windows.mutantscan.MutantScan
  ```

- **Windows.dlllist**: ritorna l'elenco di Dynamic Linked Library (DLL) caricate per un processo, accedendo alla struttura _LDR_DATA_TABLE_ENTRY. In particolare le DLL vengono aggiunte automaticamente a questo elenco quando un processo chiama LoadLibrary e non vengono rimosse finché non viene chiamato FreeLibrary e il conteggio dei riferimenti non raggiunge lo zero. E' possibile eseguire il plugin windows.dlllist tramite:

  ```
  $ python vol.py -f [image] windows.dlllist
  ```

  E' possibile visualizzare le DLL relative ad uno specifico processo concatenando la stringa "-p [PID]", passando il corrispettivo PID. Inoltre, la visualizzazione delle DLL di un processo nascosto è resa possibile specificando il corrispettivo offset fisico, ottenuto dall'esecuzione del plugin psscan, concatanando la stringa "offset=[OFFSET]".

- **Windows.mftscan**: ritorna il contenuto della Master File Table (MFT), presente nel NTFS filesystem e contenente una entry per ogni file presente su un volume del filesystem. Le voci della MFT, in particolare, comprendono tutte le informazioni relative ad uno specifico file, tra cui: la data e l'ora, le autorizzazioni e il contenuto dei dati [98]. E' possibile eseguire il plugin windows.mftscan tramite:

  ```
  $ python vol.py -f [image] windows.mftscan
  ```

- **Windows.malfind**: ritorna i risultati di una scansione nella memoria user alla ricerca di codice malevolo iniettato in un processo. In particolare, è in grado di localizzare istruzioni di codice e DLL malevole, rilevando le pagine contrassegnate dai permessi RWX [102]. Non è in grado di rilevare DLL iniettate tramite il metodo LoadLibrary, a differenza del plugin dlllist. E' possibile eseguire il plugin windows.malfind tramite:

  ```
  $ python vol.py -f [image] windows.malfind
  ```

- **Windows.registry.userassist.UserAssist**: ritorna l'elenco dei valori decrittati contenuti nella chiave di registro "UserAssist" [99]. In particolare, è possibile visualizzare una lista dei programmi recentemente eseguiti dall'utente su Windows e dei relativi metadati [98]. E' possibile eseguire il plugin windows.userassist.UserAssist:

  ```
  $ python vol.py -f [image] Windows.registry.userassist.UserAssist
  ```

- **windows.filescan**: ritorna l'elenco dei FILE_OBJECT nella memoria fisica usando la scansione dei tag del pool [99]. Per ogni file viene riportato il corrispettivo offset in memoria, nome e dimensione. Il plugin filescan trova i file aperti anche nella situazione in cui un rootkit nasconda i file sul disco e se il rootkit aggancia alcune funzioni API per nascondere gli handle aperti su un sistema attivo. E' possibile eseguire il plugin windows.filescan:

  ```
  $ python vol.py -f [image] windows.filescan
  ```

- **Windows.verinfo**: ritorna le "version information" incorporate nei file PE, quali file eseguibili e DLL. Questa caratteristica non è disponibile in tutti i file PE e viene falsificata da molti autori di malware in modo da includere dati falsi [99]. E' possibile eseguire il plugin windows.verinfo:

  $ python vol.py -f [image] windows.verinfo

- **Windows.modscan**: ritorna i risultati della scansione sui kernel driver, andando ad investigare sulle strutture LDR_DATA_TABLE_ENTRY e analizzando la memoria fisica per "pool tag" [99]. Modscan, a differenza del plugin "modules", è in grado di individuare driver nascosti dai rootkit o non collegati, riportando per ognuno il corrispettivo offset, name e path. E' possibile eseguire il plugin windows.modscan:

  $ python vol.py -f [image] windows.modscan

- **Windows.ssdt**: ritorna l'elenco delle SSDT native e GUI, visualizzando per ogni entry l'indice, il nome della funzione e il driver proprietario [99]. Nel caso di sistemi x86, il plugin esegue una scansione degli oggetti ETHREAD, in modo da raccogliere i puntatori unici ETHREAD.Tcb.ServiceTable, risultando più robusto del plugin thrdscan ed in grado di rilevare quando i rootkit creano copie degli SSDT esistenti e li assegnano a particolari thread. Nel caso di sistemi x64, viene disassemblato il codice in ntKeAddSystemServiceTable, al fine di ritornare i riferimenti alle strutture KeServiceDescriptorTable e KeServiceDescriptorTableShadow. E' possibile eseguire il plugin windows.ssdt:

  $ python vol.py -f [image] windows.ssdt

L'**analisi statica** definisce le metodologie per l'analisi del codice e della struttura di un malware per determinarne il funzionamento, consistendo nell'esamina di un file eseguibile senza la necessità di eseguirlo o visualizzare le istruzioni effettive [103], alla ricerca di indizi di intenti nocivi al sistema ospitante. Il primo obiettivo di questa analisi risiede nel fornire delle prime informazioni sulla funzionalità di un file eseguibile, in modo da riconoscere possibili indicatori di compromissione (IOC). Le proprietà statiche di interesse possono essere stringhe incorporate nel codice del malware, dettagli dell'intestazione, hash, metadati, specifiche risorse e così via. Di norma, i tool di analisi statica sono rapidi a fornire i relativi artefatti digitali, non essendo necessaria l'esecuzione del file eseguibile. Tra questi, strumenti come "Disassembler" e "Network Analyzer" possono essere utilizzati per osservare il malware senza porlo in un effettivo stato di "Run", al fine di raccogliere informazioni sul relativo funzionamento. La limitazione dell'analisi statica risiede nella possibilità di ottenere "falsi positivi", ovvero la rilevazione di indicatori di compromissione su un sistema che, effettivamente, non sono presenti. Per limitare o annullare la quantità di falsi positivi ritornata dall'analisi statica si necessita un'attività di esamina più profonda, mediante l'analisi dinamica del file eseguibile. Il framework in presentazione intende svolgere l'analisi statica del file infetto, mediante l'esecuzione dei seguenti appositi strumenti:

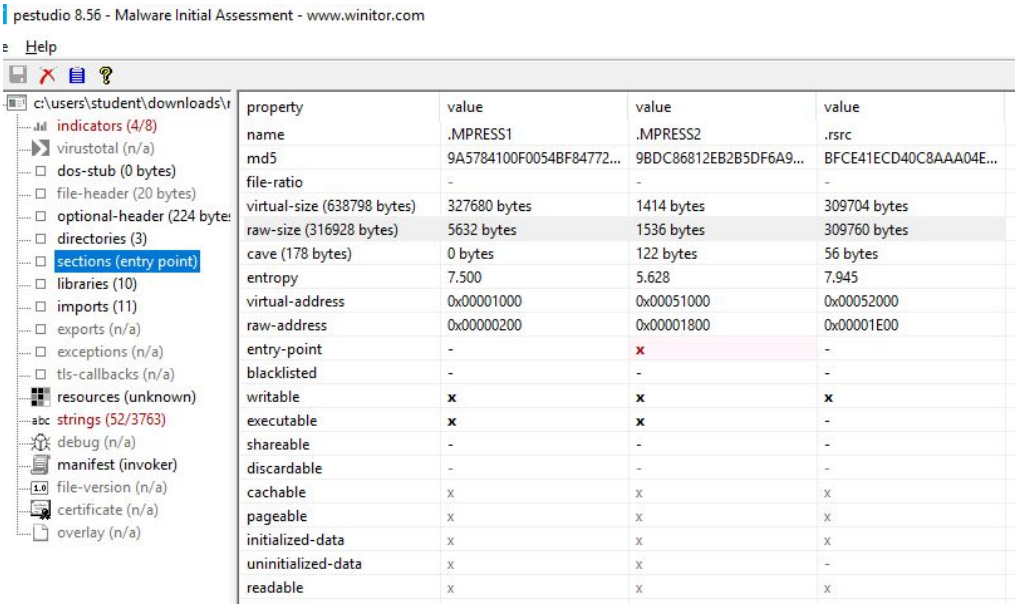
- **Cyberchef**: applicazione web gratuita, utilizzabile o scaricabile in [104], progettata per eseguire tecniche comuni di manipolazione dei dati in modo strutturato, sistematico e ripetibile. Queste operazioni comprendono semplici codifiche come XOR o Base64, crittografie più complesse come AES, DES e Blowfish, creazione di dump binari ed esadecimali, compressione e decompressione dei dati, calcolo di hash e checksum, parsing di IPv6 e X.509, modifica delle codifiche dei caratteri [105] Esso è utilizzato nel framework in presentazione principalmente allo scopo di decodifica del codice passato in input, in modo da ottenere informazioni utili, a partire dal nome del malware. Sono stati utilizzati le seguenti operazioni messe a disposizione da Cyberchef:

  – Magic tool: utile per la rilevazione di varie proprietà dei dati in ingresso. Suggerisce, inoltre, quali operazioni potrebbero aiutare a dare un senso ai dati. Tra queste si consiglia di focalizzarsi sulle informazioni che garantiscono un'entropia minore.

  - Regular expression: utile per filtrare i dati presenti nel codice. Nello specifico viene passata in input la stringa "[a-zA-Z0-9/+]30,", per filtrare i risultati dell'operazione From Base64
  - From Base64: utile per la decodifica dei dati da una stringa ASCII Base64 al suo formato grezzo. Il formato Base64 è una notazione per la codifica di dati arbitrari in byte utilizzando un insieme ristretto di simboli che possono essere comodamente utilizzati dagli esseri umani ed elaborati dai computer.
  - Remove null bytes: utile a rimuovere i "null bytes" dal codice in input
  - Generic code beautify: utile a rendere il codice maggiormente "human-readable"
  - : Find/Replace: Utile a sostituire tutte le occorrenze della prima stringa con la seconda. In particolare si intende sostituire il carattere ";" con il carattere di newline "\n", al fine di correggere l'indentazione di ogni riga di codice
  - Remove whitespaces: utile ad eliminare tutti gli spazi, i ritorni a capo, gli avanzamenti di riga e i tabulatori.

- **PeStudio**: tool gratuito, cross-platform ed open-source, scaricabile in [106] utilizzato per l'analisi statica di un file infetto, mediante un'ispezione basica del file header e delle "strings". Esso fornisce informazioni di diversa natura relative al sample passato in input. In particolare è possibile controllare la tipologia del file osservando i primi bytes in esadecimale del file header che rappresentano valori comuni di "byte pattern". La versione gratuita fornisce un set abbastanza completo di "feature" utili all'analisi. Le differenze con la versione "Pro" sono osservabili in [107]. Tra le diverse funzionalità PeStudio, successivamente al passaggio in input del sample, permette di:

  - visualizzare le varie sezioni che compongono il file eseguibile, accedendo all'area **Sections**, come raffigurato in Figura **??**



property	value	value	value
name	.MPRESS1	.MPRESS2	.rsrc
md5	9A5784100F0054BF84772...	9BDC86812EB2B5DF6A9...	BFCE41ECD40C8AAA04E...
file-ratio	-	-	-
virtual-size (638798 bytes)	327680 bytes	1414 bytes	309704 bytes
raw-size (316928 bytes)	5632 bytes	1536 bytes	309760 bytes
cave (178 bytes)	0 bytes	122 bytes	56 bytes
entropy	7.500	5.628	7.945
virtual-address	0x00001000	0x00051000	0x00052000
raw-address	0x00000200	0x00001800	0x00001E00
entry-point	-	x	-
blacklisted	-	-	-
writable	x	x	x
executable	x	x	-
shareable	-	-	-
discardable	-	-	-
cachable	x	x	x
pageable	x	x	x
initialized-data	x	x	x
uninitialized-data	x	x	-
readable	x	x	x

Figura 6.4. Visuale sulla sezione Sections di PeStudio

  - elencare le Windows API importate dal malware (IAT), mediante le quali è possibile comprendere il comportamento del malware successivamente alla compromissione del sistema, accedendo alla sezione **Imports**, descritta in Figura **??**
  - elencare tutte le stringhe in formato human-readable identificate all'interno del binario, associando ad ognuna la dimensione e se è un flag attestante la presenza in blacklist. Queste informazioni sono disponibili nella sezione **Strings**

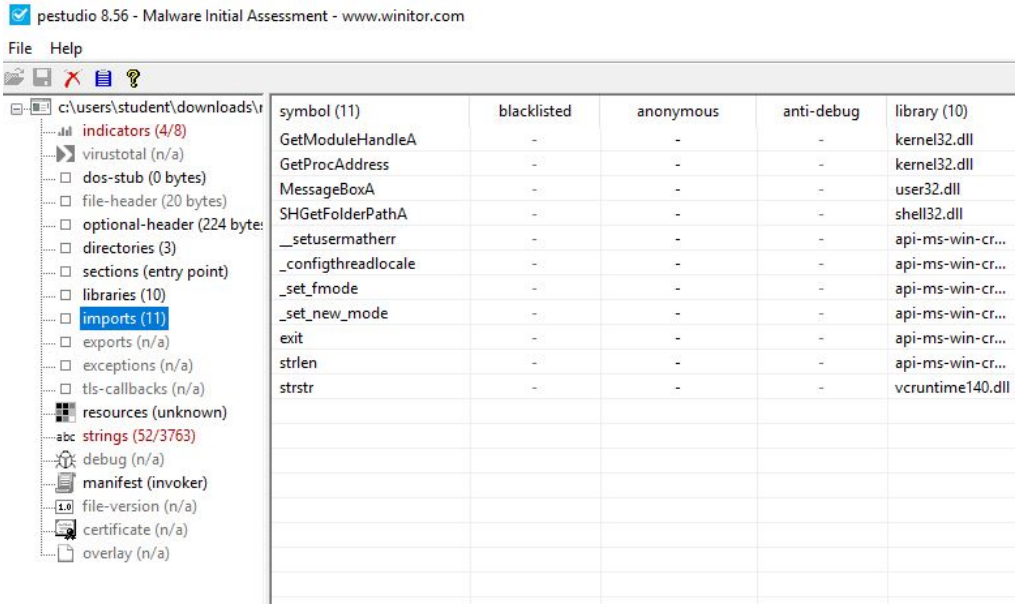


Figura 6.5. Visuale sulla sezione Imports di PeStudio

– identificare le risorse contenute nel sample, accedendo all'area **Resources**

- **Detect It Easy (DIE)**: tool gratuito, cross-platform ed open-source, scaricabile in [108], utilizzato per l'analisi statica di un file infetto, al fine di visualizzare la tipologia di Packer e il livello di entropia relativi al sample passato in ingresso [109], come raffigurato in Figura 6.6.

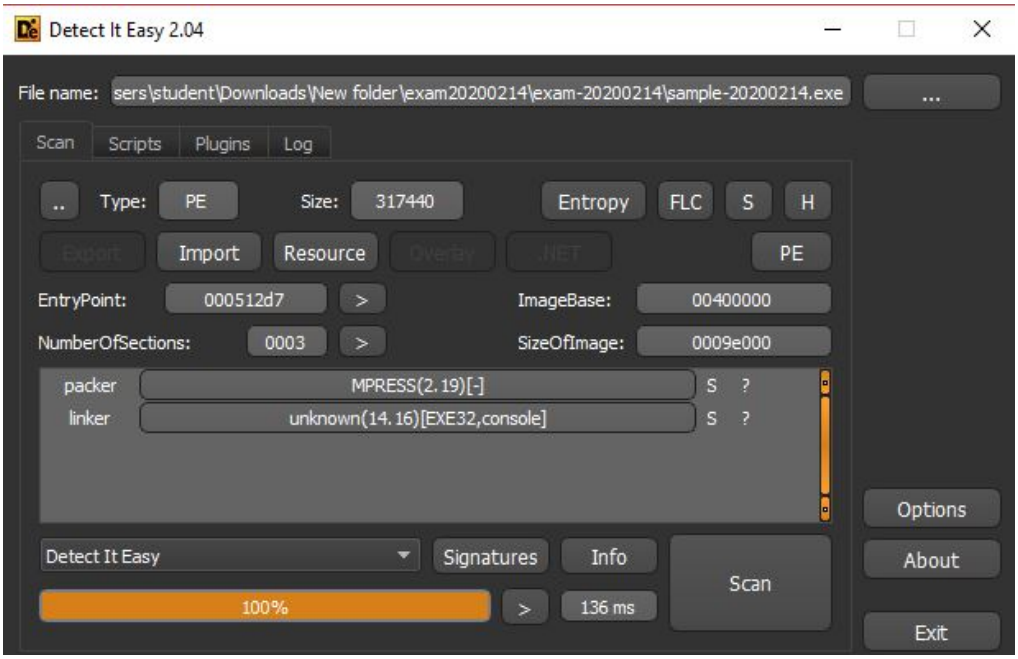


Figura 6.6. Risultati ottenibili su Detect It Easy

Un Packer, solitamente abbreviazione di "Runtime Packer", è un software che si scompone in memoria quando il file eseguibile compresso viene posto in stato di "Run" [110]. Questo tipo di compressione rende possibile diminuire la dimensione del file e privare gli utenti dalla decompressione dello stesso precedentemente all'esecuzione. Attualmente la velocità

della rete rende inusuale l'utilizzo dei Packer, che continuano ad essere largamente utilizzati per scopi malevoli, al fine di aumentare la difficoltà del "reversing" del codice sorgente e diminuire l'impronta del file malevolo sul sistema infetto. L'Entropia, invece, è un valore in un range da 0 a 8, utile per identificare se il malware è impacchettato o meno, al fine di offuscare il codice scritto dall'autore. Più alto è il valore dell'entropia, più è probabile che il malware sia stato impacchettato, con valori di 7-8 che confermano tale tesi. Un esempio è ritratto in Figura 6.7

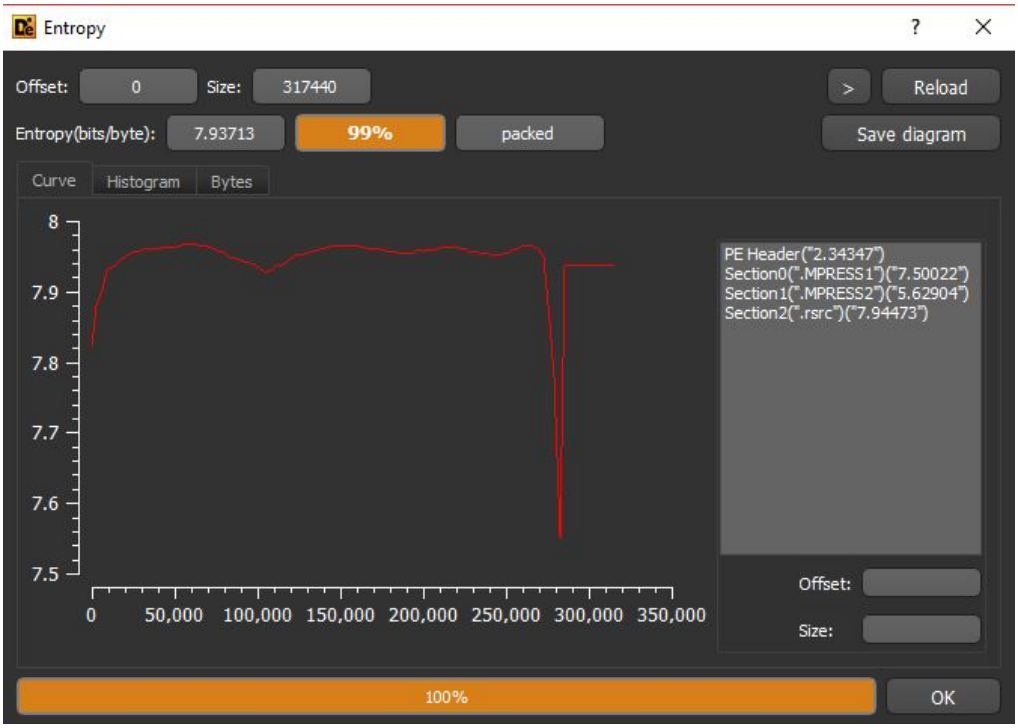


Figura 6.7. Visuale sulla curva di entropia di un file

- **Process Explorer**: programma gratuito per il Sistema Operativo Windows, creato da Microsoft SysInternals, per il controllo dei processi in esecuzione sul sistema. Process Explorer, scaricabile in [110] fornisce le funzionalità di Task Manager, offrendo un ampia gamma di funzionalità per la raccolta di informazioni relative ad i processi in esecuzione [111], come ad esempio la percentuale di risorse utilizzate in uno specifico periodo temporale. A differenza di Windows Task Manager, offre la possibilità di osservare quale thread, con lo Stack di chiamate, sta usando la CPU. Rende possibile visualizzare quale processo è responsabile dell'apertura di specifici file o directory, o quali handle e dll sono stati aperti o caricati [112]. Un esempio è rappresentato in Figura 6.8 Ogni processo attivo presente nell'elenco è associato, inoltre, al relativo account proprietario.

L'**analisi dinamica**, chiamata anche analisi del comportamento del malware, esegue il file infetto per esaminarne il comportamento. Come è noto, l'esecuzione di un file infetto potrebbe danneggiare il sistema. In relazione a ciò, si necessita la creazione e configurazione di un ambiente sicuro, detto "sandbox". In particolare, una sandbox so rivela come un ambiente virtuale, all'interno del quale è possibile effettuare l'analisi di un file infetto in modalità sicura. Quando il file infetto viene posto nello stato di "Run", è possibile osservare la presenza di indicatori tecnici di compromissione mediante l'analisi di firme di rilevamento di comportamenti sospetti. L'analisi dinamica di un file infetto è supportata da un'ampia gamma di strumenti, in modo da monitorare i comportamenti del malware, all'interno della sandbox, precedentemente e successivamente all'esecuzione. Le evidenze digitali comuuni, che possono essere tracciate da questa tipologia di analisi, possono essere: nuove chiavi di registro, indirizzi IP, nomi di dominio e posizioni dei percorsi dei file. Il passo successivo risiede nell'identificare se il file infetto sta comunicando con

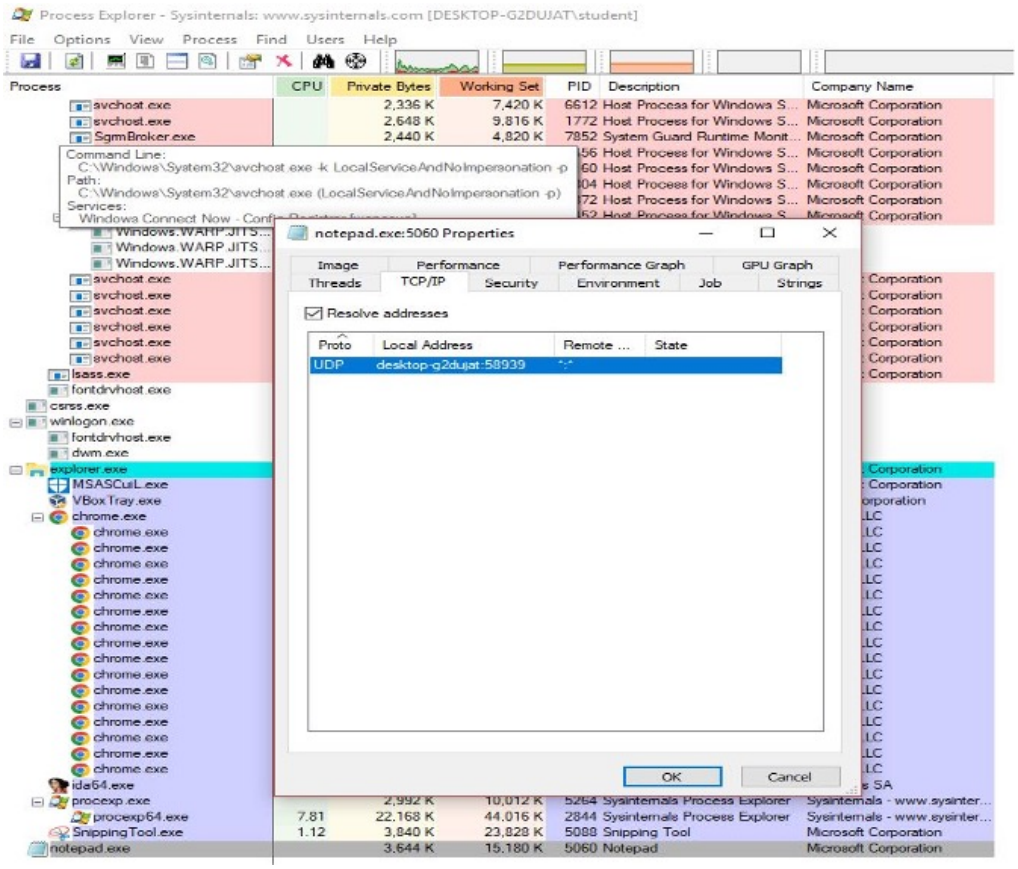


Figura 6.8. Focus sulle evidenze ritornate da Process Explorer

un server esterno (C&C) controllato da un attaccante, o meno. Inoltre, specifici strumenti, detti "Debugger", permettono l'esecuzione del file sospetto in stato di debug, al fine di studiare il comportamento effettivo del malware, compilando ogni istruzione in forma sequenziale ed eliminando possibili falsi positivi ritornati dall'esamina dello stesso in modalità statica. I criminali informatici cercano, ulteriormente, di occultare il file infetto e di renderlo inosservabile, apportando delle tecniche stealth, mascherandolo come un debugger. Ad esempio il programma potrebbe essere in grado di mettere in pausa l'esecuzione del suo payload malevolo, in attesa di specifici input dell'utente o alla ricerca di particolari processi in esecuzione nel sistema. Il framework in presentazione intende svolgere l'analisi dinamica del file infetto, mediante l'esecuzione dei seguenti appositi strumenti:

- **IDA**, abbreviazione di "Interactive Disassembler", è uno dei principali programmi usati per il reverse engineering di un file eseguibile, consentendo agli analisti di ottenere materiale prezioso per l'investigazione forense e comprendere, riscrivere o ricostruire l'architettura, la funzionalità e le strutture interne del programma [113]. IDA è fornito da Hex-Rays per disassemblare un file eseguibile, allo scopo di generare codice sorgente in linguaggio Assembly (ASM). IDA è uno strumento cross-platform, disponibile nella versione gratuita "IDA Freeware" in [114] o nella versione completa "IDA Pro" sotto licenza commerciale in [115], ed è sviluppato in modo da supportare un'ampia gamma di formati eseguibili per diversi processori e sistemi operativi. Esso, come descritto in [116], può essere ulteriormente sfruttato come debugger per file eseguibili Windows PE, Mac OS X Mach-O e Linux ELF. IDA esegue l'analisi automatica del codice in modalità dinamica, utilizzando riferimenti incrociati tra sezioni di codice, la conoscenza dei parametri delle chiamate API e altre informazioni [116]. Tutte le funzionalità di disassemblaggio e debugging offerte dall'applicativo IDA sono rese disponibili in forma interattiva mediante un'apposita GUI, al fine di semplificare all'utente l'analisi, rispetto agli strumenti eseguibili da linea di comando. In particolare, una volta

passato in input il file eseguibile, IDA dispone di due possibilità di visualizzazione, propriamente la "Text View" e "Graph View", entrambe commutabili mediante la barra spaziatrice. La Text View ritorna un elenco delle sezioni del file, all'interno delle quali è possibile seguire sequenzialmente le istruzioni Assembly associate ad ogni indirizzo. La Graph View ritorna, invece, una visuale sulla funzione corrente, rappresentata come un insieme di nodi collegati tra loro da archi [117]. Ogni nodo è rappresentato per mezzo di blocchi base, mentre gli archi simboleggiani i riferimenti incrociati, o "cross reference", del codice tra i nodi. IDA passa automaticamente alla modalità testo se l'elemento corrente non può essere visualizzato in modalità grafica. Nel framework in presentazione IDA viene utilizzato, nell'attività di reverse engineering del file eseguibile e allo scopo di ottenere il maggior numero di evidenze, seguendo seguente il flusso di pseudo-istruzioni:

1. Ottenere il "**tail jump**", o "tail call", ovvero una chiamata ad una subroutine eseguita al termine di una procedura, al fine di raggiungere l'OEP. Questi salti possono essere implementati senza l'aggiunta di un nuovo Stack frame allo Stack di chiamate, al fine di tornare direttamente alla funzione chiamante originale. Solitamente il tail jump si riconosce come il "Jump Back" più lungo all'interno della sequenza di istruzioni in analisi, come raffigurato in Figura 6.9

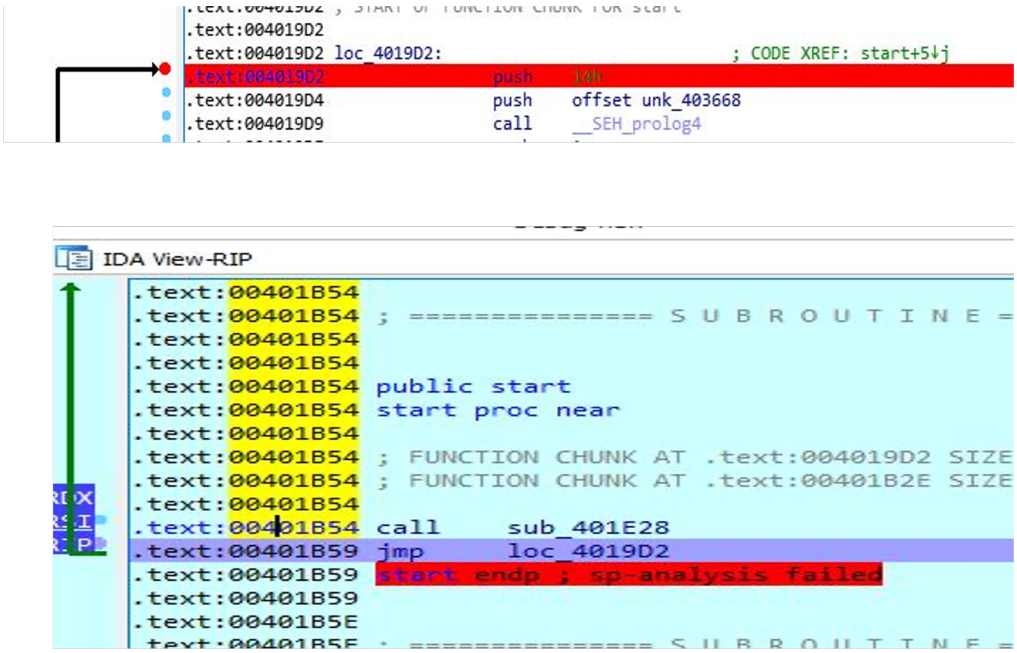


Figura 6.9. Visuale tail jump ottenibile su IDA

2. Una volta ottenuto l'indirizzo del tail jump, aggiungere un breakpoint sia all'indirizzo di partenza che all'indirizzo di destinazione del salto
3. Eseguire il debugger premendo il bottone di "Run", al fine di eseguire ogni istruzione dell'eseguibile e rilevare automaticamente l'indirizzo dell'**Original Entry Point (OEP)**. L'OEP è l'indirizzo della prima istruzione del codice malevolo prima di essere impacchettato, mediante il programma Packer, allo scopo di rendere maggiormente complicato il reverse engineering del malware [118]. L'OEP è identificato come il primo indirizzo tornato da IDA nel nuovo codice sorgente
4. Ottenuto l'indirizzo dell'OEP, esso deve essere verificato. A questo scopo viene utilizzato l'applicativo Scylla. In caso di verifica andata a buon fine, Scylla mette a disposizione la **versione unpacked** del file eseguibile, che rappresenta il risultato di reverse engineering del codice malevolo più vicino all'originale.
5. Passare la versione unpacked del file eseguibile, ottenuta da Scylla, su IDA

6. Analizzare in Graph View il flusso di esecuzione, la tabella delle funzioni e degli Imports nel codice sorgente del malware fino all'istruzione di uscita. In particolare si analizza il blocco relativo ad ogni funzione aggiungendo al suo interno un breakpoint corrispettivo all'indirizzo dell prima istruzione, saltando nel branch e premendo il bottone di "Run". Parallelamente si segue il flusso di istruzioni in Text View e si osservano le stringhe "readable" in ogni indirizzo ponendo l'osservazione in Hex-View.

- **Scylla**: tool open-source utile nell'analisi dinamica di un file eseguibile sospetto e scaricabile gratuitamente in [119]. Scylla riceve in input il sample del malware e l'indirizzo dell'Original Entry Point (OEP), recuperato mediante l'analisi effettuata sul disassembler IDA [120]. In particolare Scylla è utile per l'unpacking del codice in memoria, tramite il bottone Fix Dump, e per la ricostruzione della Import Address Table (IAT), mostrata nella sezione **Imports**. Premendo sul bottone **dump** è possibile, inoltre, salvare il dump della versione, come rappresentato in 6.10 "**unpacked**", così da poterlo analizzare su IDA senza le restrizioni apportate dal software packer.

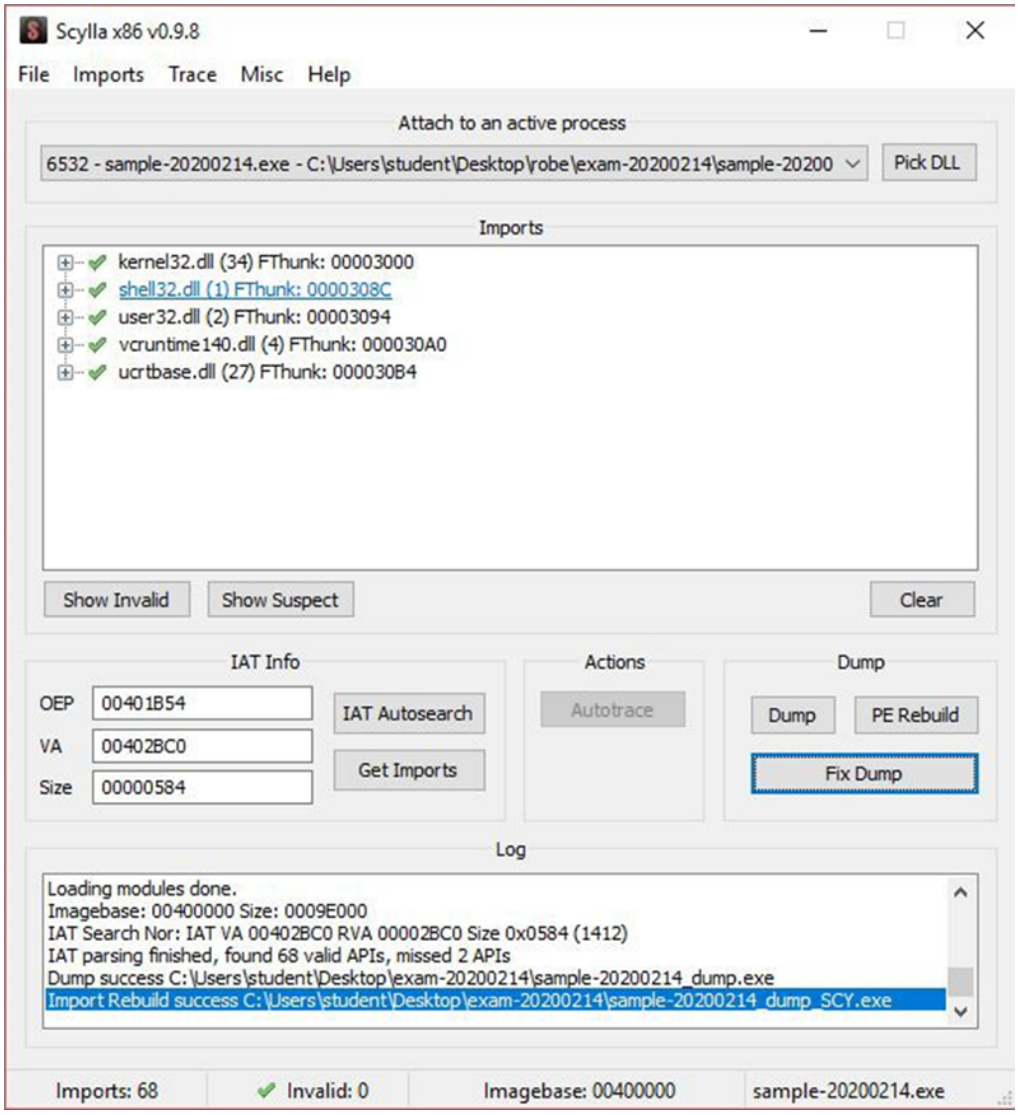


Figura 6.10. Visuale sui risultati ottenibili da Scylla

L'**analisi del traffico di rete** proveniente da switcher e router consente di identificare comportamenti anomali di diverse tipologie [121]. La Network Analysis si focalizza, quindi, nella rilevazione di tentativi, iniziati da file sospetti, di stabilire una comunicazione con il proprio

autore, fornendo un percorso per il controllo della macchina infetta. Questi processi sfruttano server intermediari, detti "Command & Control" (C&C), per il passaggio di informazioni digitali sottratte alla macchina infetta, lato malware, o impartire comandi da eseguire su di essa, lato attaccante. Tale connessione cercherà in genere di simulare il normale traffico di rete attraverso l'uso di HTTP, HTTPS o DNS. Una situazione comune è rappresentata dal malware in che si pone in ascolto di comandi, senza ricevere nulla, producendo in tal modo una quantità identica di dati trasmessi al medesimo indirizzo di destinazione. Il framework in presentazione intende svolgere l'analisi dinamica del file infetto, mediante l'esecuzione dei seguenti appositi strumenti:

- **Wireshark**: Wireshark è il più diffuso analizzatore di protocolli di rete, o "packet sniffer", e fornisce un ricco insieme di funzionalità per esaminare a livello microscopico le connessioni di rete [122]. Wireshark è sviluppato per essere disponibile su diverse piattaforme, tra cui Windows, Linux e macOS ed è scaricabile e installabile in [123]. Lo strumento svolge, nel particolare, tre funzioni [124]:
  - Cattura di pacchetti: permette di ascoltare una connessione di rete in tempo reale e catturare ogni pacchetto che viene trasmesso nel flusso di dati.
  - Filtraggio: permette di filtrare l'enorme insieme di pacchetti secondo diversi filtri, tra cui indirizzi IP sorgente e destinazione, porte sorgente e destinazione o protocolli di rete. Applicando un filtro, è possibile ottenere solo le informazioni necessarie all'analisi.
  - Visualizzazione: permette di osservare nel dettaglio ogni pacchetto di rete e visualizzare intere comunizazioni e flussi di rete, come ritratto in Figura 6.11

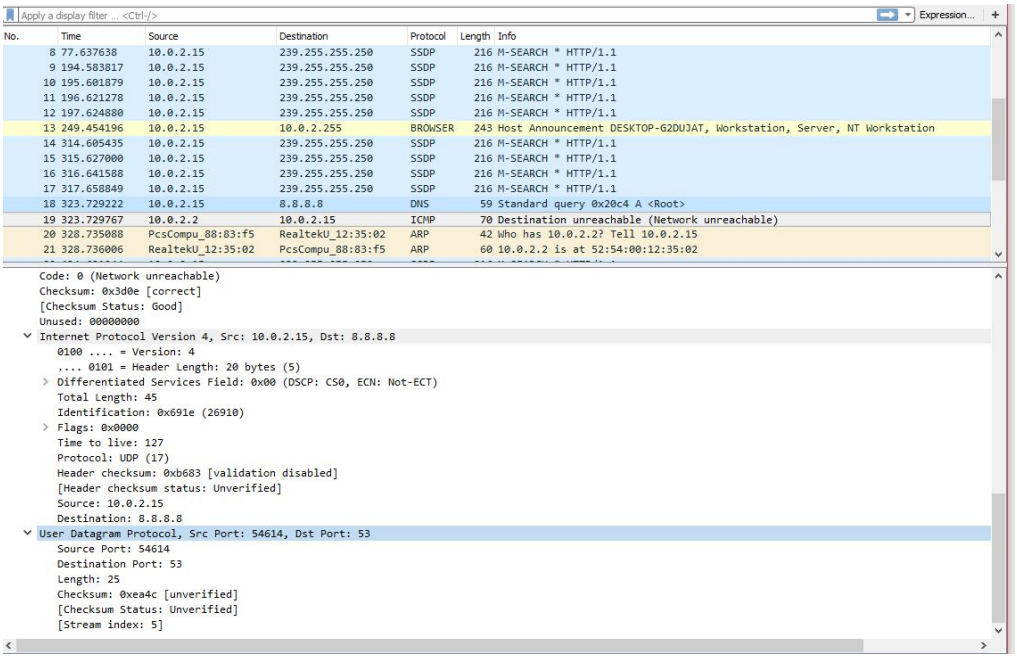


Figura 6.11. Visualizzazione dettaglio dei pacchetti di rete su Wireshark

Il framework in presentazione sfrutta le funzionalità offerte da Wireshark al fine di analizzare le comunicazioni di rete aventi come indirizzo IP sorgente o destinazione il corrispettivo ritrovato all'interno del codice sorgente o durante le varie fasi di analisi. L'analisi di rete ha lo scopo di confermare o eliminare il sospetto che la macchina sotto analisi forense si trovi in comunicazione con un server Command & Control (C&C), al fine di inviare informazioni o ricevere comandi dall'esterno.

- **Fakenet**: strumento di simulazione di rete, sviluppato in Python, utile per l'analisi dinamica dei malware e scaricabile gratuitamente seguendo le istruzioni disponibili in [125]. Fakenet è in grado di reindirizzare tutto il traffico in uscita da una specifica macchina verso

il localhost, incluso il traffico IP codificato e il traffico DNS. Inoltre, lo strumento è in grado di ascoltare il traffico su qualsiasi porta e d implementare diversi protocolli per garantire che il codice dannoso continui a essere eseguito e possa essere osservato da un analista. Fakenet viene sfruttato, inoltre, per l'analisi e il controllo dei beacon generati dal sample in fase di esecuzione e, nel particolare, per l'osservazione di quali informazioni vengono trasmesse in rete allo scopo di Command & Control (C&C). Fakenet può essere eseguito su Windows direttamente dal Command Prompt, con il comando "fakenet.exe", visualizzabile ad esempio in 6.12. Nel framework in presentazione è stata applicata la configurazione di default, buona abbastanza per condurre l'attività, ma è possibile in ogni caso personalizzare la configurazione in base al malware in analisi [125]. In particolare il file di configurazione, disponibile di default nel percorso "configs\default.ini", può essere modificato con il comando "fakenet -c FILE".

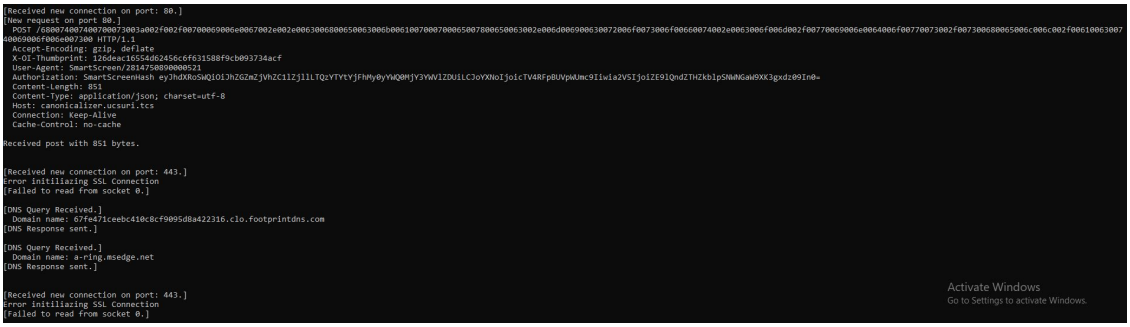


Figura 6.12. Output restituito da Fakenet

Si procede quindi all'applicazione degli strumenti di **Open-Source Intelligence (OSINT)**, un'attività di indagine volta alla raccolta di informazioni reperibili da fonti pubblicamente accessibili ed è parte di una serie di processi di natura ''Intelligence". Le pratiche di OSINT, in forma generica, si differenziano a seconda del target fisico o virtuale che si vuole investigare. Il primo passo in questa direzione risiede nella selezione degli strumenti necessari alla collezione dei dati, derivante dallo studio di un'ampia gamma di tool automatici, piattaforme social e siti web dedicati. L'analisi dei risultati avviene successivamente alla correlazione e rielaborazione delle evidenze riscontrate e verificate da diverse fonti. Nel dettaglio dell'analisi del malware le attività di OSINT forniscono un vasto insieme di informazioni, sotto forma di report, relative ad uno specifico malware, passando solitamente in input il corrispettivo hash o un url malevolo. A questo scopo, l'OSINT si colloca nelle fasi principali della ricerca, in modo da semplificare di gran lunga i processi di analisi, poichè i risultati ottenuti sono utili ad orientare le fasi successive dell'investigazione forense o a confermare le evidenze riscontrate. Nel framework in presentazione le procedure di OSINT vengono, difatto, sfruttate negli ultimi step dell'analisi in modo da confermare eventuali sospetti su evidenze riscontrate nelle fasi precedenti o, nel caso, per tornare sui propri passi ed operare in forma più profonda. Lo svantaggio di queste soluzioni risiede nella difficoltà o impossibilità di ritornare evidenze utili all'indagine forense nel caso di malware non noti e, quindi, non analizzati in precedenza. Sono ritenuti utili all'analisi di malware, stealthy e non, i seguenti strumenti di OSINT:

- **hybrid-analysis**: servizio open-source di analisi malware automatizzata, alimentato da CrowdStrike Falcon Sanbox, utile ad approfondire le proprietà e il livello di minaccia di un file sospetto o malevolo passato in ingresso. Hybrid-Analysis è disponibile in [126] segue un approccio analitico basato sulla combinazione di dati a Run-time, ovvero ottenuti dall'esecuzione in ambiente controllato, con l'analisi dei dump di memoria, al fine di estrarre tutti i possibili percorsi di esecuzione [127]. Poichè Hybrid-Analysis è una sandbox, i dati che si otterranno saranno strettamente correlati a potenziali attività dannose e sospette [128]. E' possibile passare come input file compressi contenenti file sospetti, url malevole o passare alla ricerca di indicatori di compromissione (IOC) a partire da stringhe sospette ottenute durante i vari step di analisi. Hybrid-Analysis, a differenza di diversi tool della stessa categoria, fornisce l'ulteriore possibilità di passare il file in input ed eseguire una YARA search,

in grado di comparare firme ottenute all'interno del file con firme specifiche di malware noti o famiglie di malware. Lo strumento fornisce in output una serie di informazioni relative al file passato in ingresso, in particolare:

  - **Analysis Overview**: ritorna informazioni generali circa il file in ingresso, come sha256, dimensione, famiglia di malware, livello di minaccia, sistema operativo target.
  - **Anti-Virus Scanner Results**: ritorna informazioni sulla percentuale di minaccia ritornata dalle scansioni antivirus su CrowdStrike Falcon, MetaDefender e VirusTotal. Al relativo punteggio possono essere associati dettagli sulla scansione, come rappresentato in Figura 6.13.

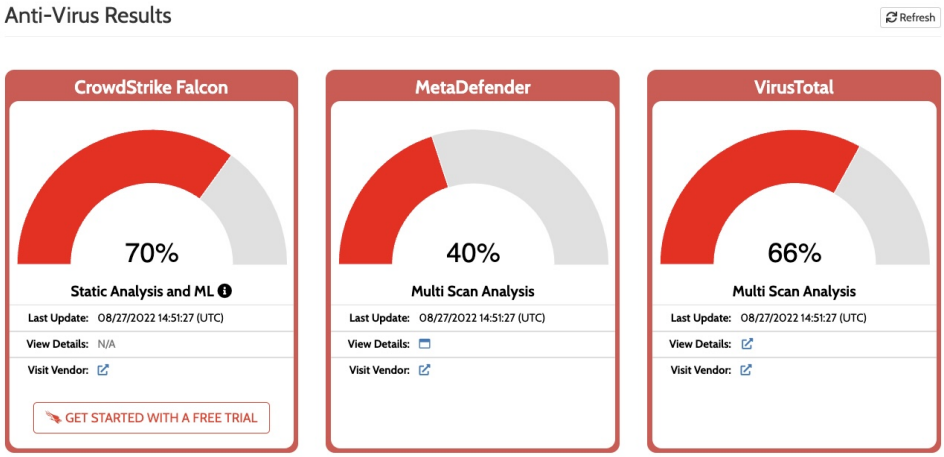


Figura 6.13. Comparazione Anti-Virus Scanner Results da Hybrid-Analysys

  - **Related Hashes**: ritorna I valori sha256 noti, relativi al malware in input o ai file estratti durante la detonazione.
  - **Falcon Sandbox Report**: ritorna un report dettagliato e completo di evidenze di categorie diverse. Nel report sono presenti, in particolare, indicatori di compromissione (IOC) con relativa mappatura MITRE ATT&CK, evidenze proprie dell'analisi statica di malware, flusso di esecuzione del malware, analisi di rete, stringhe estratte e file estratti con relativo livello di minaccia. Vengono, inoltre, fornite delle screenshot in grado di dettagliare a video i vari comportamenti del file malevolo.
  - **Incident Response**: ritorna informazioni sommarie di Risk Assessment.

- **Joe Sandbox Cloud**: tool open-source di analisi malware automatica disponibile in [129], in grado di eseguire file e url in un ambiente controllato , al fine di monitorare il comportamento delle applicazioni e del sistema operativo durante l'indagine forense [130]. I report generati da Joe Sandbox Cloud sono tra i più dettagliati e completi nel panorama OSINT ristretto all'analisi dei malware, focalizzandosi sul comportamento del sistema, della rete, del browser e della manipolazione del codice e consentendo ai professionisti della sicurezza informatica di distribuire, implementare e sviluppare strategie di difesa e meccanismi di protezione adeguati [131]. Una peculiarità di questo strumento risiede nel permettere l'analisi un file o url in input sui sistemi operativi Windows, Android, macOS e Linux, al fine di esaminare comportamenti diversi a seconda del sistema operativo del sistema infettato. Differentemente da molti strumenti OSINT Joe Sandbox Cloud è completamente privato, quindi nessun campione o dato di analisi viene condiviso o caricato a terzi, aumentando il livello di privacy nell'indagine. Tra la vastissima quantità di informazioni ritornate da Joe Sandbox Cloud, si citano in base alla particolarità:

  - **Signature Overview**: in questa sezione sono presenti tutte le firme riscontrate all'interno del file sospetto, opportunamente classificate e associate ad un grado di minaccia, visualizzabili ad esempio in Figura 6.15

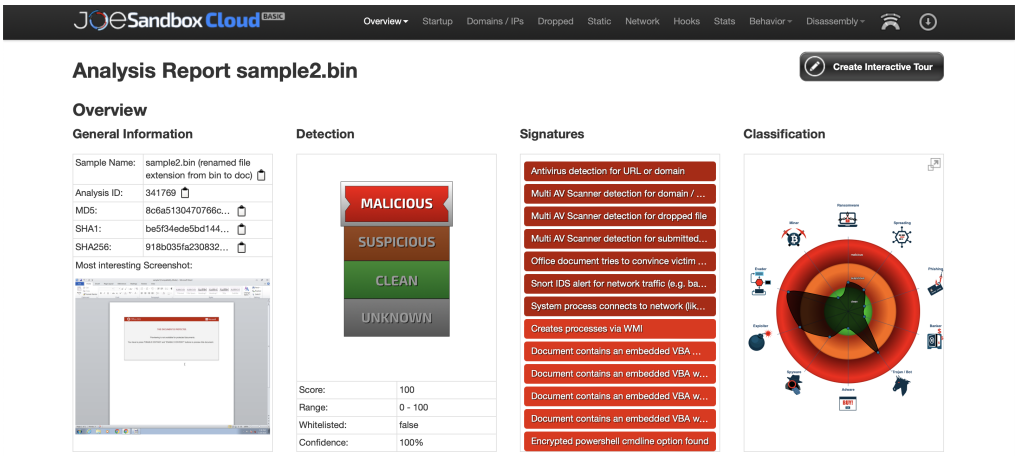


Figura 6.14. Signature Overview restituita da Joe Sandbox Cloud

- **Behavior Graph**: in questa sezione viene riportato un grafo rappresentante ogni fase del flusso di esecuzione del malware e il relativo comportamento. Ad ogni azione è associato uno specifico grado di minaccia.
- **Domain and IPs**: in questa sezione vengono riportati gli IP con cui il malware tenta di comunicare e la relativa geolocalizzazione, visualizzabile ad esempio in Figura 6.16

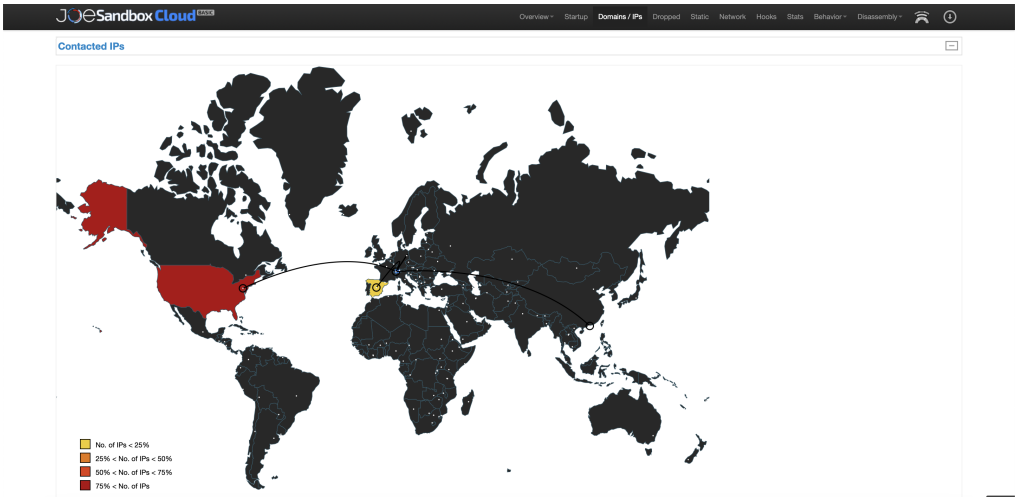


Figura 6.15. Geolocalizzazione IPs contattati da Joe Sandbox Cloud

- **Statistics**: in questa sezione vengono riportate delle statistiche calcolate durante l'esecuzione del file passato in input. Tra queste si registra la presenza di grafici dettaglianti la percentuale di utilizzo di CPU e memoria da parte del file sospetto e una distribuzione delle aree di interesse del file sospetto (file, registry, network), visualizzabile ad esempio in Figura 6.16.
- **Disassembly**: in questa sezione è possibile accedere ai risultati del disassemblaggio del file sospetto. Da qui è possibile accedere all'analisi delle funzioni del codice, propriamente categorizzare, e alla verifica delle relative firme tramite le YARA rule.

- **VirusTotal**: uno dei più famosi tool di analisi malware nel contesto OSINT. VirusTotal si offre come un servizio open-source volto all'analisi di file e URL alla ricerca di virus, worm, trojan e altri tipi di contenuti dannosi, allo scopo di rendere Internet un luogo più sicuro attraverso la collaborazione di una community di ricercatori e professionisti nel campo della sicurezza informatica [132]. Lo strumento, utilizzato principalmente per confrontare

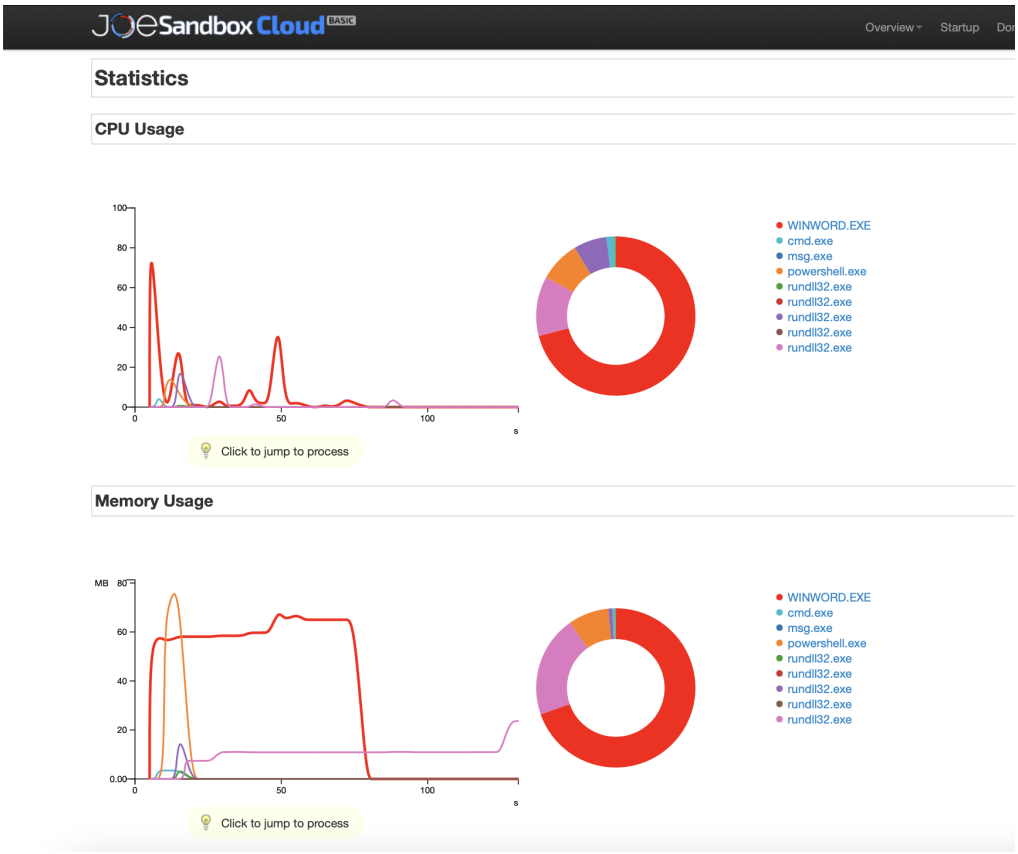


Figura 6.16. Statistiche legate all'esecuzione del file da Joe Sandbox Cloud

l'efficacia dei programmi antivirus, ispeziona l'elemento passato in input (file, url, hash, IP Address, Domain) con oltre 70 scanner antivirus e servizi di URL/domain blocklisting in aggiunta ad una vastissima gamma di strumenti utili all'estrazione di contenuti e dettagli di interesse [133]. VirusTotal può essere utile per individuare contenuti dannosi quanto per identificare i falsi positivi, ovvero elementi normali e innocui rilevati come dannosi da uno o più scanner. VirusTotal fornisce diverse possibilità di inoltro di file o URL, passando per l'homepage, le estensioni del browser e le API. Una limitazione, presente in molti strumenti di OSINT, risiede nella politica adottata da VirusTotal di memorizzare e condividere con i propri partner gli elementi passati in input, al fine di migliorare i propri sistemi e contribuire ad aumentare il livello di sicurezza informatica globale. Questo si traduce, però, in una limitazione alla privacy dell'autore della ricerca, per cui è consigliato non caricare nel sistema file con contenuti personali e/o sensibili. Se la pagina di "Detection", contenente il confronto dei risultati ritornati dagli antivirus, può servire unicamente per verificare la reale minaccia del file sospetto, più interessanti sono le schede di:

- **Details**: sono presenti informazioni generali del file stesso, come la data di creazione, il relativo hash calcolato in vari formati, le dll importate e ulteriori caratteri relativi alla struttura del file.
- **Relations**: sono presenti informazioni relative ai domini o indirizzi IP contattati dal file, al processo padre, ai file contenuti e rilasciati in fase di detonazione. Il tutto è riassunto mediante un apposito grafo, visualizzabile in Figura 6.17.
- **Behavior**: sono presenti informazioni ritornate dall'analisi in 4 sandbox (C2AE, QiAnXin RedDrip, MS Sysinternals, Zenbox), come visualizzabile in Figura 6.18. In questa sezione il malware viene classificato in base alla famiglia di appartenenza, si associano le relative "MITRE Signature", si descrivono i comportamenti di rete e le modifiche apportate al filesystem, se ne descrivono i comportamenti in termini di "Execution", "Persistence", "Privilege Escalation", "Defense Evasion", "Discovery", "Command

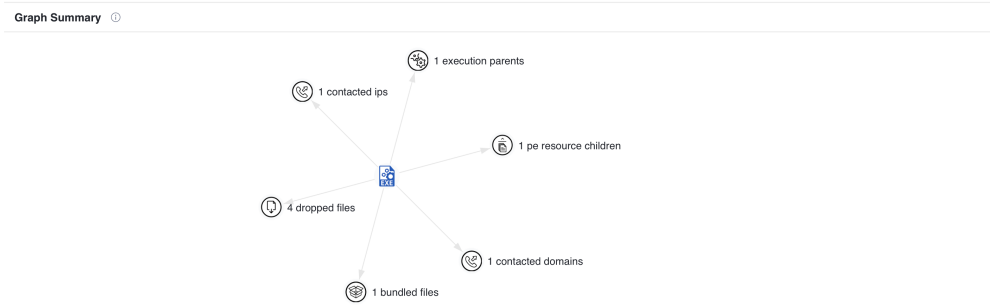


Figura 6.17. Graph Summary ritornato da VirusTotal

And Control". Infine viene mostrato il corrispettivo flusso di esecuzione, associato al "Process Tree".

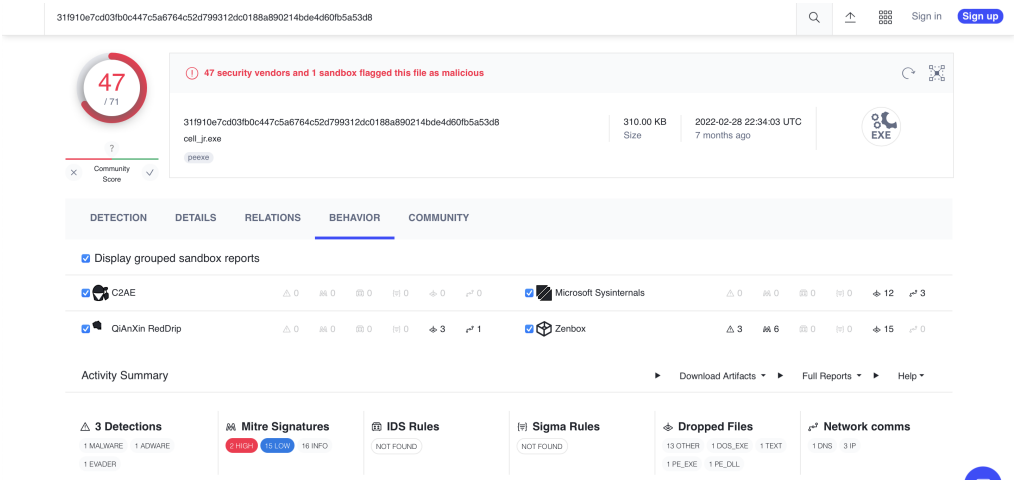


Figura 6.18. Confronto analisi C2AE, QiAnXin RedDrip, MS Sysinternals, Zenbox riportata da VirusTotal

- **MetaDefender Cloud**: servizio open-source di analisi malware e prevenzione avanzata delle minacce offerto da OPSWAT [134] e basato sul cloud. Questo strumento mette a disposizione una sandbox, combinata con l'analisi in tempo reale di hash, IP e domini, resa possibile dal database di Intelligence sulle minacce di livello mondiale di cui OPSWAt è proprietario [135]. MetaDefender Cloud supporta in input file, IP address, domain, hash o CVE e, in primo luogo, esegue una scansione multipla su 34 soluzioni antivirus, raffigurate ad esempio in Figura 6.19, in modo da poterne confrontare i risultati.

  Parallelamente, rende possibile visualizzare nella sezione "PE Information" evidenze digitali derivanti dall'analisi statica del malware, talvolta non possibile in assenza del relativo file eseguibile. Tra queste si citano:

  – **Version Information**: utili per ottenere dati sul "Vendor" o chi ha firmato il file
  – **PE Header Basic Information**: utili per comprendere il timestamp di prima compilazione del file sospetto,
  – **PE Sections**: utili ad ottenere indirizzo virtuale, dimensione ed entropia di ogni sezione del file. Qui è possibile identificare se il file è stato passato ad un software Packer o meno.
  – **PE Imports**: utile per identificare le librerie importate dal file in analisi

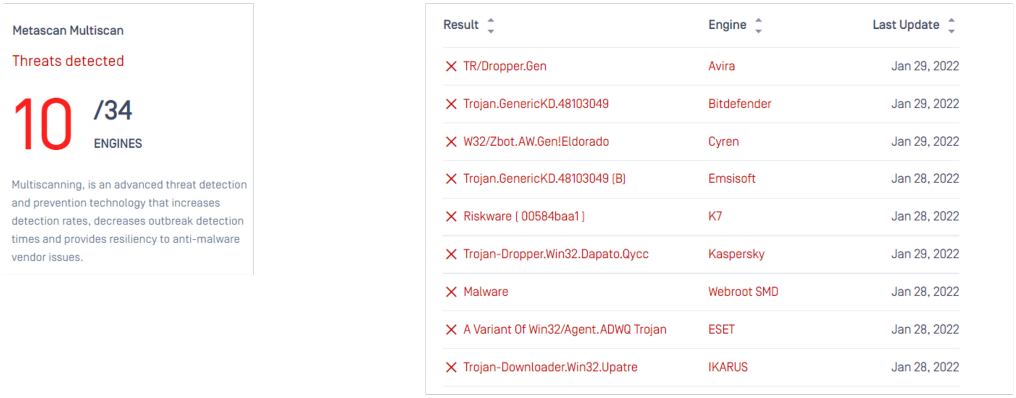


Figura 6.19. Scansione multipla su 34 soluzioni antivirus

6.1 Ambiente di analisi

Il framework in presentazione è stato testato in un ambiente controllato e sicuro, composto da una coppia di macchine virtuali, configurate nel medesimo modo: la prima utilizzata per la detonazione dei malware e per la cattura del memory dump, la seconda utile alle varie fasi di analisi. Le macchine virtuali, che sfruttano VirtualBox come applicativo di virtualizzazione, montano:

- Sistema Operativo Windows 10 a 64 bit
- 2 processori, utilizzabili al 100% dalla macchina virtuale
- Memoria RAM di 5 GB
- Host-only network adapter

La configurazione dell'ambiente di esecuzione è disponibile in Figura 6.20.

6.2 Testing del framework

In questa sezione si intende riportare i risultati ottenuti dall'analisi, mediante il framework presentato, di tre malware con caratteristiche stealth: cell_jr, emotet e stuxnet. Tra questi si riconosce la caratteristica di cell_jr, a differenza degli altri due vettori citati, di essere un file-based malware, con proprietà stealth. Il motivo di inclusione di quest'ultimo risiede nella volontà di confermare quanto il framework presentato possa essere utilizzato in equal modo per il rilevamento e l'analisi di malware completamente stealth e non. I risultati ottenuti sono riportati successivamente seguendo i passi del processo forense, che ha contribuito alla raccolta e analisi dei dati, finalizzata al rilevamento e all'esamina comportamentale dei malware citati. Come si potrà notare, le prime due fasi di Data Collection e Data Examination sono descritte a titolo collettivo, non essendoci state in queste specifiche differenze ottenute a seconda del malware in analisi, mentre le fasi di Data Analysis e Reporting saranno dipendenti dal malware sotto investigazione.

Data Collection: in seguito alla detonazione del malware in ambiente controllato, si procede alla cattura dell'immagine di memoria RAM, al fine di poterla analizzare in ambiente distinto mediante gli strumenti di Memory Forensics. Il file memdump.mem generato ha una dimensione di 5.50 GB e viene salvato nel percorso "E:\Mem". Come è possibile notare, la dimensione risultante del file memdump.mem è maggiore della dimensione totale della memoria RAM. Questa discrepanza è dovuta al formato del file ed è diversa a seconda del tool di "memory capture" utilizzato, come descritto in [136]. In particolare il testing di Stuxnet è stato effettuato direttamente su un memory dump pubblicamente disponibile, al contrario degli altri due malware detonati in ambiente controllato. Non è stato possibile detonare Stuxnet in ambiente virtualizzato in quanto

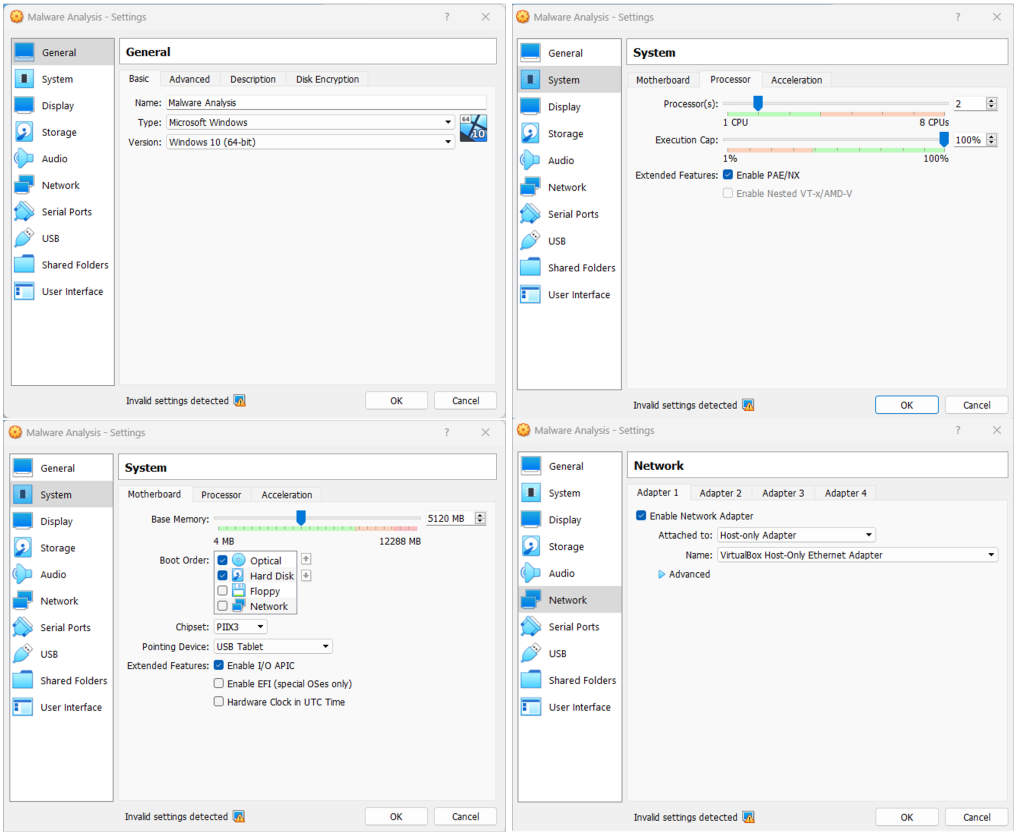


Figura 6.20. Configurazione ambiente controllato

esso necessita di un ambiente specifico per esplodere tutte le sue funzionalità. Questa decisione è stata presa mantenendo la volontà di focalizzare principalmente questo progetto sull'analisi forense, più che sulla "Malware Analysis", simulando la fase di ricezione del memory dump, in questo caso risalente all'anno 2010.

Data Examination: in seguito alla collezione del materiale da analizzare, che nella tipologia di investigazione forense in presentazione si identifica con l'immagine di memoria volatile catturata nella fase precedente, si procede alla valutazione e selezione dei dati digitali in essa contenuti. Lo scopo principale di questo processo risiede nel focalizzarsi unicamente sulle evidenze etichettate da J. Yaacoub in [4] come "Relevant", minimizzando l'insieme delle risorse disponibili allo stretto necessario all'analisi. In tal modo, vengono reputati di maggiore interesse forense per il rilevamento e l'analisi di malware di tipo stealth le seguenti categorie di artefatti digitali:

- **Informazioni sul sistema infetto**: al fine di identificare le caratteristiche del sistema nel quale il malware ha avuto successo. E' importante notare come gli autori di malware spesso focalizzano l'attacco su specifici sistemi operativi (Es. Windows), rendendolo inefficace su altri (Es. macOS)

- **Elenco dei processi in memoria**: al fine di stabilire quali processi contribuiscono al flusso di esecuzione del malware sucessivamente al timestamp di detonazione del malware, se disponibile

- **Elenco dei comandi eseguiti da un processo**: al fine di stabilire il flusso cronologico di comandi eseguiti dai processi in seguito al timestamp di detonazione del malware, se disponibile

- **Elenco delle connessioni di rete**: al fine di stabilire il flusso cronologico di trasmissione e ricezione nel canale di comunicazione instaurato dal malware succesivamente alla detonazione. Questi artefatti evidenziano specifiche attività di Command&Control (C&C)
- **Elenco delle chiavi di registro**: al fine di identificare recenti modifiche attuate dal malware per procederei nei propri fini e, in particolare, per ottenere la proprietà di persistenza.
- **Elenco delle stringhe note come "malevole"**: al fine di individuare la presenza di modelli di malware o famiglie di malware, noti per relativi set di istruzioni comuni
- **Elenco dei file caricati in memoria**: al fine di identificare quali file vengono caricati in memoria dal malware, associati ai relativi metadati.
- **Elenco dei process handle**: al fine di rilevare eventuali handle agganciati ai processi, interni al flusso di esecuzione del malware, ed usati come riferimento ad essi in ogni chiamata a funzione [137]
- **Elenco delle entry nella SSDT**: al fine di identificare eventuali casi di "SSDT Hooking", utilizzata dai malware di tipo stealth per ottenere la capcità di offuscamento
- **Elenco dei mutant**: al fine di individuare gli oggetti kernel mutant, o mutex, di cui è noto lo sfruttamento delle capcità di sincronizzazione tra processi, allo scopo di non reinfettare il sistema più di una volta
- **Elenco dei kernel driver**: al fine di individuare i driver dei dispositivi che possono essere nascosti dai rootkit
- **Elenco delle "Sections"**: al fine di visualizzare le varie sezioni che compongono il malware in analisi, in aggiunta al relativo Packer e grado di entropia
- **Elenco degli "Imports"**: al fine di visualizzare l'insieme di API e DLL importate dinamicamente e utilizzate dal malware per per recare danno al sistema
- **Elenco delle "Strings"**: al fine di identificare le stringhe in formato human-readable presenti nel binario del malware in analisi
- **Elenco delle "Resources"**: al fine di identificare le risorse rilasciate dal malware sul sistema

6.3 Cell_jr

La fase di **Data Analysis** si è focalizzata nell'estrazione delle evidenze digitali presenti nel sistema, e in particolare nella memoria RAM, successivamente alla detonazione del malware riconosciuto come cell_jr.exe. Si riportano di seguito le evidenze digitali rilevate, sulla base della selezione definita nella fase di Data Examination:

- **Informazioni sul sistema infetto**: è stato identificato Windows 10 x64 come sistema operativo della macchina infettata dal malware, con l'ausilio del Volatility plugin info, come rappresentato in Figura 6.21. Si riscontra, inoltre la presenza di due processori, la root del sistema operativo configurata sul drive C:\Windows e il time-stamp di creazione della macchina virtuale in "6 Jul 2018 6:57:56", precedente alla data di inizio analisi del malware, in quanto snapshot di una macchina virtuale contenente diversi tool di malware analysis. Si nota inoltre come l'indirizzo del Kernel in memoria sia riconosciuto in 0xf802c0492000
- **Elenco dei processi in memoria**: è stato identificato, mediante i plugin Volatility pslist, pstree e psscan, il flusso di esecuzione del malware, che si avvia alle 18:22:29 del 23-08-22 dalla detonazione del file "Sample-20200214" (PID 1504). In successione, come mostrato in Figura 6.22, vengono eseguiti i processi malevoli "notepad.exe" (PID 3704) e "cell_jr.exe" (PID 5524). Il flusso di esecuzione è stato confermato durante l'analisi OSINT mediante Hybrid-Analysis [138] e VirusTotal [139]. Si può inoltre notare come il malware sia stato

```
PS C:\Users\student\Desktop\volatility3> python vol.py -f E:\Mem\memdump.mem windows.info
Volatility 3 Framework 2.3.0
Progress:  100.00               PDB scanning finished
Variable        Value

Kernel Base     0xf802c0492000
DTB     0x1aa000
Symbols file:///C:/Users/student/Desktop/volatility3/volatility3/symbols/windows/ntkrnlmp.pdb/311A83B581114CD
0BEB21F065438BFAA-1.json.xz
Is64Bit True
IsPAE   False
layer_name      0 WindowsIntel32e
memory_layer    1 FileLayer
KdVersionBlock  0xf802c0838d50
Major/Minor     15.17134
MachineType     34404
KeNumberProcessors      2
SystemTime      2022-08-23 18:22:42
NtSystemRoot    C:\Windows
NtProductType   NtProductWinNt
NtMajorVersion  10
NtMinorVersion  0
PE MajorOperatingSystemVersion  10
PE MinorOperatingSystemVersion  0
PE Machine      34404
PE TimeDateStamp        Fri Jul  6 06:57:56 2018
```

Figura 6.21. Informazioni del sistema infettato da cell_jr.exe

eseguito 3 volte, allo scopo di testing, dalla ripetizione del medesimo flusso di esecuzione. In particolare, durante l'analisi dinamica mediante IDA e con l'ausilio di ProcMon, viene rilevata nel codice disassemblato la subroutine sub_401230, all'interno della quale si riscontra la creazione del processo "notepad.exe", con cui si porta avanti l'injection

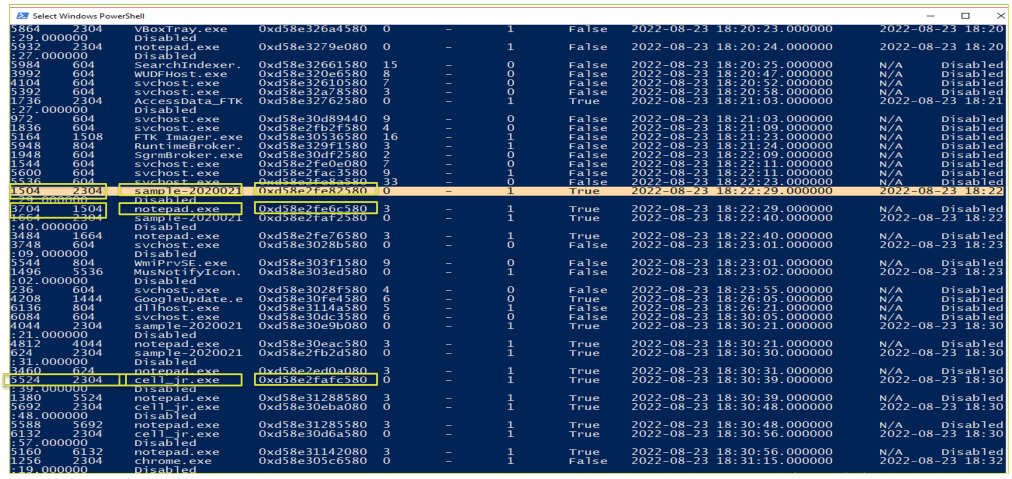


Figura 6.22. Flusso di esecuzione del malware cell_jr.exe

- **Elenco dei comandi eseguiti da un processo**: il flusso di esecuzione riscontrato nel punto precedente è stato confermato eseguendo il plugin cmdline. In esso si riscontra il passaggio dei processi con PID 1504, 3704, 5524 al processo cmd.exe. L'esecuzione del plugin UserAssist ha riscontrato entry con valori offuscati, precisamente agli indirizzi 0xa40ad6e62000, 0xa40adee41000, 0xa40ad6478000. Nel medesimo output si evidenzia in un timestamp leggermente successivo alla detonazione, un'operazione nella registry key userAssist con l'inserimento della entry Microsoft.LockApp_cw5n1h2txyewy!WindowsDefaultLockScreenG. In [140] si esplicita quanto sia solito per i malware della famiglia Trojan iniettare codice nei processi di sistema %WINDIR%\systemapps\microsoft.lockapp_cw5n1h2txyewy\lockapp.exe, che rappresenta il processo corrispettivo al Default Lock Screen. In [141] si articola come spesso questo processo venga utilizzato dai malware per il proprio camuffamento e, quindi, al fine di ottenere proprietà stealth

- **Elenco delle connessioni di rete**: è stato identificato come il processo notepad.exe (PID 3704) tenti di avviare una connessione di rete UDPv4 nel preciso istante della detonazione del malware, eseguendo il Volatility plugin netscan, come illustrato in Figura 6.23. Tale connessione è confermata ulteriormente visionando le proprietà TCP/IP del file notepad.exe

mediante Process Explorer, come mostrato in 6.7. A tal riguardo, non è possibile identificare l'indirizzo di destinazione dal memory dump o da Process Explorer, essendo vittima di offuscamento. Eseguendo il plugin mftscan è stata rilevata, e illustrata in Figura 6.24, la presenza di una connessione HTTP, immediatamente successiva all'istante di detonazione del malware, con stato "Network Persistent State", ovvero che rimane aperta per ulteriori richieste e risposte HTTP, senza chiudersi dopo un singolo scambio di dati. L'analisi dinamica, mediante IDA, ha evidenziato la presenza di una DNS Request, successivamente alla chiamata alla funzione GetCurrentProcessId, indirizzata all'IP 8.8.8.8, con l'invio di dati mediante la funzione sendTo. Questa connessione UDP viene rilevata ulteriormente in Wireshark, con IP sorgente 10.0.2.15, IP destinazione 8.8.8.8, protocollo DNS e dimensione 53 B, come riprodotto in Figura 6.10. L'analisi OSINT mediante Hybrid-Analysis in [138] conferma la trasmissione di dati verso un indirizzo ip. Al momento della detonazione, si riscontra in Fakenet una nuova connessione aperta alla porta 80 HTTP verso l'host "canonicalizer.ucsuri.tcs", con riferimento "Keep-Alive" e inizializzata dallo User-Agent SmartScreen/2814750890000521. In essa, descritta in Figura 6.11, viene rilevata una POST HTTP/1.1 di 851 B. Si identifica questa connessione come parte del sistema SmartScreen anti-malware di Windows in [142]. Si rileva in Fakenet, inoltre, una DNS query con dominio "http://67fe471ceebc410c8cf9095d8a422316.clo.footprintdns.com", con relativa DNS Response.

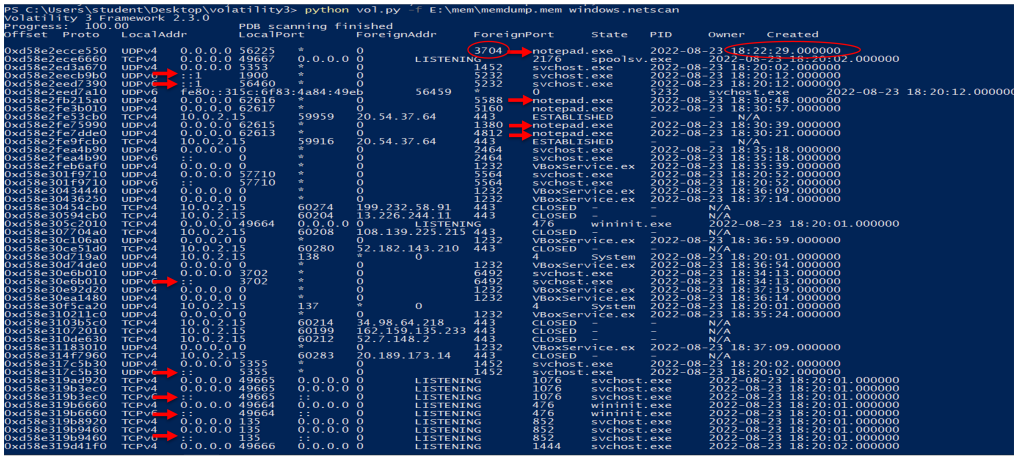


Figura 6.23. Connessioni di rete in memoria

Figura 6.24. Impronta di una connessione di rete all'interno della Master File Table (MFT)

- **Elenco delle chiavi di registro**: sono state identificate, eseguendo il Volatility plugin registry.printkey e filtrando il tempo di ultima modifica in modo da essere successivo alla data di detonazione del malware come mostrato in Figura 6.25, corruzioni interne alla registry key
  \SystemRoot\System32\Config\DEFAULT. Si notano inoltre corruzioni alla registry key \??\C:\Users\student\ntuser.dat. Quest'ultima, seppur precedente alla data di esecuzione del malware è correlata ad un artefatto riscontrato durante l'analisi dinamica del malware mediante l'applicativo IDA, per questo se ne giustifica la presenza in occasione di una possibile detonazione out-of-analysis al fine di testing. In aggiunta l'hive relativo a quest'ultima chiave di registro corrisponde al medesimo sito in 0xa40ad6e62000 identificato in forma offuscata nell'output dei valori relativi alla chiave UserAssist, di norma localizzata in

NTUSER.DAT\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist [143]. E' noto come l'estensione .dat sia usata per i file in cui sono memorizzate le informazioni di esecuzione da eseguire ad ogni avvio e, di conseguenza, come tali file siano spesso corrotti dai malware per modificare le impostazioni di boot ed ottenere la caratteristica di persistenza, nota delle famiglie stealth. Eseguendo il plugin hivelist, si identifica la presenza in memoria di diversi file con estensione .dat, che possono essere stati modificati per ottenere tali fini. Inoltre si riscontra la presenza del file Amcache.hve, spesso usato per registrare le tracce di programmi anti-forensi, Pe file e dispositivi di archiviazione esterni [144] Si rileva, inoltre, la corruzione della chiave di registro "WALLPAPER", seguendo il path "HKCU\CONTROL PANEL\DESKTOP". Il valore in essa contenuto viene sovrascritto in "%USERPROFILE%\Pictures\cell_jr.jpg", ovvero allegando il percorso dell'immagine "cell_jr.jpg", che verrà impostata come immagine desktop. Questa evidenza viene confermata da Hybrid Analysis in [138].

Figura 6.25. Focus sulle chiavi di registro caricate in memoria

- **Elenco delle stringhe note come "malevole"**: sono stati identificati riscontri su diversi modelli di modelli di malware, eseguendo il Volatility plugin vadyarascan. In particolare, i moduli di regole yara su cui si è trovato riscontro sono i seguenti:

  - **with_sqlite**: regola per rilevare la presenza di dati SQLite nella raw image [145]
  - **Spyeye plugins**: regola per rilevare la presenza di dll utilizzate comunemente nei malware banker, come Spyeye. I valori di stringhe riscontrati sono $o, equivalente a "rdp.dll", e $b, equivalente a "config.dat". Viene, in ogni caso, descritta la presenza di possibili False Positive (FP) in [146]
  - **SharedStrings**: regola per rilevare la presenza di modelli interni trovati nei sample relativi a LURK0/CCTV0. Si riscontra il valore $m4, che tuttavia non viene descritto in [147]
  - **Memory Shylock**: regola per identificare modelli interni propri del Trojan bancario Shylock. Si riscontra, in particolare, il valore $b, corrispondente all'URI beacon, o porzione, di Shylock ed equivalente a "/id=[A-F0-9]32/" [148]

  - **Bolonyokte**: regola per identificare modelli interni al malware Bolonyokte. Si riscontrano i valori $ebanking4 = “login”, $ebanking5 = ”en ligne”, $ebanking12 = “Power“
  - **UPX**: regola per identificare l’utilizzo del Packer UPX, al fine di offuscamento del codice. In particolare si riscontrano i valori $a = “UPX0”, $b = “UPX1”, $c = “UPX!”. Si identificano quindi due sezioni all’interno della versione packed del file eseguibile. UPX! viene posto comunemente al termine del PE Header [149].
  - **lsadump**: regola per identificare stringhe nel dominio del Local Security Authority (LSA), responsabile per l’autentificazione degli utenti di un sistema e dei loro privilegi [150]. In particolare, si riscontra il valore : $str_sam_inc = ”\\Domains\\Account”.

- **Elenco dei file caricati in memoria**: sono stati identificati i file in memoria caricati in risposta alla detonazione del malware, eseguendo il Volatility plugin filescan. E’ possibile riscontrare il file eseguibile “cell_jr.exe” creato dal malware durante la detonazione nel percorso “Users/Student/Documents”. Successivamente all’esecuzione del sample vengono caricati in memoria file con estensione .dat, il cui valore potrebbe essere corrotto. Si cita a tal riguardo il file settings.dat relativo al Package di MicrosoftCalculator. Molti valori rilasciati dal plugin filescan appaiono offuscati, per cui si esplicita il sospetto di relative tecniche di mascheramento. In particolare, si nota il file “Sample-20200214-unpacked.exe”, contenente il PE del malware nella versione unpacked rilasciata da Scylla, nel percorso “\Exam_Papers\new\exam-20200214”. L’esecuzione del plugin mftscan rende possibile ottenere il contenuto della Master File Table, presente nel NTFS filesystem, e ha riscontrato la presenza delle versioni packed e unpacked del sample. In particolare si nota l’occorrenza del file “SAMPLE 2.PF”, che riporta l’estensione propria dei file password-protected crittati da Stuffit Deluxe. Si rileva, quindi, l’utilizzo della crittografia come metodo di offuscamento del codice. I sample sono, inotre, presenti in nomi di file con estensione .gzf/.qzf, usati per indicare quale applicativo può aprire il file. Filtrando la ricerca nella MFT in modo da soddisfare la stringa “cell_jr”, sono stati trovati 5 offset relativi al medesimo file eseguibile, tutti caricati contemporaneamente nell’istante della detonazione e visibili in Figura 6.26. La presenza di diversi file cell.exe nel filesystem è stata confermata ulteriormente durante l’analisi dinamica, mediante IDA e ProcMon. In essa si rileva come nella sub_4016A0 il malware copia in READ_ONLY se stesso in varie cartelle, di cui l’elenco dei percorsi contenenti cell_jr.exe:

  - C: \Users\student\Documents\
  - C: \Users\student\AppData\Local\
  - C: \Users\student\Favorites\
  - C: \Users\student\Pictures
  - C: \Users\student\AppData\Roaming\ Microsoft\Windows\StartMenu\Programs\StartUp
  - C: \Users\student\AppData\Roaming\Microsoft\Windows\Start Menu\
  - C: \Users\student\Videos

  In particolare si può notare la presenza all’interno di C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp, contenente i file che devono essere eseguiti all’avvio. Questo è sicuramente un meccanismo per ottenere la persistenza. Si osserva, mediante analisi del filesystem e confermato successivamente da VirusTotal [139], il caricamento nel percorso “C: \Users\student\Pictures” dell’immagine “cell_jr.jpg”, che verrà impostata come immagine Desktop, come mostrato in Figura 6.27

- **Elenco delle entry nella SSDT**: sono stati identificate diverse entry nell’output restituito dall’esecuzione del Volatility plugin ssdt che potrebbero essere correlate all’esecuzione del malware. In particolare le minacce informatiche utilizzano i metodi C&C per ottenere il massimo delle informazioni sul dispositivo. A tal riguardo si citano:

  - **NTGetPnpProperty**: se viene specificata una proprietà, il suo valore viene restituito in uscita dal server

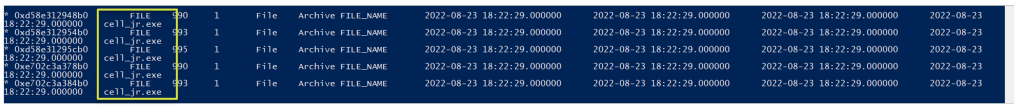


Figura 6.26. File cell_jr.exe caricati in Master File Table (MFT)

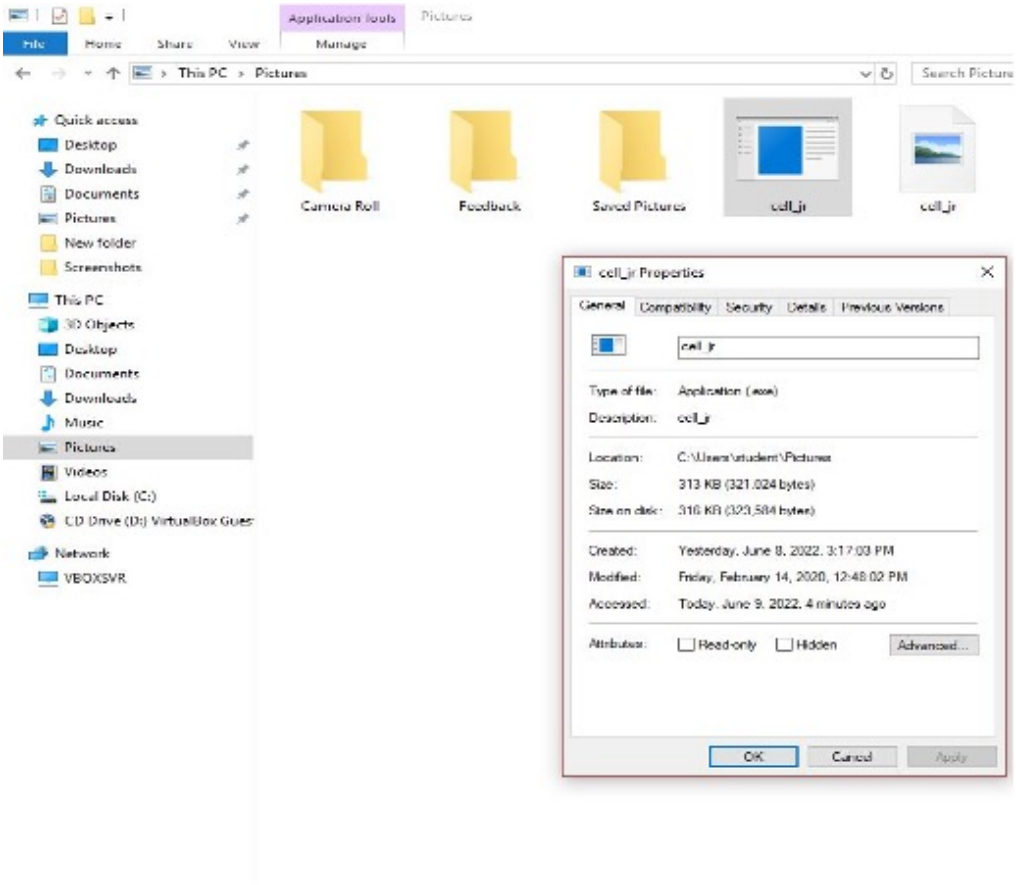


Figura 6.27. Contenuto della cartella Users/student/Pictures

- **NTGetProcessorId**: recupera la classe di priorità del processo specificato
- **NTGetContextThread**: recupera il contesto del thread specificato
- **NTGetCurrentProcessorNumber**: fornisce informazioni per stimare le prestazioni del processo corrente
- **NTGetCurrentProcessorNumberEx**: recupera un elenco di processi in esecuzione mantenuto dal sistema
- **NTGetDevicePowerState**: determina se un dispositivo è in working state o di basso consumo
- **NTGetMUIRegistryInfo**: ogni volta che si inizia a utilizzare una nuova applicazione, il sistema operativo Windows estrae automaticamente il nome dell'applicazione dalla risorsa di versione del file .exe e lo memorizza per utilizzarlo in seguito.
- **NTGetNextProcess**: restituisce il prossimo processo all'interno della lista concatenata
- **NTGetNextThread**: restituisce il prossimo thread dall'interno della lista concatenata
- **NTGetNlsSectionPtr**: determina il comportamento specifico locale sia sul client che sul server

  - **NTGetNotificationResourceManager**: recupera la prossima notifica di transazione dalla coda di notifica di un gestore di risorse specificato
  - **GetWriteWatch**: ripristina lo stato di tracciamento della scrittura per una regione di memoria virtuale

- **Elenco dei mutant**: sono stati identificati, dall'esecuzione del Volatility plugin mutantscan, i seguenti mutant, che potrebbero essere utilizzati dal malware per assicurarsi di non reinfettare la stessa macchina ed eseguire solo una singola copia del malware:

  - **BcdSyncMutant**: può utilizzare bcdedit per modificare le impostazioni di avvio di Windows
  - **WilStaging_02/WilError_01**: usato per C&C dai Remote Access Trojan (RAT) [151]
  - **_SHuassist.mtx**: può avere la capacità di alterare i file, la manipolazione dei processi e le connessioni di rete [152]
  - **ZonesCacheCounterMutex, ZonesLockedCacheCounterMutex**: Crea i mutex per assicurarsi che sia in esecuzione una sola istanza di se stesso[153]
  - **DBWinMutex**: è usato solo da OutputDebugString(), per evitare che più thread scrivano contemporaneamente sul buffer di output. Non è necessario che un monitor di debug utilizzi DBWinMutex [154]
  - **MidiMapper_modLongMessage_RefCnt**: può creare il file malware e, se lo trova di nuovo, può sostituire il file con la stessa posizione

- **Elenco dei kernel driver**: sono stati identificati, mediante l'esecuzione del Volatility plugin modscan, i seguenti kernel driver:

  - **USBSTOR**: driver della porta di archiviazione USB, viene utilizzato quando l'unità USB è collegata e questo file inizia a comunicare con essa
  - **USBXHCI**: richiede l'accesso diretto alle finestre interne
  - **Volsnap.sys**: driver di Windows che consente al computer di comunicare con l'hardware o i driver collegati
  - **Volume.sys**: è utilizzato per la comunicazione con l'hardware
  - **usbhub.sys**: è il driver per ogni hub USB. Viene caricato se l'enumeratore del bus PCI rileva un hub USB. Espone l'interfaccia del driver USB

  Questi artefatti, illustrati in Figura 6.28 introducono al pensiero che una chiavetta USB sia stata introdotta precedentemente all'esecuzione del malware e possa essere stata utilizzata come vettore di attacco del malware. Si conferma, infatti, l'utilizzo di una chiavetta USB come vettore del malware.

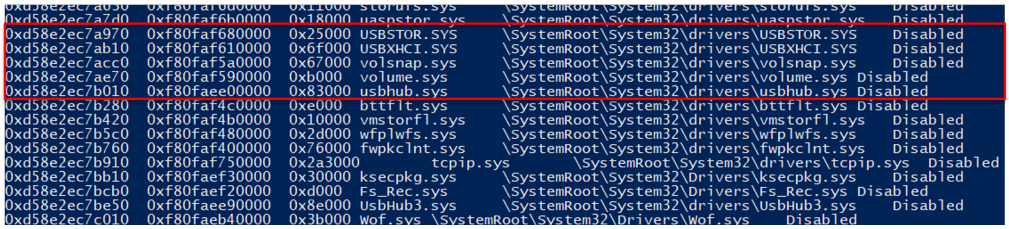


Figura 6.28. Kernel driver caricati in memoria

- **Elenco delle "Sections"**: sono state identificate, mediante l'applicativo PeStudio, tre sezioni all'interno del PE file, con il nome di MPRESS1, MPRESS2 e rsrc, illustrate in Figura 6.4. Si riscontra quindi l'utilizzo del MPRESS Packer, a differenza di come identificato nella yara rule UPX, che anche se generando un falso positivo aveva, ad ogni modo, identificato due sezioni di valore. L'utilizzo del Packer MPRESS è confermato ulteriormente da Detect

It Easy, che ne identifica la versione in MPRESS 2.19, come descritto in Figura **??**. La prima sezione ha un'entropia di 7,5, mentre la seconda ha un'entropia pari a 5,6 e contiene l'EOP, come raffigurato in Figura 6.4. Su entrambi le sezioni si hanno permessi di scrittura ed esecuzione, segno che il sample andrà a scrivere dati su di esse durante l'esecuzione. Ulteriore segno della presenza di un malware risiede nella differenza dei valori nei campi ''virtual-size" e ''raw-size" di ogni sezione. Le evidenze sono confermate, in aggiunta, dall'analisi OSINT mediante Hybrid-Anlsysis [138], convalidando il sospetto di tecniche di anti-reverse engineering utilizzate dal malware.

- **Elenco degli "Imports"**: è stata identificata, mediante il programma PeStudio, la lista di 11 Windows API importate dal malware (IAT) dalla quale è possibile comprenderne il comportamento dopo aver compromesso il sistema, disponibile in Figura 6.5. Tra queste API si citano GetModuleHandleA e GetProcAddress, che vengono usate per la procedura di unpacking. La lista delle dll importate e associate alle API riscontrate è la seguente:
  - api-ms-win-crt-math-l1-1-0.dll
  - api-ms-win-crt-string-l1-1-0.dll
  - api-ms-win-crt-runtime-l1-1-0.dll
  - KERNEL32.DLL
  - VCRUNTIME140.dll
  - SHELL32.dll
  - api-ms-win-crt-stdio-l1-1-0.dll
  - api-ms-win-crt-locale-l1-1-0.dll
  - api-ms-win-crt-heap-l1-1-0.dll
  - USER32.dll

  Alcune di esse sono state ritrovate in RAM, in aggiunta, eseguendo il plugin dlllist.

- **Elenco delle "Strings"**: si identificano 52 stringhe in formato human-readable presenti nel binario del malware in analisi, mediante l'applicativo PeStudio. In particolare si citano come rilevanti all'analisi la OpenProcess, CopyFile, CreateFile, ExitProcess

- **Elenco delle "Resources"**: sono identificato 3 risorse rilasciate nel sistema, mediante l'applicativo PeStudio. Si riscontrano 2 file .JPG, 1 file .PUH, 1 manifest. L'analisi dinamica mediante IDA ha, in particolare, evidenziato il percorso di una di queste risorse in C:\Users\student\Pictures\pride_troopers.jpg Si tratta del file usato come immagine desktop

L'analisi dinamica svolta mediante l'applicativo di reverse-engineering IDA, con l'ausilio del tool Scylla, ha localizzato l'indirizzo del tail jump ad 0x00451577: si tratta di un 'unconditional jump', eseguito successivamente alla procedura di unpacking. Inserendo un breakpoint e runnando, il codice salta in una locazione che sembra essere il prologo di un programma, ovvero una sezione di codice ASM che si trova all'inizio di una funzione e si occupa di preparare lo stack e i registri per il relativo utilizzo. Successivamente viene riconosciuto l'indirizzo dell'OEP in 0x00401B54 e viene confermato da Scylla, che in aggiunta riconosce l'IAT con dimensione 584 B nell'indirizzo 0x00402BC0. L'analisi dinamica ha, inoltre, evidenziato la presenza della suddivisione del codice in due parti. Nella prima viene effettuato un controllo sui processi(Sub_401180 e Sub_401000) e viene copiata l'immagine da impostare nel Desktop nella cartella apposita (Sub_401210, Sub_401120, Sub_401860). Il controllo sui processi avviene chiamando la funzione K32EnumProcesses (Sub_401180) e iterando nella lista ritornata mediante i metodi K32EnumProcessModules e K32GetModuleBaseNameA (Sub_401000). Se si riscontra nel ciclo una sottostringa di processi che potrebbero comportare problemi all'esecuzione (Sub_401210), mostrati in Figura 6.29, si esce dalla funzione main (sub_401120), altrimenti si procede a copiare l'immagine del sample in cartelle distinte (Sub_4016A0, Sub_401650, Sub_401610), a creare il processo notepad.exe e a copiare nel filesystem la mirai.dll (Sub_401300 , Sub_401230, Sub_4012B0). Prima di uscire da questa fase viene chiamata la SystemParametersInfoA, con parametro 14h,

che comporta la modifica dell'immagine desktop. Nella seconda parte avviene l'esecuzione della dll iniettata tramite un thread creato appositamente. Si riscontra lo startAddress del thread (Sub_10001280) e la registry query che viene richiesta (Sub_10001560). La dll injection eseguita al processo notepad.exe può essere suddivisa, a sua volta, in due fasi.

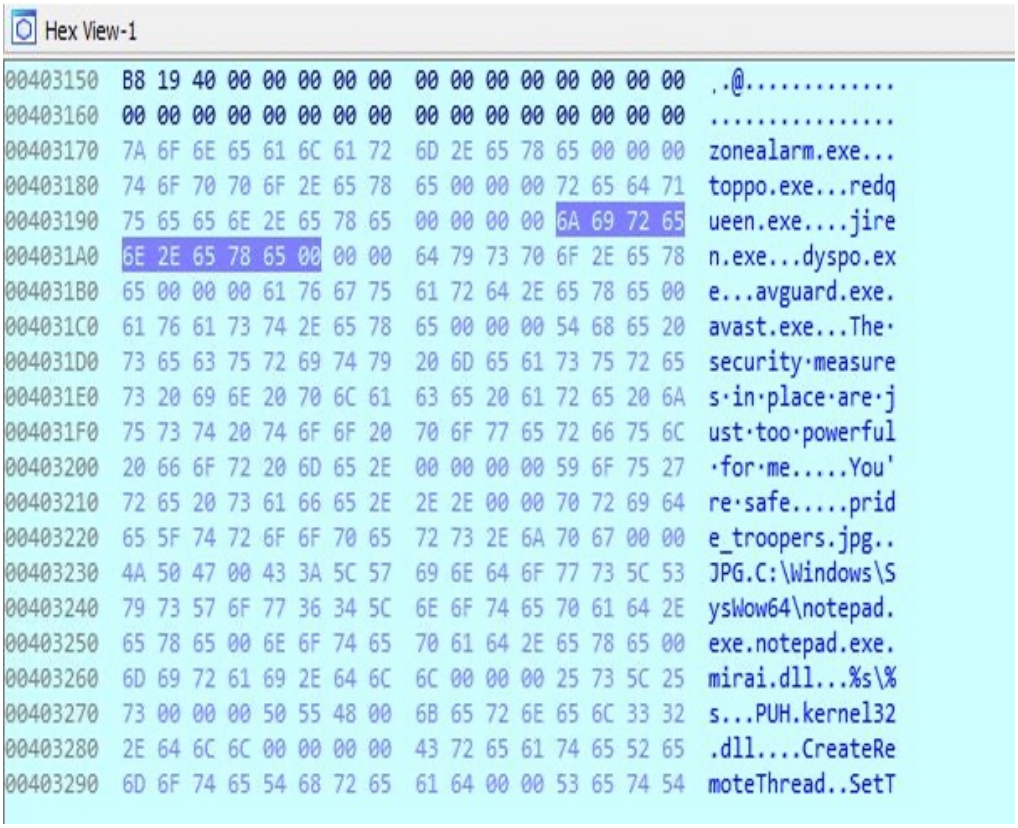


Figura 6.29. Controllo sui processi in esecuzione

Nella prima fase avviene:

1. Creazione di istanze di notepad.exe (sub_401230)
2. Estrazione del codice malevolo dalla risorsa 113 (sub_401750), andando a copiare il contenuto della risorsa in "C:\Users\student\mirai.dll" (sub_4012B0)

Nella seconda fase, a sua volta, avviene:

1. Chiamata di GetProcAddress più volte, al fine di ottenere tutte le funzioni richieste per l'injection
2. Chiamata di VirtualAllocEx , al fine di allocare memoria nel processo vittima
3. Chiamata di WriteProcessMemory, al fine di scrivere il path per mirai.dll
4. Chiamata di CreateRemoteThread, passando la funzione LoadLibrary e il path per mirai.dll

Il malware in analisi procede, in aggiunta, con diverse tecniche di offuscamento presentando quindi ulteriori caratteristiche stealth, anche se l'analisi ha evidenziato la sua natura "file-based".

Al fine di rendere i dati testuali e binari più difficili da comprendere avviene:

- Creazione di multiple copie dei file cell_jr.exe

- Caricamento e inserimento in un array di diverse funzioni tramite la GetProcAddress, rendendo complicata la compresione di quali, tra esse, contribuiscano realmente al comportamento del malware
- Codifica del nome LoadLibrary (sub_401300)
- Codifica del nome "Kernel32.dll"
- Offuscamento dell'indirizzo IP di destinazione della comunicazione di rete

Data Reporting: in questa fase viene mostrato un "summary" delle caratteristiche del malware, rilevate durante l'analisi post-mortem.

Malware name: cell_jr.exe, sample-2020021.exe

Malware family: Trojan.Generic [138], Adware Evader [139]

Threat score: 100% malevolo [138]

SHA256: 31f910e7cd03fb0c447c5a6764c52d799312dc0188a890214bde4d60fb5a53d8

MD5: ed46b37d56c2004519e454dca015f9a6

Size: 310 KB

File type: MS-DOS WIN32 executable

Creation Time: 2020-02-14 11:48:02 UTC

Sistema Operativo: Windows (x64)

Memory Dump: memdump.mem (5.50 GB)

Detonation time: 23-08-2022 18:22:29

Entrypoint: 0x401B54

Anti-Virus Results: 47/71 security vendors e 1 sandbox hanno indicato questo file come malevolo [139]

MITRE ATT&CK Techniques Detection: 14 attack techniques found [138], 17 MITRE signatures found (2 high, 15 low) [139]

MITRE ATT&CK ID: T1547.001, T1055.003, T1055, T1027.002, T1056.004, T1082, T1012, T1057, T1114, T1491, T1129, T1037.005, T1574.002, T1055, T1027.002, T1036, T1071, T1095

Installation/Execution: inietta dei file all'interno di applicazioni Windows (%WINDIR%\System32\notepa

Persistence: crea una voce nel menù di avvio in "\Users\student\AppData\Roaming\Microsoft\Windows\St
e rilascia un file eseguibile nella cartella "\Users\student\AppData\Roaming\Microsoft\Windows\StartMenu\Pro

Evasive: il malware presenta abilità di offuscamento di file e informazioni, mascheramento del file eseguibile e process injection.

Discovery: il processo ottiene una lista di tutti i processi in esecuzione e ha la capacità di leggere le software policy

Anti-Reverse Engineering: il file eseguibile si presente in formato "Packed", comprimendo i dati contenuti mediante il sofware Packer MPRESS_v2.19. Il PE contiene due sezioni di valore, MPRESS1 e MPRESS2, e la seconda contiene l'Original Entrypoint (EOP) del malware. Entrambi le sezioni presentano elevati livelli di entropia, rispettivamente 7,5 e 5,6.

Privilege Escalation: il processo di sistema si connette alla rete, successivamente ad una iniezione di codice, alloca la memoria in altri processi e sovrascrive dati, inietta file nell'applicazione Windows, crea un thread in un altro processo esistente.

Network Communication: il malware tenta un DNS lookup aprendo una connessione UDPv4 verso l'indirizzo 8.8.8.8, contattando la porta 53.

Process Tree:

- Sample-2020021 (PID 1504)
  - Notepad.exe (PID 3704)
  - Cell_jr.exe (PID 5524)

File Dropped: il malware rilascia il file mirai.dll e 5 copie del file malevolo cell_jr.exe in distinte locazioni del filesystem

Resources Dropped: il malware rilascia 3 risorse, di cui 2 con estensione .jpg e 1 con estensione PUH.

Registry Keys Set: il malware corrompe i valori all'interno delle chiavi di registro:

- \SystemRoot\System32\Config\DEFAULT
- \CONTROL PANEL\DESKTOP\WallPaper
- \SOFTWARE\POLICIES\MICROSOFT\WINDOWS\SAFER\CODEIDENTIFIERS

Screenshot:

Analisi comportamentale: il malware sotto analisi, viene introdotto nel sistema con l'ausilio di un dispositivo USB e, successivamente alla detonazione, alloca memoria e avvia un processo notepad.exe, visibile a video. Viene poi modificata l'immagine del Desktop impostando la risorsa "cell_jr.jpg", presente nella cartella "C: \Users\student\Pictures". Precedentemente all'injection del processo notepad.exe viene effettuato un controllo dei processi in esecuzione, alla ricerca di stringhe specifiche, come ad esempio la denominazioni di soluzioni anti-virus commerciali. In tal caso il processo termina, senza allarmare i sistemi di monitoraggio. Per ottenere la caratteristica di persistenza viene copiato il processo malevolo cell_jr.exe nella cartella di sistema Startup, contenente i processi da avviare ad ogni boot del sistema, e in altre 5 locazioni nel filesystem. Inoltre, il malware è in grado di offuscare file e processi, complicando l'analisi della memoria volatile. Durante l'esecuzione viene eseguita una DNS lookup su canale UDP, avente come destinazione l'indirizzo IP 8.8.8.8 .

Figura 6.30. Modifica dell'immagine Desktop con cell_jr.jpg

6.4 Stuxnet

La fase di **Data Analysis** si è focalizzata nell'estrazione delle evidenze digitali presenti nella memoria RAM del sistema infetto successivamente alla ricezione del malware riconosciuto come Stuxnet. Si riportano di seguito le evidenze digitali rilevate, sulla base della selezione definita nella fase di Data Examination:

- **Informazioni sul sistema infetto**: è stato identificato Windows x32 come sistema operativo della macchina infettata dal malware, con l'ausilio del Volatility plugin info, come raffigurato in figura 6.31. Si riscontra, inoltre la presenza di un processore, la root del sistema operativo configurata sul drive C:\Windows e il time-stamp di creazione della macchina virtuale in "3 Aug 2008 04:31:36". Si nota inoltre come l'indirizzo del Kernel in memoria sia riconosciuto in 0x804d7000.

```
┌──(parallels㉿kali-linux-2021-3)-[~/volatility3]
└─$ python3 vol.py -f /home/parallels/Desktop/stuxnet.vmem windows.info.Info
Volatility 3 Framework 2.4.1
Progress:  100.00               PDB scanning finished
Variable        Value

Kernel Base     0x804d7000
DTB     0x319000
Symbols file:///home/parallels/volatility3/volatility3/symbols/windows/ntkrnlpa.pdb/30B5FB31AE7E4ACAABA750AA241FF331-1.json.xz
Is64Bit False
IsPAE   True
layer_name      0 WindowsIntelPAE
memory_layer    1 FileLayer
KdDebuggerDataBlock     0x80545ae0
NTBuildLab      2600.xpsp.080413-2111
CSDVersion      3
KdVersionBlock  0x80545ab8
Major/Minor     15.2600
MachineType     332
KeNumberProcessors      1
SystemTime      2011-06-03 04:31:36
NtSystemRoot    C:\WINDOWS
NtProductType   NtProductWinNt
NtMajorVersion  5
NtMinorVersion  1
PE MajorOperatingSystemVersion  5
PE MinorOperatingSystemVersion  1
PE Machine      332
PE TimeDateStamp        Sun Apr 13 18:31:06 2008
```

Figura 6.31. Informazioni del sistema infettato da Stuxnet

- **Elenco dei processi in memoria**: è stato identificato, mediante i plugin Volatility pslist, pstree e psscan, il flusso di esecuzione del malware, che si avvia alle 17:08:54 del 29-10-2010 dall'injection sul processo benevolo "winlogon.exe" (PID 624). In successione vengono eseguiti i processi "lsass.exe" (PID 680), benevolo, e "service.exe" (PID 668), malevolo. Quest'ultimo processo esegue, a sua volta, altri 4 processi riconosciuti come malevoli, o sospetti:

"svchost.exe" (PID 1032), "Lsass.exe" (PID 868), "Lsass.exe" (PID 1928), "svchost.exe" (PID 940). L'albero di esecuzione dei processi è raffigurato in figura 6.33. In particolare, "lsass.exe" è il processo Microsoft Local Security Authentication Server (Autenticazione di Sicurezza Locale), responsabile dell'autenticazione e identificazione dell'utente e dell'applicazione delle politiche di sicurezza. Esso è riconosciuto come file Microsoft "attendibile" [155], ma spesso vittima di mascheramento, soprattutto se viene memorizzato in una posizione differente da "C:\Windows\System32", come in questo caso accade per i processi con PID 868 e 1928, ritrovati in "C:\Windows\\System32", come raffigurato in figura 6.34. Il sospetto sui processi malevoli con PID 868 e 1928 viene confermata dal plugin "ldrmodules", che rende possibile rilevare casi di "process injection" analizzando la presenza dei processi in 3 liste concatenate legate alla struttura Process Environment Block (PEB).

```
┌──(parallels㉿kali-linux-2021-3)-[~/volatility3]
└─$ python3 vol.py -f /home/parallels/Desktop/stuxnet.vmem windows.pstree.PsTree
Volatility 3 Framework 2.4.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime

4       0       System  0x823c8830      59      403     N/A     False   N/A     N/A
* 376   4       smss.exe        0x820df020      3       19      N/A     False   2010-10-29 17:08:53.000000      N/A
** 600  376     csrss.exe       0x821a2da0      11      395     0       False   2010-10-29 17:08:54.000000      N/A
** 624  376     winlogon.exe    0x81da5650      19      570     0       False   2010-10-29 17:08:54.000000      N/A
**** 680 624     lsass.exe       0x81e70020      19      342     0       False   2010-10-29 17:08:54.000000      N/A
*** 668 624     services.exe    0x82073020      21      431     0       False   2010-10-29 17:08:54.000000      N/A
***** 1664       668     vmtoolsd.exe    0x81fe52d0      5       284     0       False   2010-10-29 17:09:05.000000      N/A
****** 968       1664    cmd.exe 0x81c0cda0      0       -       0       False   2011-06-03 04:31:35.000000      2011-06-03 04:31:36.000000
******* 304      968     ipconfig.exe    0x81f14938      0       -       0       False   2011-06-03 04:31:35.000000      2011-06-03 04:31:36.000000
***** 1412       668     spoolsv.exe     0x81fee8b0      10      118     0       False   2010-10-29 17:08:56.000000      N/A
***** 868        668     lsass.exe       0x81c498c8      2       23      0       False   2011-06-03 04:26:55.000000      N/A
***** 1032       668     svchost.exe     0x822843e8      61      1169    0       False   2010-10-29 17:08:55.000000      N/A
****** 2040      1032    wscntfy.exe     0x820ecc10      1       28      0       False   2010-10-29 17:11:49.000000      N/A
****** 976       1032    wuauclt.exe     0x822b9a10      3       133     0       False   2010-10-29 17:12:03.000000      N/A
***** 1928       668     lsass.exe       0x81c47c00      4       65      0       False   2011-06-03 04:26:55.000000      N/A
***** 940        668     svchost.exe     0x81e61da0      13      312     0       False   2010-10-29 17:08:55.000000      N/A
***** 844        668     vmacthlp.exe    0x823315d8      1       25      0       False   2010-10-29 17:08:55.000000      N/A
***** 1580       668     jqs.exe 0x81e0eda0      5       148     0       False   2010-10-29 17:09:05.000000      N/A
***** 1200       668     svchost.exe     0x81ff7020      14      197     0       False   2010-10-29 17:08:55.000000      N/A
***** 1816       668     VMUpgradeHelper 0x821a0568      3       96      0       False   2010-10-29 17:09:08.000000      N/A
***** 756        668     imapi.exe       0x82279998      4       116     0       False   2010-10-29 17:11:54.000000      N/A
***** 856        668     svchost.exe     0x81db8da0      17      193     0       False   2010-10-29 17:08:55.000000      N/A
****** 1872      856     wmiprvse.exe    0x81fa5390      5       134     0       False   2011-06-03 04:25:58.000000      N/A
***** 1080       668     svchost.exe     0x81e18b28      5       80      0       False   2010-10-29 17:08:55.000000      N/A
***** 188        668     alg.exe 0x8205ada0      6       107     0       False   2010-10-29 17:09:09.000000      N/A
1196    1728    explorer.exe    0x820ec7e8      16      582     0       False   2010-10-29 17:11:49.000000      N/A
* 324   1196    TSVNCache.exe   0x81e86978      7       54      0       False   2010-10-29 17:11:49.000000      N/A
* 1356  1196    VMwareUser.exe  0x81e6b660      9       251     0       False   2010-10-29 17:11:50.000000      N/A
* 1712  1196    jusched.exe     0x8210d478      1       26      0       False   2010-10-29 17:11:50.000000      N/A
* 660   1196    Procmon.exe     0x81c543a0      13      189     0       False   2011-06-03 04:25:56.000000      N/A
* 1912  1196    VMwareTray.exe  0x81fc5da0      1       50      0       False   2010-10-29 17:11:50.000000      N/A
```

Figura 6.32. Flusso di esecuzione del malware Stuxnet

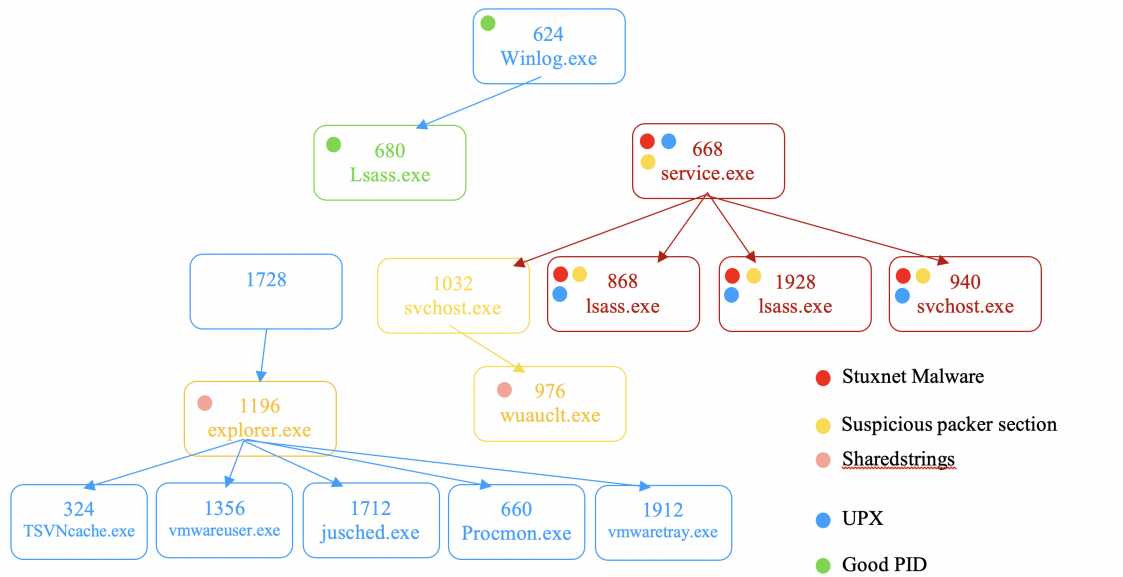


Figura 6.33. Distinzione processi benevoli e malevoli successivi alla detonazione di Stuxnet

- **Elenco dei comandi eseguiti da un processo**: il flusso di esecuzione riscontrato nel punto precedente è stato confermato eseguendo il plugin cmdline, il cui output è disponibile

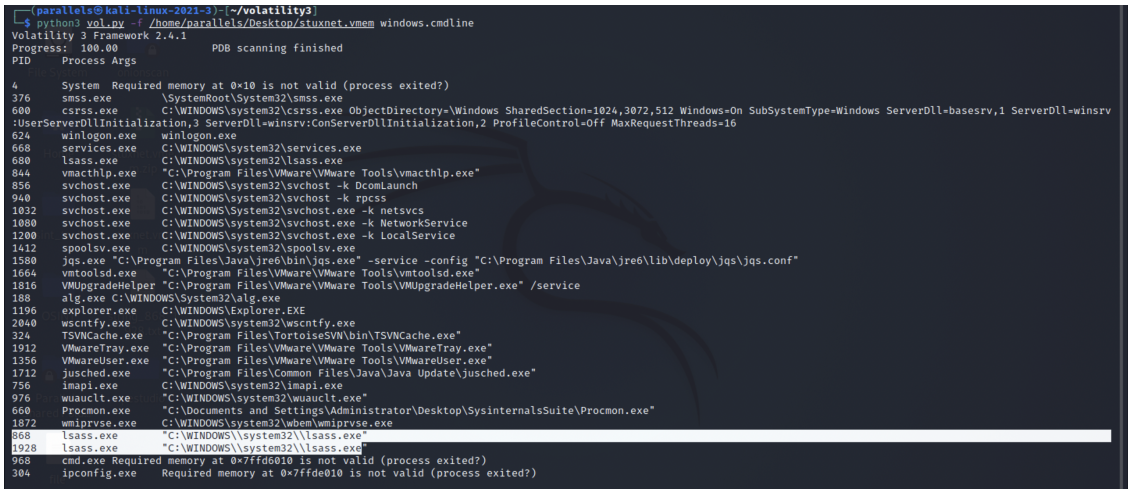


Figura 6.34. Tecnica di camuffamento dei processi con PID 868 e 1928

in figura 6.34. In esso si riscontra il passaggio dei processi con PID 624, 668, 680, 1032, 868, 1928, 940 al processo cmd.exe. L'output ritornato mostra il path di ogni processo da cui è possibile identificare come malevoli i processi con PID 868 e 1928, riscontrati nel direttorio "C:\Windows\\System32". In particolare, analizzando la entry relativa al processo "svchost.exe" (PID 940) si nota l'esecuzione della dll "rpcss", come si può notare in figura 6.35. Questo processo avvia un Server RPC (Remote Procedure Call), volto ad ascoltare le connessioni in entrata dal altri computer infetti sulla stessa rete locale [156]. Dalla fonte citata si comprende come, usando un GUID fisso, gli Stuxnet peer siano in grado di "identificarsi, comunicare e aggiornarsi a vicenda, consentendogli di propagare gli aggiornamenti anche se non possono raggiungere il server di comando e controllo esterno a causa di un firewall o della mancanza di connettività Internet".

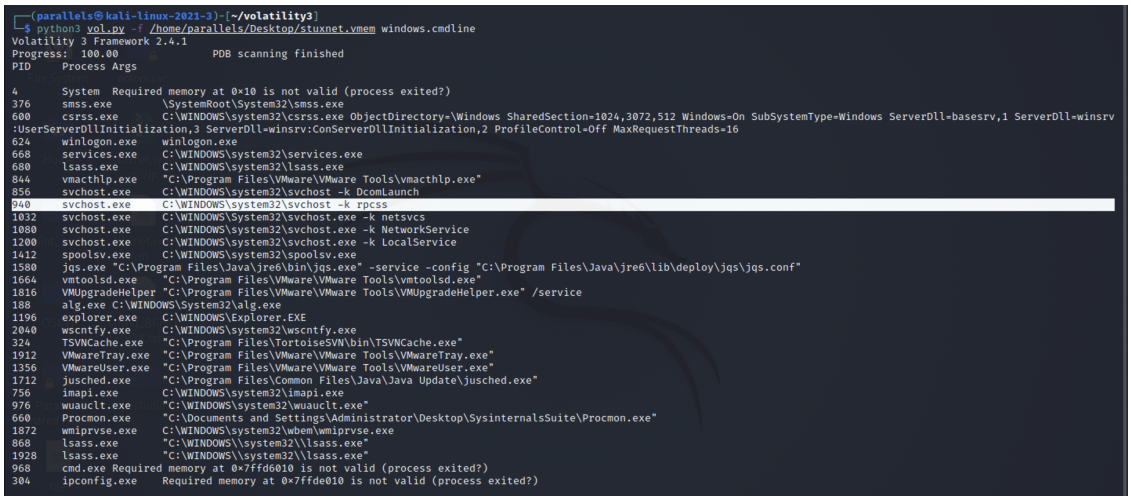


Figura 6.35. Evidenza di esecuzione della Remote Procedure Call (RPC)

- **Elenco delle connessioni di rete**: sono state identificate, mediante il plugin sockets, diverse connessioni di rete instaurate dai processi sospetti, in momenti diversi successivi alla detonazione del malware, come raffigurato in figura 6.36. Il processo 940 instaura al momento della detonazione del malware una connessione TCP alla porta 135 con l'indirizzo 0.0.0.0. Il processo con PID 680, successivamente, apre due connessioni UDP alle porte 500 e 4500 anch'esse con indirizzo di destinazione 0.0.0.0. Quest'ultimo è identificato come un indirizzo "non-routable" usato in ipv4 per designare target invalidi o sconosciuti. In

particolare questa potrebbe essere un'evidenza di tecniche di offuscamento da parte del malware in analisi. A distanza di tempo, il processo 1032 apre una connessione di rete alla porta 123 verso l'indirizzo localhost. Quest'ultimo, essendo datato 2011-06-03, rappresenta un'evidenza della caratteristica di persistenza del malware in analisi. L'analisi OSINT ha evidenziato successivamente come il malware identificato, nell'analisi, come Stuxnet tenti di contattare un server web remoto per testare la connettività di rete utilizzando url lecite come msn.com o windows.update.com. A tal riguardo il plugin mftscan ha individuato un file archiviato sospetto denominato "administrator@exp.www.msn[1].txt".

```
┌──(parallels㉿kali-linux-2021-3)-[~/volatility]
└─$ python2 vol.py -f /home/parallels/Desktop/stuxnet.vmem python3 vol.py -f /home/parallels/Desktop
Volatility Foundation Volatility Framework 2.6.1
0x81dc2008      680    500     17 UDP             0.0.0.0         2010-10-29 17:09:05 UTC+0000
0x82294aa8      940    135      6 TCP             0.0.0.0         2010-10-29 17:08:55 UTC+0000
0x81da4d18      680      0    255 Reserved        0.0.0.0         2010-10-29 17:09:05 UTC+0000
0x81fdbe98     1032    123     17 UDP             127.0.0.1       2011-06-03 04:25:47 UTC+0000
0x82060008      680   4500     17 UDP             0.0.0.0         2010-10-29 17:09:05 UTC+0000
```

Figura 6.36. Dump delle connessioni di rete

- **Elenco delle stringhe note come "malevole"**: sono stati identificati diversi modelli di modelli di malware, eseguendo il Volatility plugin vadyarascan. In particolare, i moduli di regole yara su cui si è trovato riscontro sono i seguenti:
  - **suspicious_packer_section**: regola per identificare l'utilizzo generico di un Packer, al fine di offuscamento del codice. In particolare si riscontrano i valori $s49 = "UPX0", $s50 = "UPX1", $s63 = "UPX!".
  - **UPX**: regola per identificare l'utilizzo del Packer UPX, al fine di offuscamento del codice. In particolare si riscontrano i valori $a = "UPX0", $b = "UPX1", $c = "UPX!". Si identificano quindi due sezioni all'interno della versione packed del file eseguibile. UPX! viene posto comunemente al termine del PE Header [149].
  - **StuxNet_Malware_1**: regola per rilevare la presenza di modelli interni trovati nei sample relativi a Stuxnet. Si riscontrano i valori $op1, $op2 e $op3, ognuno dei quali si riferisce a specifiche sequenze di istruzioni attuate dal malware.
  - **StuxNet_Malware_3**: regola per rilevare la presenza di modelli interni trovati nei sample relativi a Stuxnet. Si riscontrano i valori specifici di stringhe che rimandano al malware Stuxnet:
    - $x1 = "SHELL32.DLL.ASLR."
    - $s1 = "W̃TR4141.tmp"
    - $s2 = "W̃TR4132.tmp"
    - $s3 = "totalcmd.exe
    - $s4 = "wincmd.exe"
    - $s5 = "http://www.realtek.com0"
    - $s6 = "%08x-%08x-%08x-%08x"
  - **SharedStrings**: regola per rilevare la presenza di modelli interni trovati nei sample relativi a LURK0/CCTV0. Si riscontra il valore $m4, che tuttavia non viene descritto in [147] e probabilmente è un False Positive.
- **Elenco delle chiavi di registro**: sono state identificate, eseguendo i Volatility plugin registry.printkey e hivelist e filtrando il tempo di ultima modifica in modo da essere successivo alla data di detonazione del malware, corruzioni interne a diverse chiavi di registro:
  - \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY: al momento della detonazione del malware sono stati caricati in memoria gli indirizzi di alcune sottochiavi della chiave "SECURITY", ovvero "POLICY", "RXACT" e "SAM". Analizzando le singole chiavi per "data di ultima modifica" si rileva come siano state

effettivamente modificati i valori delle sotto-chiavi "RXACT" e "SAM". In particolare nella chiave "SAM" è stato sovrascritto il valore impostando un "symbolik link" verso "\Registry\Machine\SAM\SAM". Eseguendo il plugin filescan è stato individuato un artefatto del file "REGISTRY_MACHINE_SAM", recuperato con il plugin dumpfiles. L'analisi di quest'ultimo non ha prodotti risultati evidenti, per via della crittatura del file .dat ritornato dal plugin. L'estrazione delle stringhe con il comando "Strings" ha evidenziato in ogni caso parametri di configurazione di account admin, probabilmente al fine di privilege escalation.

- \Device\HarddiskVolume1\WINDOWS\system32\config\SYSTEM: al momento della detonazione del malware sono stati caricati in memoria gli indirizzi di alcune sottochiavi della chiave "SYSTEM", ovvero "ControlSet001", "ControlSet002", "LastKnownGoodRecovery", "MountedDevices", "Select", "Setup", "WPA" e "CurrentControlSet". Analizzando le singole chiavi per "data di ultima modifica" si rileva come siano state effettivamente modificati i valori delle sotto-chiavi "ControlSet001\Services", "ControlSet001\Control", "CurrentControlSet"

  La chiave "CurrentControlSet" identifica l'attuale ControlSet utilizzato e memorizza le informazioni su ogni servizio registrato nel sistema. Essa viene settata al momento della detonazione a "ControlSet001". Parallelamente in "Controlset001\Hardware Profiles\current" viene fissato come profilo hardware corrente il "0001". In "ControlSet001\Services" si rileva l'aggiunta dei driver "MRXNET" e "MRXCRLS" registrati nel gruppo "Network" e collegati agli omonimi file .sys nel direttorio, come mostrano le figure 6.37 e 6.38
  "C:\WINDOWS\system32\Drivers\". Essi sono due rootkit a livello di driver hardware che rimandano direttamente al malware Stuxnet [156].
  In "Controlset001\Control\computername\activecomputername" viene fissato il valore di "JAN-DF663B3DBF1" dal malware Stuxnet come analizzato in [157]. In "ControlSet001\Services\Kbdclass" viene impostata la "Keyboard class" impostando il link al driver "system32\DRIVERS\kbdclass.sys"

- NONAME: al momento della detonazione del malware sono stati caricati in memoria gli indirizzi di alcune sottochiavi della chiave "[NONAME]", ovvero "MACHINE", "USER", "ACPI", "DESCRIPTION", "DEVICEMAP", "RESOURCEMAP". La denominazione "NONAME" è associata a chiavi sconosciute o nascoste e può essere dunque un'evidenza di offuscamento da parte del malware Stuxnet. Si sospetta, quindi, l'impostazione di configurazioni personalizzate mediante collegamenti alle sottochiavi sopra citate. La chiave "MACHINE", con data di ultima modifica coincidente con il timestamp di detonazione del malware, non presenta valori. La chiave "USER\S-1-5-18" viene collegata alla configurazione "\Registry\User\.Default". Viene iniettato del codice nella registry key personalizzata "ACPI\dsdt\ptltd_\custom__\06040000". Vengono modificate e aggiunte sottochiavi della chiave "DESCRIPTION\system\". In "DEVICEMAP\KeyboardClass" viene forzata la "KeyboardClass0" e la "KeyboardClass1" al medesimo valore, impostando il link alla chiave
  "\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\Kbdclass", settato come descritto a "system32\DRIVERS\kbdclass.sys". In "DEVICEMAP\PointerClass" vengono forzate le 4 classi di puntatore mouse al driver impostato in
  "ControlSet001\Services\Mouclass" e disponibile in
  "WINDOWS\system32\drivers\mouclass.sys". In "RESOURCEMAP" vengono modificati i valori di componenti hardware basso-livello.

- **Elenco dei file caricati in memoria**: non sono stati identificati file rilasciati sul filesystem e caricati in memoria dal malware Stuxnet, dall'esecuzione dei plugin "mftscan" e "filescan". Questo conferma la caratteristica stealth del malware in analisi, ovvero l'effettiva caratteristica di rimanere nascosto all'interno del sistema senza lasciare tracce sul filesystem.

- **Elenco delle entry nella SSDT**: l'analisi del dump delle SSDT caricate in memoria, output del Volatility plugin "ssdt", non ha riportato nessun caso di SSDT Hooking. Ogni voce della tabella contiene indirizzi che puntano all'interno del modulo NT. Se il modulo fosse stato diverso si sarebbe parlato di "ssdt Hooking".

```
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\system
Key name: MRxNet (S)
Last updated: 2011-06-03 04:26:47 UTC+0000

Subkeys:
  (V) Enum

Values:
REG_SZ        Description     : (S) MRXNET
REG_SZ        DisplayName     : (S) MRXNET
REG_DWORD     ErrorControl    : (S) 0
REG_SZ        Group           : (S) Network
REG_SZ        ImagePath       : (S) \??\C:\WINDOWS\system32\Drivers\mrxnet.sys
REG_DWORD     Start           : (S) 1
REG_DWORD     Type            : (S) 1
```

Figura 6.37. Dump della chiave di registro MRxNet

- **Elenco dei mutant**: l'analisi dei mutant in memoria, rilasciati dal plugin "mutantscan" non ha evidenziato entry direttamente collegate al malware in analisi. L'analisi OSINT riporta, al contrario, l'utilizzo di mutex globali.
- **Elenco dei kernel driver**: sono stati identificati, mediante l'esecuzione del Volatility plugin modscan, i seguenti kernel driver legati al malware in analisi, come mostra la figura 6.39:
  - **mrxnet.sys**: è un rootkit iniettato da Stuxnet nel sistema che viene utilizzato per occultare i file dannosi sul computer infettato e per sostituire i file di Stuxnet sul disco se vengono rimossi [157]. Si rileva, analizzando le Strings, come il driver sia firmato in modo legittimo da Realtek e quindi in grado di operare senza essere bloccato dai controlli anti-virus. Dall'analisi OSINT su [157] si comprende come il rootkit sia in grado di intercettare le richieste IRP, D richieste di scrittura/lettura sul filesystem ntfs, sulla fat o su cd-rom. Dall'analisi dell'output ritornato dal plugin "callbacks" si nota come il driver mrxnet.sys sia registrato ad una routine di callback per la registrazione del filesystem, denominata "IoRegisterFsRegistrationChange", in modo da agganciare gli oggetti del filesystem appena creati.
  - **mrxcls.sys**: è un rootkit iniettato da Stuxnet nel sistema che funge da "Load Point" [157], con l'obiettivo di iniettare ed eseguire copie di Stuxnet in processi specifici. Anch'esso ha un certificato firmato da Realtek. In particolare, il rootkit consente l'esecuzione di Stuxnet a ogni avvio di un sistema infetto e funge quindi da punto di caricamento principale per la minaccia. A tal riguardo è stata creata un'apposita chiave di registro che permette il caricamento ad ogni avvio di Windws. Dall'analisi OSINT su [157] si è riscontrato come il rootkit decritti e legga il valore binario "Data" presente nella citata chiave di registro, che contiene un elenco di coppie (nome del processo di destinazione, modulo da iniettare). In questo modo si comprende come il malware si replichi in processi specifici (Services.exe, s7tgtopx.exe, CCProjectMgr.exe) con il file oem7a.pnf, riconosciuto come una copia della principale dll di Stuxnet. Il file è stato ritrovato in memoria all'indirizzo "\Device\HarddiskVolume1\WINDOWS\inf\", eseguendo il plugin "filescan".

  Quando questi file vengono creati, l'ora del file viene modificata in modo che corrisponda a quella di altri file nella directory di sistema per evitare sospetti. Una volta che questi file sono stati eliminati, Stuxnet crea le voci di registro necessarie per caricare questi file come servizi che verranno eseguiti automaticamente all'avvio di Windows, come rilevato nella sezione relativa alle chiavi di registro.
- **Elenco delle "Sections"**: è stata riscontrata la presenza di due sezioni di valore a composizione del PE file di Stuxnet, mediante il tool da riga di comando "Strings" eseguito sul

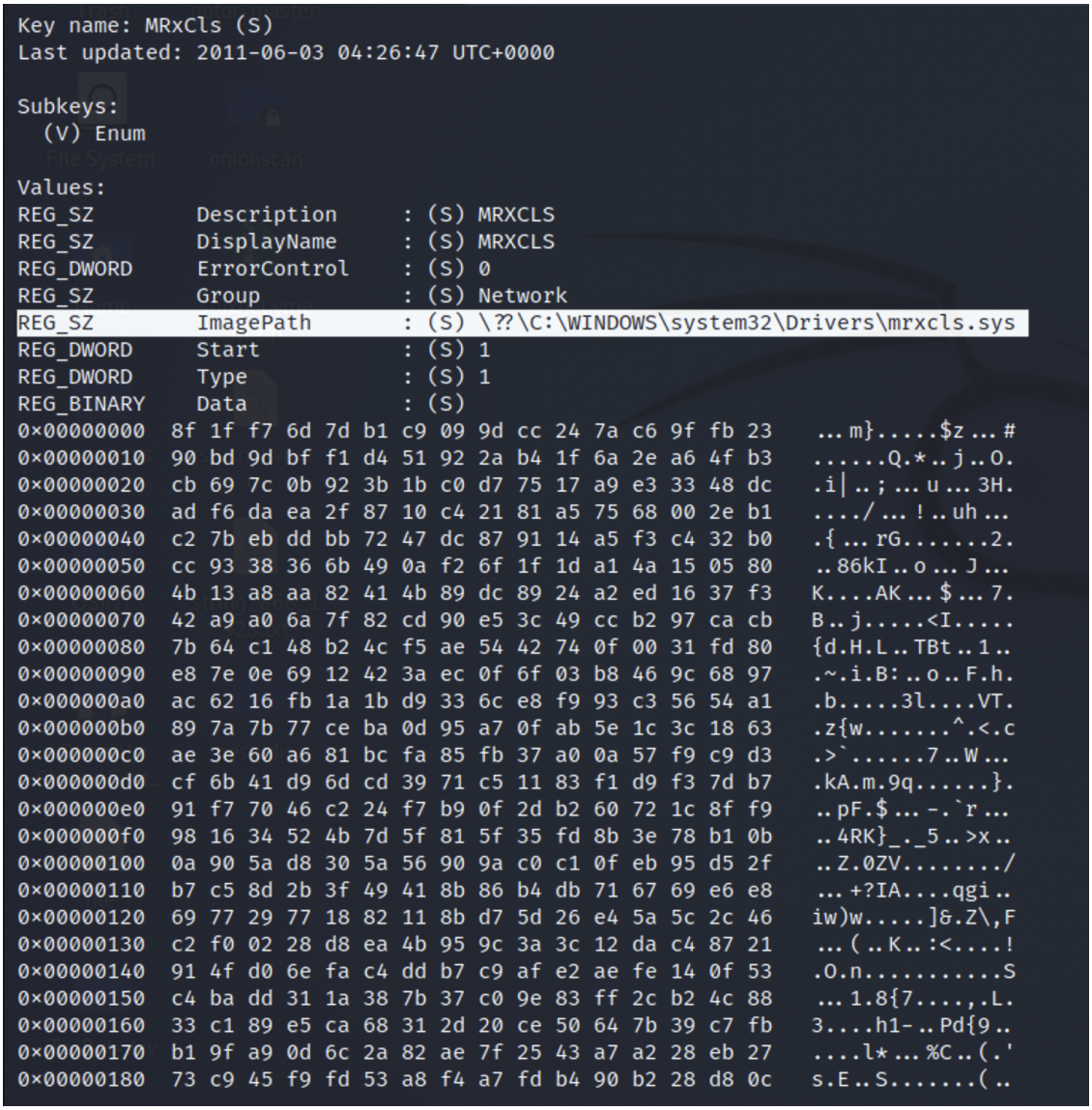


Figura 6.38. Dump della chiave di registro MRxCls

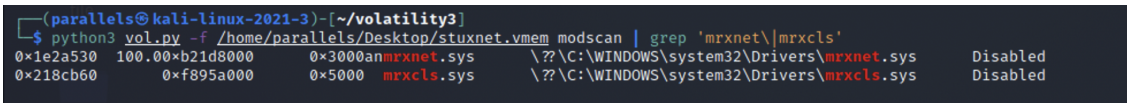


Figura 6.39. Focus sui driver installati da Stuxnet sul sistema

dump dei processi con PID 868 e 1928. In particolare le sezioni, denominate "UPX0" e "UPX1", sono compresse dal Packer UPX. Il dump effettuato sui processi malevoli è stato passato come input all'applicativo di analisi statica PeStudio e, a conferma dei risultati, all'applicativo di OSINT Hybrid-Analysis [158] [159], al fine di recuperarne i valori di entropia e confermare l'utilizzo di un Packer sul codice sorgente. L'entropia riscontrata sulla sezione .text relativa ad entrambi i dump dei processo 868 e 1928 è pari a 6.23. Nell'analisi statica si individua, inoltre, l'indirizzo dell'entrypoint a 0x10014bd nella sezione ".verif".

- **Elenco degli "Imports"**: sono state riscontrate, eseguendo il plugin dlllist, diverse dll importate dai processi relativi al malware Stuxnet:
  - PID 868: il processo malevolo carica 8 DLL utili per le istruzioni che dovrà eseguire

sul sistema, come visibile in figura 6.40. L'analisi OSINT mediante Hybrid-Analysis sul dump del processo con PID 868 [158] ha rilevato come esso tenti il dll hooking, iniettando del codice al relativo indirizzo virtuale delle seguenti: ntdll.dll, kernelbase.dll, gdi32.dll, win32u.dll, kernel32.dll, user32.dll. Di particolare interesse sono la advapi32.dll, che fornisce l'accesso ai componenti avanzati del core di Windows, come il gestore dei servizi e il registro di sistema; la rpcrt4.dll, che è necessaria alla trasmissione e ricezione di Remote Procedure Call (RPC) utilizzate dal malware per collegarsi ai relativi peer.

```
lsass.exe pid:    868
Command line : "C:\WINDOWS\\system32\\lsass.exe"
Service Pack 3

Base             Size  LoadCount LoadTime                       Path
---------- ---------- ---------- ------------------------------ ----
0x01000000     0x6000     0xffff                                C:\WINDOWS\system32\lsass.exe
0x7c900000    0xaf000     0xffff                                C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff                                C:\WINDOWS\system32\kernel32.dll
0x77dd0000    0x9b000     0xffff                                C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000    0x92000     0xffff                                C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000    0x11000     0xffff                                C:\WINDOWS\system32\Secur32.dll
0x7e410000    0x91000     0xffff                                C:\WINDOWS\system32\USER32.dll
0x77f10000    0x49000     0xffff                                C:\WINDOWS\system32\GDI32.dll
```

Figura 6.40. DLL caricate dal processo con PID 868

– PID 1928: il processo malevolo carica 28 DLL utili per le istruzioni che dovrà eseguire sul sistema, come visibile in figura 6.41. L'analisi OSINT mediante Hybrid-Analysis sul dump del processo con PID 1928 [159] ha rilevato come esso tenti il dll hooking, iniettando del codice al relativo indirizzo virtuale delle seguenti: ntdll.dll, kernelbase.dll, gdi32.dll, win32u.dll, kernel32.dll, user32.dll. Anche il PID 1928 carica le dll advapi32.dll e rpcrt4.dll. Di particolare interesse è la kernel32.dll.aslr.0360b7ab, responsabile dell'Address Space Layout Randomization delle sezioni del codice eseguibile, per il fatto che essa viene nascosta da Stuxnet. Infatti è visualizzabile eseguendo il plugin dlllist, ma non compare nell'output del plugin ldrmodules filtrando per il PID 1928 all'offset 0x00870000, come mostra la figura 6.42.

```
lsass.exe pid:   1928
Command line : "C:\WINDOWS\\system32\\lsass.exe"
Service Pack 3

Base             Size  LoadCount LoadTime                       Path
---------- ---------- ---------- ------------------------------ ----
0x01000000     0x6000     0xffff                                C:\WINDOWS\system32\lsass.exe
0x7c900000    0xaf000     0xffff                                C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff                                C:\WINDOWS\system32\kernel32.dll
0x77dd0000    0x9b000     0xffff                                C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000    0x92000     0xffff                                C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000    0x11000     0xffff                                C:\WINDOWS\system32\Secur32.dll
0x7e410000    0x91000     0xffff                                C:\WINDOWS\system32\USER32.dll
0x77f10000    0x49000     0xffff                                C:\WINDOWS\system32\GDI32.dll
0x00870000   0x138000        0x1                                C:\WINDOWS\system32\KERNEL32.DLL.ASLR.0360b7ab
0x76f20000    0x27000        0x2                                C:\WINDOWS\system32\DNSAPI.dll
0x77c10000    0x58000       0x27                                C:\WINDOWS\system32\msvcrt.dll
0x71ab0000    0x17000        0xa                                C:\WINDOWS\system32\WS2_32.dll
0x71aa0000     0x8000        0x8                                C:\WINDOWS\system32\WS2HELP.dll
0x76d60000    0x19000        0x2                                C:\WINDOWS\system32\IPHLPAPI.DLL
0x5b860000    0x55000        0x2                                C:\WINDOWS\system32\NETAPI32.dll
0x774e0000   0x13d000        0x5                                C:\WINDOWS\system32\ole32.dll
0x77120000    0x8b000        0x4                                C:\WINDOWS\system32\OLEAUT32.dll
0x76bf0000     0xb000        0x2                                C:\WINDOWS\system32\PSAPI.DLL
0x7c9c0000   0x817000        0x2                                C:\WINDOWS\system32\SHELL32.dll
0x77f60000    0x76000        0x8                                C:\WINDOWS\system32\SHLWAPI.dll
0x769c0000    0xb4000        0x2                                C:\WINDOWS\system32\USERENV.dll
0x77c00000     0x8000        0x2                                C:\WINDOWS\system32\VERSION.dll
0x771b0000    0xaa000        0x2                                C:\WINDOWS\system32\WININET.dll
0x77a80000    0x95000        0x2                                C:\WINDOWS\system32\CRYPT32.dll
0x77b20000    0x12000        0x2                                C:\WINDOWS\system32\MSASN1.dll
0x71ad0000     0x9000        0x2                                C:\WINDOWS\system32\WSOCK32.dll
0x773d0000   0x103000        0x2                                C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_
35d4ce83\comctl32.dll
0x5d090000    0x9a000        0x1                                C:\WINDOWS\system32\comctl32.dll
```

Figura 6.41. DLL caricate dal processo con PID 1928

– PID 940: il processo carica 45 DLL utili per le istruzioni che dovrà eseguire sul sistema. Si rilevano tra queste le 28 caricate dal processo con PID 1928, evidenza del fatto che Stuxnet sia caricato su più processi. A differenza dal processo con PID 1928 [159],

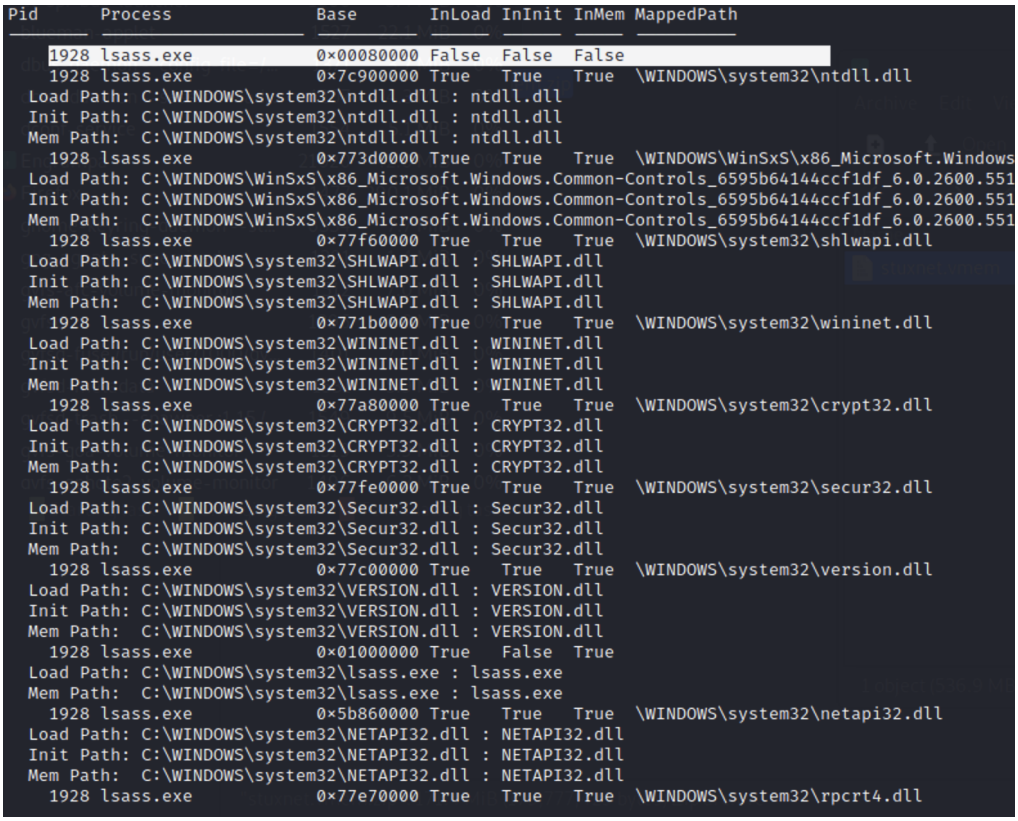


Figura 6.42. Evidenza di hiding della dll kernel32.dll.aslr.0360b7ab

Hybrid-Analysis non etichetta il processo 940 come malevolo, evidenziando però la possibilità di essere utilizzato a fini di privilege escalation [160].

In particolare, dall'analisi delle Strings viene rilevata la "s7otbxsx.dll". Dall'analisi di [157] si comprende come la stringa "s7otbxsx.dll" venga ricercata sul sistema infetto per individuare la presenza del software Siemens SIMATIC Step7. Una volta trovato, Stuxnet inietta del codice in esso al fine di aggiungere una dozzina di funzioni critiche "relative all'accesso, lettura, scrittura e cancellazione di blocchi del codice sul PLC". Se il file non viene trovato Stuxnet non esegue ulteriori istruzioni malevole sull'host del computer.

- **Elenco delle "Strings"**: si identificano 43593 stringhe estratte dai dump dei processi malevoli con PID 868 e 1928. Si citano nel particolare le seguenti:
  - **Hooking**: SetWindowsHookExW, UnhookWindowsHookEx, CallNextHookEx
  - **RPC**: RpcEpUnregister, RpcServerUseProtseqEpW, RpcBindingFree, RpcMgmtEpEltInqDone, RpcStringFreeW, RpcMgmtEpEltInqNextW, RpcMgmtEpEltInqBegin, RpcServerRegisterIf, RpcServerInqBindings, RpcServerListen, RpcEpRegisterW, RpcMgmtStopServerListening, RpcServerRegisterIf2, RpcStringBindingParseW, RpcServerUnregisterIf, RpcBindingToStringBindingW
  - **Certificati TLS**: VeriSign, Thawte, Microsoft Corporation, Realtek Semiconductor
  - **Privilege Escalation**: GetTokenInformation, OpenThreadToken, LookupPrivilegeValueW, ImpersonateLoggedOnUser, AdjustTokenPrivileges, EqualSid, InitializeSecurityDescriptor, SetSecurityDescriptorDacl, AllocateAndInitializeSid, ConvertStringSecurityDescriptorToSecurityDescriptorW, GetSecurityDescriptorSacl, SetSecurityDescriptorSacl
  - **Resource**: FindResourceExW, LoadResource, SizeofResource, LockResource

- **Elenco delle "Resources"**: dall'analisi della memoria volatile non sono state riscontrate particolari risorse in riferimento al malware Stuxnet e rilasciate sul filesystem. Il Symantec Report [161] cita la presenza di 15 risorse con Resource ID differente, ognuna volta ad una specifica funzione.

L'analisi OSINT sul malware in esamina è stata avviata in seguito alla generazione di 13 file .dmp dei processi sospetti, output del plugin malfind. Per ogni file è stato, successivamente, generato il relativo digest con il comando "sha256sum" e passato in input agli applicativi di OSINT Virus Total e Hybrid-Analysis:

- abce3e79e26b5116fe7f3d40d21eaa4c8563e43366086f9ec07c2925593f69dc: non è stato trovato nessun match sospetto
- 2b2945f7cc7cf5b30ccdf37e2adbb236594208e409133bcd56f57f7c009ffe6d: il digest si riferisce al processo lsass.exe (PID 1928), classificato come malevolo da 57/71 soluzioni antivirus. Il processo viene costruito con tecniche di anti-reverse engineering, presentando sezioni pacchettizzate mediante il Packer UPX. In particolare, viene rilevata un elevato livello di entropia, pari a 7.99, nella sezione UPX1 di questo processo. Il report sul processo in esamina evidenzia 6 indicatori mappati su 4 tecniche di attacco.
- 163b7da37df4ae6dafbfb5bf88b319dabf7846cee73d4192c6a7593e835857a8: il processo viene rilevato come sicuro
- 10f07b9fbbc6a8c6dc4abf7a3d31a01e478accd115b33ec94fe885cb296a3586: il digest si riferisce al processo lsass.exe, classificato come malevolo da 44/70 soluzioni antivirus. Il processo possiede un certificato X509 emesso da Verisign e assegnato ad Realtek Semiconductor.
- e97d61f7393ac5838a1800f3e9aa22c6205f4d7e2bde494573d35c57bc967819: non è stato trovato nessun match sospetto
- a4b4b29f0df45283b629203b080c09ddb5bc6eb4cd8e9b725f75121a8b7e728e: il digest si riferisce al processo lsass.exe, classificato come malevolo da 47/71 soluzioni antivirus.
- 46046bdfc8e1ef280d613bb20fabe0be721f16c0d9ba16e94c09ba396e91c822: non è stato trovato nessun match sospetto
- 5bef6c7d4f197c61ee99ddbda0e30e0027646c1bfedc9e673ecbe680912ffa98: il processo viene rilevato come sicuro
- 649b0be48e2daf7551f074efe6d9e0d55a907a2fbdd8a461a48fbf65f4f1ef98: il digest si riferisce al processo scvhost.exe, classificato come malevolo da 37/66 soluzioni antivirus. Il processo possiede un certificato X509 emesso da Verisign e assegnato ad Realtek Semiconductor. Viene trovato un riferimento a una stringa di query WMI, nota per essere utilizzata per il rilevamento delle macchine virtuali. Il report sul processo in esamina evidenzia 13 indicatori mappati su 10 tecniche di attacco.
- 1e7d0ad9b03f260550f34be9345452342ec9058065c2b5b845bb5bf9f60e443a: il processo viene rilevato come sicuro
- 6f97d5b608dbc8a3eedc12582f4bafc1692a24d140442def50483aeba2bca485: il processo viene rilevato come sicuro
- 5d803593ef2c22b7f86d5e0c06039a161cf25b6fa112bea05f191006ff95c0ec: il digest si riferisce al processo services.exe, classificato come malevolo da 42/69 soluzioni antivirus. Il processo possiede un certificato X509 emesso da Verisign e assegnato ad Realtek Semiconductor. Viene trovato un riferimento a una stringa di query WMI, nota per essere utilizzata per il rilevamento delle macchine virtuali. Il report sul processo in esamina evidenzia 3 indicatori mappati su 2 tecniche di attacco.
- f6813ac88ad15c684a6ff753480b2e5ddc12d0932b11e5a4fbeelaaa0cd8aa0b: non è stato trovato nessun match sospetto

Sono stati poi effettuati i dump degli eseguibili dei processi malevoli con PID 868 [158], 1928 [159] e 940 [160]. I relativi file sono stati passati in input alla soluzione OSINT Hybrid-Analysis, al fine di produrre i relativi report di analisi malware. Le evidenze riscontrate in questi sono state descritte nelle sezioni di "Data Analysis" a conferma degli artefatti riscontrati durante l'analisi della memoria volatile.

Data Reporting: in questa fase viene mostrato un "summary" delle caratteristiche del malware, rilevate durante l'analisi post-mortem.

Malware name: Stuxnet

Malware family: Trojan.Win32.Stuxnet, APT, Dropper

Threat score: 100% malevolo [158] [159]

SHA256:

- PID 868: 6f293f095e960461d897b688bf582a0c9a3890935a7d443a929ef587ed911760
- PID 1928:20a3c5f02b6b79bcac9adaef7ee138763054bbedc298fb2710b5adaf9b74a47d

MD5:

- PID 868: 7b62da1a65ffc31c55da778b276ad1e2
- PID 1928: e1e00c2d5815e4129d8ac503f6fac095

Size:

- PID 868: 9 Kb
- PID 1928: 9 Kb

- PID 868: 7b62da1a65ffc31c55da778b276ad1e2
- PID 1928: e1e00c2d5815e4129d8ac503f6fac095

File type: WIN32 executable

Creation Time: 2010-01-13 10:00:53 UTC

Sistema Operativo: Windows (x32)

Memory Dump: stuxnet.vmem (512 MB)

Detonation time: 29-10-2010 17:08:54

Entrypoint: 0x10014bd

Anti-Virus Results:

- PID 868: 57/72 security vendors hanno indicato questo file come malevolo [158]
- PID 1928: 52/67 security vendors hanno indicato questo file come malevolo [159]

MITRE ATT&CK Techniques Detection: 7 tecniche di attacco, sfruttate in 5 tattiche [158]

MITRE ATT&CK ID: T1134, T1548, T1027.002, T1056.004, T1082, T1083, T1012

Installation/Execution: viene verificato che la minaccia sia in esecuzione su una versione compatibile di Windows, controllato se il computer è già infetto o meno, elevato il privilegio del processo corrente a sistema, controllato quali prodotti antivirus sono installati. Quindi viene iniettato il file .dll in processi specifici.

Persistence: il malware per ottenere persistenza sovrascrive il valore di diverse chiavi di registro. In particolare vengono create le chiavi di registro "MRXNET" e "MRXCRLS" in "ControlSet001\Services", legate simbolicamente a rootkit iniettati dal malware nel sistema.

Evasive: il malware presenta abilità di offuscamento di file,informazioni e chiavi di registro. Inoltre, introduce tecniche di process injection ed evasione delle soluzioni antivirus, principalmente camuffandosi come processo benigno, utilizzando certificati tls validi

Discovery: il processo ottiene una lista di tutti i processi in esecuzione, ha la capacità di leggere le software policy e gli antivirus attivi sul sistema. Inoltre, il malware ricerca il software Siemens SIMATIC Step7 per procedere con le proprie operazioni.

Anti-Reverse Engineering: il file eseguibile si presente in formato "Packed", comprimendo i dati contenuti mediante il sofware Packer UPX. Il PE contiene due sezioni di valore, UPX0 e UPX1. Le sezioni presentano un elevato valore di entropia pari a 6,23.

Privilege Escalation: il processo di sistema si connette alla rete, successivamente ad una iniezione di codice, alloca la memoria in altri processi e sovrascrive dati, inietta file nell'applicazione Windows e tenta la privilege escalation a sistema.

Network Communication: il malware testa la connessione di rete sul dominio benigno www.msn.it, per poi avviare procedure RPC (Remote Procedure Call) verso gli altri peer infetti finalizzate ad instaurare una comunicazione C&C. Vengono rilevate, inoltre, esecuzioni di DNS lookup da parte del malware in analisi.

Process Tree:

- winlog.exe (PID 624)
  - service.exe (PID 668)
    - lsass.exe (PID 868)
    - lsass.exe (PID 1928)
    - svchost.exe (PID 940)

File Dropped: il malware rilascia il file oem7a.pnf contenente la principale dll di Stuxnet

Resources Dropped: il malware non rilascia alcuna risorsa nel filesystem che possa essere individuata mediante metodi di Memory-Forensics

Registry Keys Set: il malware corrompe i valori all'interno delle chiavi di registro:

- \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY

- \Device\HarddiskVolume1\WINDOWS\system32\config\SYSTEM
- \NONAME

Screenshot: N/A

Analisi comportamentale: il malware in esamina, identificato come Stuxnet, infetta un sistema Windows x32 andando ad iniettare una dll malevola su processi specifici. Una delle prime azioni effettuate riguarda la ricerca del software Siemens SIMATIC Step7 al fine di iniettare del codice in esso e modificare le funzionalità dell'applicativo, in modo da poter controllare i PLC dei sistemi SCADA associati. Se assente, Stuxnet non esegue ulteriori istruzioni malevole sull'host del sistema infetto. Il malware utilizza tecniche di privilege escalation per ottenere i privilegi di sistema. Stuxnet, per quanto riguarda la connessione di rete, testa la connettività inviando una richiesta DNS a indirizzi benigni, per poi instaurare comunicazioni con gli altri peer infetti mediante il protocollo RPC (Remote Procedure Call). Per ottenere la caratteristica di persistenza il malware sovrascrive i valori di diverse chiavi di registro e inietta due rootkit, sfruttati per il Load Point e per sostituire i file di Stuxnet sul disco se vengono rimossi. I rootkit possiedono entrambi certificati X509 emessi da Verisign e assegnati a Realtek Semiconductor Corp, che permettono l'evasione dai sistemi di monitoraggio antivirus. Inoltre, il malware è in grado di offuscare file e processi, complicando l'analisi della memoria volatile

6.5 Emotet

La fase di **Data Analysis** si è focalizzata nell'estrazione delle evidenze digitali presenti nel sistema, e nel particolare nella memoria RAM del sistema infetto, successivamente alla detonazione del malware riconosciuto come Emotet. Si riportano di seguito le evidenze digitali rilevate, sulla base della selezione definita nella fase di Data Examination:

- **Informazioni sul sistema infetto**: e stato identificato Windows 10 x64 come sistema operativo della macchina infettata dal malware, con l'ausilio del Volatility plugin info. Si riscontra, come rappresentato in Figura 6.43 la presenza di due processori, la root del sistema operativo configurata sul drive C:\Windows e il time-stamp di creazione della macchina virtuale in "6 Jul 2018 6:57:56", precedente alla data di inizio analisi del malware, in quanto snapshot di una macchina virtuale contenente diversi tool di malware analysis. Si nota inoltre come l'indirizzo del Kernel in memoria sia riconosciuto in 0xf80086e10000.

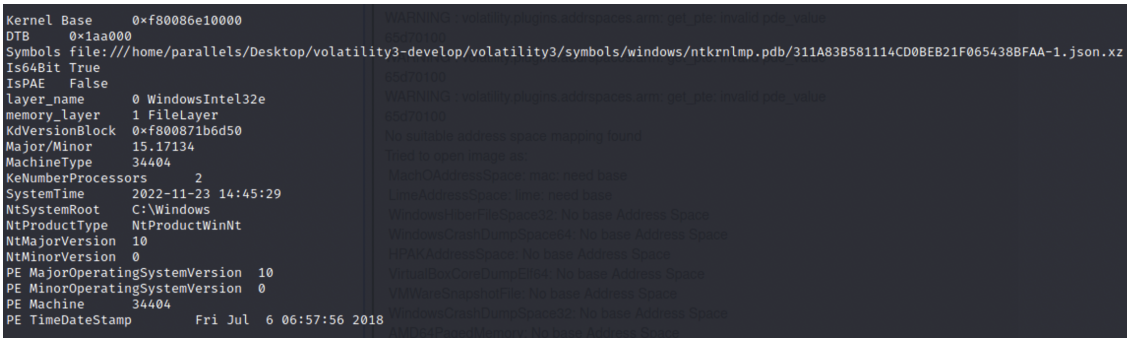
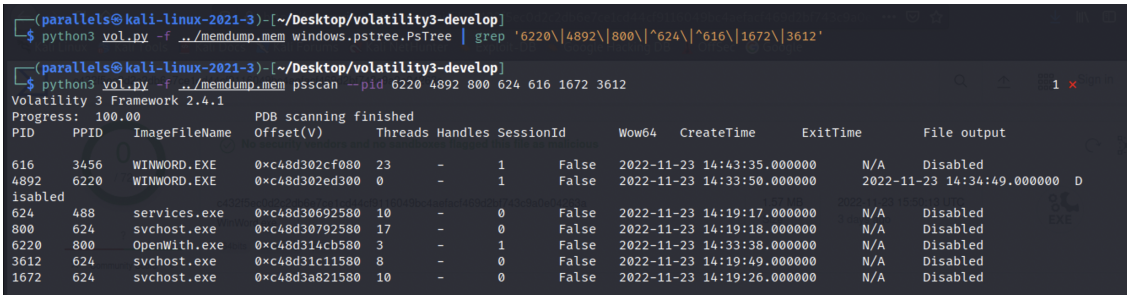


Figura 6.43. Informazioni del sistema infettato da Emotet

- **Elenco dei processi in memoria**: l'esecuzione dei plugin pslist, psscan e pstree ha evidenziato diversi sospetti nell'elenco dei processi in memoria. In primo luogo, vengono ritornati da pslist e psscan 131 processi, mentre il plugin pstree (rappresentazione ad albero dell'output di pslist) ne riporta solo 38. Un dettaglio sui relativi output è riportato in Figura **??**.

Tra questi si identifica il processo WINWORD.EXE (PID 4892) come vettore del malware in esamina, con data di creazione fissata a "2022-11-23 14:33:50" e data di termine "2022-11-23 14:34:49". Il processo sospetto viene generato da OpenWith.exe (PID 6220), a conferma della modalità di apertura del file malevolo con estensione ".docx". In particolare, si nota come il processo con PID 6220 venga generato da svchost.exe (PID 800), figlio di service.exe (PID 624). Si nota come service.exe sia padre di 67 processi, dei quali si annotano come sospetti i svchost.exe con PID 7084, creato 5 secondi successivamente alla data di detonazione del malware, e PID 1672, responsabile di diverse connessioni sospette registrate da Fakenet in seguito alla detonazione. Dall'analisi delle connessioni di rete, mediante il plugin netstat, si annota come sospetto il processo svchost.exe (PID 3612). Eseguendo il plugin Malfind si nota la presenza di tracce di code injection nel processo WINWORD.EXE (PID 616), come si può notare nella Figura 6.45. Analizzando nel profondo, si nota come esso a differenza dell'omonimo con PID 4892 non abbia un tempo di terminazione. Questo rappresenta l'evidenza che il malware si sia replicato nel processo 616, con data di creazione "2022-11-23 14:43:35" per ottenere la caratteristica di persistenza. Il processo 616, a differenza del processo 4892, presenta 23 thread, ulteriore evidenza di attività sospette, confermate dalla lista di handle relativi al processo 616 ritornata dal plugin handles. L'esecuzione di Process Explorer durante la fase di detonazione del malware ha rilevato processi nel flusso di esecuzione, avviato da WINWORD.EXE, che non risultano in memoria: cmd.exe, powershell.exe e conhost.exe. Il particolare è rappresentato in Figura 6.46. Il flusso viene, inoltre, confermato dai risultati dell'analisi OSINT su Virus Total [162] e Hybrid-Analysis [163], ma non trova conferma sull'output del plugin psscan. L'esecuzione del plugin UserAssist evidenzia, però, la presenza degli eseguibili powershell.exe e cmd.exe nella chiave chiave di registro User, come raffigurato in Figura 6.47. I timestamp di ultima modifica sono successivi alla data di detonazione del malware e l'indirizzo hive è offuscato.



```
┌──(parallels㉿kali-linux-2021-3)-[~/Desktop/volatility3-develop]
└─$ python3 vol.py -f ../memdump.mem windows.pstree.PsTree | grep '6220\|4892\|800\|^624\|^616\|1672\|3612'

┌──(parallels㉿kali-linux-2021-3)-[~/Desktop/volatility3-develop]
└─$ python3 vol.py -f ../memdump.mem psscan --pid 6220 4892 800 624 616 1672 3612                    1 ×
Volatility 3 Framework 2.4.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime        File output

616     3456    WINWORD.EXE     0×c48d302cf080  23      -       1       False   2022-11-23 14:43:35.000000      N/A     Disabled
4892    6220    WINWORD.EXE     0×c48d302ed300  0       -       1       False   2022-11-23 14:33:50.000000      2022-11-23 14:34:49.000000  D
isabled
624     488     services.exe    0×c48d30692580  10      -       0       False   2022-11-23 14:19:17.000000      N/A     Disabled
800     624     svchost.exe     0×c48d30792580  17      -       0       False   2022-11-23 14:19:18.000000      N/A     Disabled
6220    800     OpenWith.exe    0×c48d314cb580  3       -       1       False   2022-11-23 14:33:38.000000      N/A     Disabled
3612    624     svchost.exe     0×c48d31c11580  8       -       0       False   2022-11-23 14:19:49.000000      N/A     Disabled
1672    624     svchost.exe     0×c48d3a821580  10      -       0       False   2022-11-23 14:19:26.000000      N/A     Disabled
```

Figura 6.44. Risultati dei plugin Psscan e Pstree filtrati sui processi sospetti

- **Elenco dei comandi eseguiti da un processo**: il flusso di esecuzione riscontrato nel punto precedente è stato confermato eseguendo il plugin cmdline. In esso, come rappresentato in Figura 6.48, si riscontra cronologicamente il passaggio dei processi con PID 624, 800, 1672, 3612, 6220, 4892, 7084, 616. In particolare, il PID 800 avvia automaticamente al boot la procedura DcomLaunch, mentre il PID 1672 procede con la Dnscache. Il processo con PID 3612 avvia la procedura SSDPSRV (SSDP Discovery), che risulta sospetto in quanto in [164] si descrive come la procedura non venga avviata automaticamente, bensì dall'utente. Il protocollo rende possibile ottenere dispositivi con accesso internet (modem, router, etc) [165]. Successivamente alla detonazione di WINWORD.EXE, il processo svchost.exe con PID 7084 avvia la procedura di diagnostica del sistema WdiSystemHost. Viene, inoltre, rilevata la locazione del file .docx malevolo in
"C:\Users\student\Downloads\1f\
efb6b3a11d1d7f55bb0696453e31a0d6602f03ae74324b502277c7a245f6e5ef".

- **Elenco delle connessioni di rete**: l'analisi delle connessioni di rete è avvenuta esaminando l'output del plugin netstat, eseguito sul dump di memoria, e correlando gli artefatti con timestamp successivo alla data di detonazione con i pacchetti di rete registrati da Fakenet e Wireshark in fase di esecuzione del malware. I risultati di netstat, opportunamente filtrati,

```
616     WINWORD.EXE     0×7ffb5c220000  0×7ffb5c22ffff  VadS    PAGE_EXECUTE_READWRITE  16      1       Disabled
64 74 72 52 00 00 00 00 dtrR....
00 00 00 00 00 00 00 00 ........
60 2d 22 5c fb 7f 00 00 `-"\....
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
0×7ffb5c220000: je      0×7ffb5c220075
0×7ffb5c220003: push    rdx
0×7ffb5c220004: add     byte ptr [rax], al
0×7ffb5c220006: add     byte ptr [rax], al
0×7ffb5c220008: add     byte ptr [rax], al
0×7ffb5c22000a: add     byte ptr [rax], al
0×7ffb5c22000c: add     byte ptr [rax], al
0×7ffb5c22000e: add     byte ptr [rax], al
616     WINWORD.EXE     0×7ffb5bb60000  0×7ffb5bb6ffff  VadS    PAGE_EXECUTE_READWRITE  16      1       Disabled
64 74 72 52 00 00 00 00 dtrR....
00 00 00 00 00 00 00 00 ........
e0 01 b6 5b fb 7f 00 00 ...[....
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
0×7ffb5bb60000: je      0×7ffb5bb60075
0×7ffb5bb60003: push    rdx
0×7ffb5bb60004: add     byte ptr [rax], al
0×7ffb5bb60006: add     byte ptr [rax], al
0×7ffb5bb60008: add     byte ptr [rax], al
0×7ffb5bb6000a: add     byte ptr [rax], al
0×7ffb5bb6000c: add     byte ptr [rax], al
0×7ffb5bb6000e: add     byte ptr [rax], al
0×7ffb5bb60010: loopne  0×7ffb5bb60013
0×7ffb5bb60012: mov     dh, 0×5b
0×7ffb5bb60014: sti
0×7ffb5bb60015: jg      0×7ffb5bb60017
```

Figura 6.45. Porzione di output del plugin Malfind

sono disponibili in Figura 6.49 A tal riguardo sono stati coinvolti nell'analisi i processi svchost.exe (PID 1672) e svchost.exe (PID 3612). Per quanto riguarda il processo identificato dal PID 1672, il plugin netstat rileva 6 UDP endpoint. Filtrando il file .pcap in cui Wireshark ha registrato le connessioni di rete precedentemente e successivamente alla detonazione del malware, si nota come il PID 1672 trasmetta diverse richieste DNS, sfruttando i protocolli MDNS e LLMNR. I pacchetti vengono trasmessi dalle porte 5353 e 5355, utilizzando il protocollo UDP allo strato trasporto. L'analisi delle richieste DNS viene effettuata su Fakenet, dove si ottengono le richieste per i seguenti domini, da parte del processo con PID 1672:

- www.msftconnecttest.com
- self.events.data.microsoft.com
- activation-v2.sls.microsoft.com
- v10.events.data.microsoft.com
- gize24.com
- kanmoretail.com
- www.harboursidechurch.org
- imbiss-catering.com
- www.jjackmedia.com
- watson.telemetry.microsoft.com
- www.virustotal.com
- fonts.gstatic.com
- isvkzkgp.local
- jlullzhs.local
- ouugadaurxfexf.local
- wpad.local
- clients2.google.com

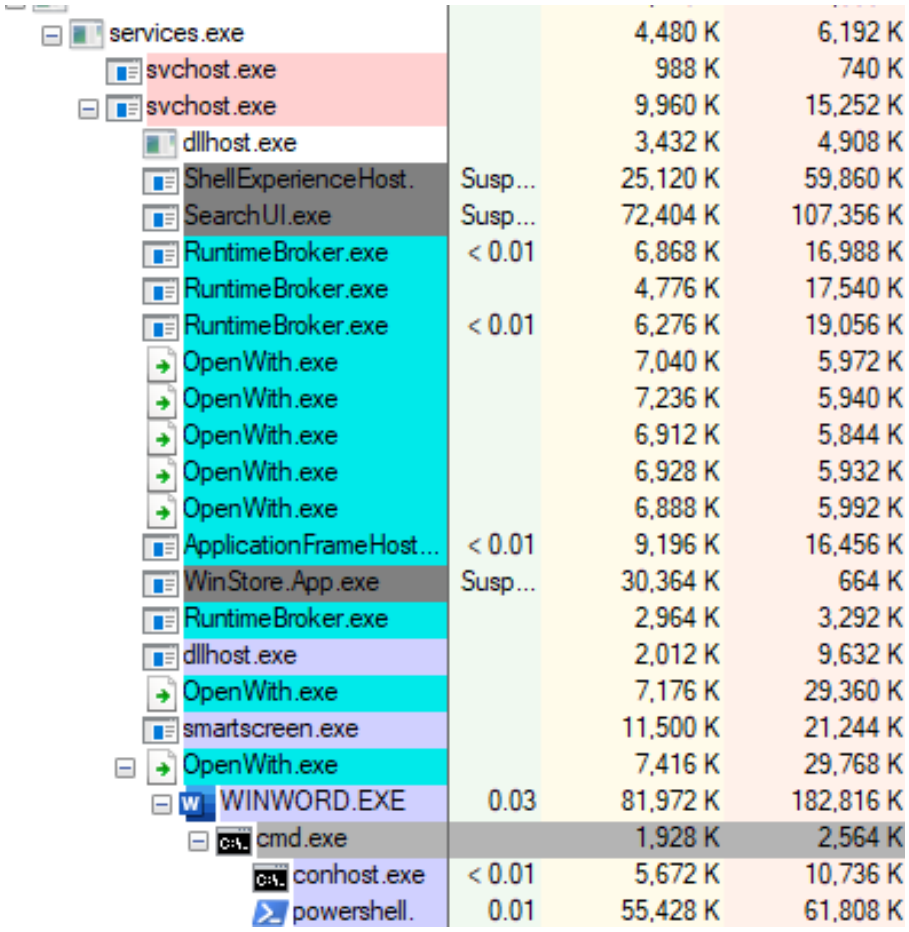


Figura 6.46. Flusso di esecuzione catturato da Process Explorer

```
┌──(parallels㉿kali-linux-2021-3)-[~/Desktop/volatility3-develop]
└─$ python3 vol.py -f ../memdump.mem userassist  | grep -i 'cmd.exe'
* 0x9e047a3200000       \??\C:\Users\student\ntuser.dat ntuser.dat\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF
lue     %windir%\system32\cmd.exe       N/A     0       7       0:00:40.906000  N/A
* 0x9e047a320000        \??\C:\Users\student\ntuser.dat ntuser.dat\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF
lue     %windir%\system32\cmd.exe       N/A     9       15      0:07:51.112000  2019-09-30 09:40:19.000000

┌──(parallels㉿kali-linux-2021-3)-[~/Desktop/volatility3-develop]
└─$ python3 vol.py -f ../memdump.mem userassist  | grep -i 'powershell.exe'
* 0x9e047a3200000       \??\C:\Users\student\ntuser.dat ntuser.dat\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF
lue     %windir%\system32\WindowsPowerShell\v1.0\powershell.exe N/A     3       5       0:05:30.733000  2018-08-29 07:32:48.000000
* 0x9e047a320000        \??\C:\Users\student\ntuser.dat ntuser.dat\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF
lue     %windir%\system32\WindowsPowerShell\v1.0\powershell.exe N/A     5       12      0:05:47.987000  2018-10-04 13:37:18.000000
```

Figura 6.47. Evidenza dei processi powershell.exe e cmd.exe nella chiave di registro User

– _googlecast._tcp.local

La maggior parte dei domini indicati viene riconosciuta, mediante ricerche di OSINT, nelle stringhe del malware Emotet e molti presentano delle vulnerabilità che sono state sfruttate dal malware nel corso della sua divulgazione. Eseguendo il dump del processo 1672 e analizzandone le Strings, si nota la presenza della dll "RPCRT4.dll", che rende possibile la "chiamata di procedura remota", sfruttata per le comunicazioni C&C.

Per quanto riguarda il processo identificato dal PID 3612, il plugin netstat rileva 8 UDP endpoint, Filtrando il file .pcap in cui Wireshark ha registrato le connessioni di rete precedentemente e successivamente alla detonazione del malware, si nota come il PID 3612 trasmetta diverse richieste SSDP, al fine di identificare quali host sono attivi sulla rete. Queste richieste vengono etichettate come sospette, in quanto potrebbero essere correlate alla modalità in cui avviene la diffusione del malware nei sistemi presenti nella

```
┌──(parallels㉿kali-linux-2021-3)-[~/Desktop/volatility3-develop]
└─$ python3 vol.py -f ../memdump.mem cmdline | grep '6220\|4892\|800\|^624\|^616\|1672\|3612\|7084'

624     services.exe    C:\Windows\system32\services.exe
800     svchost.exe     C:\Windows\system32\svchost.exe -k DcomLaunch -p
1672    svchost.exe     c:\windows\system32\svchost.exe -k networkservice -p -s Dnscache
3612    svchost.exe     c:\windows\system32\svchost.exe -k localserviceandnoimpersonation -p -s SSDPSRV
6220    OpenWith.exe    C:\Windows\system32\OpenWith.exe -Embedding
4892    WINWORD.EXE     Required memory at 0x8a99e12020 is not valid (process exited?)
7084    svchost.exe     c:\windows\system32\svchost.exe -k localsystemnetworkrestricted -p -s WdiSystemHo
616     WINWORD.EXE     "C:\Program Files\Microsoft Office\root\Office16\Winword.exe" /n "C:\Users\studen
5bb0696453e31a0d6602f03ae74324b502277c7a245f6e5ef
```

Figura 6.48. Dettaglio sui processi sospetti passati a cmd.exe

stessa rete locale. I pacchetti vengono trasmessi dalle porte 1900, 49166, 49167, 49168 e 49169, utilizzando il protocollo UDP allo strato trasporto.

Offset	Proto	LocalAddr	LocalPort	ForeignAddr	ForeignPort	State	PID	Owner	Created
C48D327EBC50	TCPv4	192.168.56.101	139	0.0.0.0	0	LISTENING	4	System	2022-11-23 14:34:43.000000
C48D32940370	UDPv4	192.168.56.101	137	*	0		4	System	2022-11-23 14:34:43.000000
C48D32940660	UDPv4	192.168.56.101	138	*	0		4	System	2022-11-23 14:34:43.000000
C48D32CEB470	UDPv4	0.0.0.0	5353	*	0		1672	svchost.exe	2022-11-23 14:34:43.000000
C48D32CEB470	UDPv6	::	5353	*	0		1672	svchost.exe	2022-11-23 14:34:43.000000
C48D327E68E0	UDPv4	0.0.0.0	5353	*	0		1672	svchost.exe	2022-11-23 14:34:43.000000
C48D327CF1A0	UDPv4	0.0.0.0	5355	*	0		1672	svchost.exe	2022-11-23 14:34:43.000000
C48D327CF1A0	UDPv6	::	5355	*	0		1672	svchost.exe	2022-11-23 14:34:43.000000
C48D326B2830	UDPv4	0.0.0.0	5355	*	0		1672	svchost.exe	2022-11-23 14:34:43.000000
C48D2F4EEEB0	UDPv6	fe80::315c:6f83:4a84:49eb	1900	*	0		3612	svchost.exe	2022-11-23 14:34:44.000000
C48D327EBAF0	UDPv6	::1	1900	*	0		3612	svchost.exe	2022-11-23 14:34:44.000000
C48D3123C010	UDPv4	192.168.56.101	1900	*	0		3612	svchost.exe	2022-11-23 14:34:44.000000
C48D2FE98420	UDPv4	127.0.0.1	1900	*	0		3612	svchost.exe	2022-11-23 14:34:44.000000
C48D32CD6BB0	UDPv6	fe80::315c:6f83:4a84:49eb	49166	*	0		3612	svchost.exe	2022-11-23 14:34:44.000000
C48D3058F500	UDPv6	::1	49167	*	0		3612	svchost.exe	2022-11-23 14:34:44.000000
C48D31EE6520	UDPv4	192.168.56.101	49168	*	0		3612	svchost.exe	2022-11-23 14:34:44.000000
C48D306D3980	UDPv4	127.0.0.1	49169	*	0		3612	svchost.exe	2022-11-23 14:34:44.000000

Figura 6.49. Output del plugin Netstat filtrato per "Created Time" successivo al timestamp di detonazione

- **Elenco delle chiavi di registro**: l'esecuzione dei plugin hivelist e printkey non ha rilevato nessuna chiave di registro modificata in seguito alla detonazione del malware Emotet. Questo si è rilevato molto sospetto in fase di verifica OSINT, dove viene inserito in input l'hash del dump del file sospetto in diversi strumenti di OSINT. A tal riguardo, Virus Total e Intezer nel particolare, hanno riscontrato l'aggiunta, la modifica e la cancellazione di diverse chiavi di registro non confermata nel dump. L'analisi è stata quindi ripetuta più volte, testando la produzione del memory dump in modalità diverse, anche attendendo del tempo tra la detonazione e la cattura dell'immagine di memoria per assicurarsi l'esecuzione di istruzioni anche in presenza di specifici timer. Non avendo ricevuto riscontri positivi, si ipotizza la possibilità che il malware rimanga persistente in memoria fino alla ricezione di un comando C&C o che gli agganciamenti alle chiavi di registro vengano offuscati o, ancora, che il malware non completi l'infezione del sistema per problemi legati all'ambiente di detonazione. Questa percezione diventa più probabile andando ad analizzare i risultati del plugin handles, filtrati sul processo con PID 616. L'output evidenzia quanto il processo WINWORD.EXE con PID 616 sia legato ad operazioni su 184 chiavi di registro, che risultato nascoste nell'output dei moduli hivelist e printkey. È stato tentato di ricavare il valore delle chiavi di registro

ritornate dal plugin handles andando ad eseguire il plugin printkey e filtrando sulla specifica chiave. Anche in questo caso non sono state riscontrate particolari evidenze.

- **Elenco delle stringhe note come "malevole"**: sono stati identificati diversi modelli di modelli di malware, eseguendo il Volatility plugin vadyarascan. In particolare, i moduli di regole yara su cui si è trovato riscontro sono i seguenti:
  - **Ponmocup**: regola per rilevare la presenza di modelli interni trovati nei sample relativi alla botnet Ponmocup [166]
  - **with_sqlite**: regola per rilevare la presenza di dati SQLite nella raw image [145]
  - **SharedStrings**: regola per rilevare la presenza di modelli interni trovati nei sample relativi a LURK0/CCTV0. Si riscontra il valore $m4, che tuttavia non viene descritto in [147] e probabilmente è un False Positive.

  Come si può notare, non sono state rilevate regole yara conformi a modelli del malware "Emotet". Questo aumenta il sospetto dello sfruttamento di tecniche evasive da parte del malware in presenza di ambiente virtualizzato.

- **Elenco dei file caricati in memoria**: l'analisi degli output dei plugin mftscan e filescan non ha rilevato la presenza di particolari file correlati al malware, ad esclusione del file .docx e .exe di detonazione e delle relative dll caricate e descritte nella sezione "Imports". Si rilevano come sospetti, per data di creazione successiva al timestamp di detonazione, i file: "RUNDLL32.EXE-BBC6AFBE.pf", "DLLHOST.EXE-F2DCEF0D.pf", "SVCHOST.EXE-E45D8788.pf", "efb6b3a11d1d7f55bb0696453e31a0d6602f03ae74324b502277c7a245f6e5ef.LNK".

- **Elenco delle entry nella SSDT**: l'analisi del dump delle SSDT caricate in memoria, output del Volatility plugin "ssdt", non ha riportato nessun caso di SSDT Hooking. Ogni voce della tabella contiene indirizzi che puntano all'interno del modulo NT. Se il modulo fosse stato diverso si sarebbe parlato di "ssdt Hooking".

- **Elenco dei kernel driver**: sono stati identificati, mediante l'esecuzione del Volatility plugin modscan, diversi kernel driver, tra cui si ritengono di interesse:
  - **USBSTOR**: driver della porta di archiviazione USB, viene utilizzato quando l'unità USB è collegata e questo file inizia a comunicare con essa
  - **USBXHCI**: richiede l'accesso diretto alle finestre interne
  - **Volsnap.sys**: driver di Windows che consente al computer di comunicare con l'hardware o i driver collegati
  - **Volume.sys**: è utilizzato per la comunicazione con l'hardware
  - **win32k.sys**: driver di Windows che presenta una vulnerabilità critica, offrendo all'aggressore la possibilità di violare le sandbox di sicurezza che i browser utilizzano per evitare che del codice "untrusted" possa interagire con parti sensibili del Sistema Operativo [167].

  Questi artefatti introducono al pensiero che una chiavetta USB sia stata introdotta precedentemente all'esecuzione del malware e possa essere stata utilizzata come vettore di attacco del malware. Si conferma, infatti, l'utilizzo di una chiavetta USB come vettore del malware.

- **Elenco delle "Sections"**: non è stata rilevata nessuna informazione specifica per descrivere le sezioni del malware in esamina, dall'esecuzione di PeStudio, Detect It Easy e gli applicativi di OSINT Virus Total, Hybrid-Analysis, OPSWAT Metadefender Cloud e Intezer. La motivazione risiede nel fatto che in input non si sta passando un file eseguibile (PE) ma un "doc" file. Unicamente, PeStudio rileva un valore di entropia pari a 5.757 sul documento .docx in ingresso, la cui evidenza è raffigurata in Figura 6.50. Passando in input il dump del processo WINWORD.EXE (PID 616) si riscontrano 8 sezioni, con il valore maggiore di entropia pari a 3.47 (.rdata). Si ipotizza quindi che non venga utilizzato alcun Packer nell'eseguibile WINWORD.EXE. In esso, si riscontra l'entrypoint nella sezione .text all'indirizzo 0x7ff637bb1630.

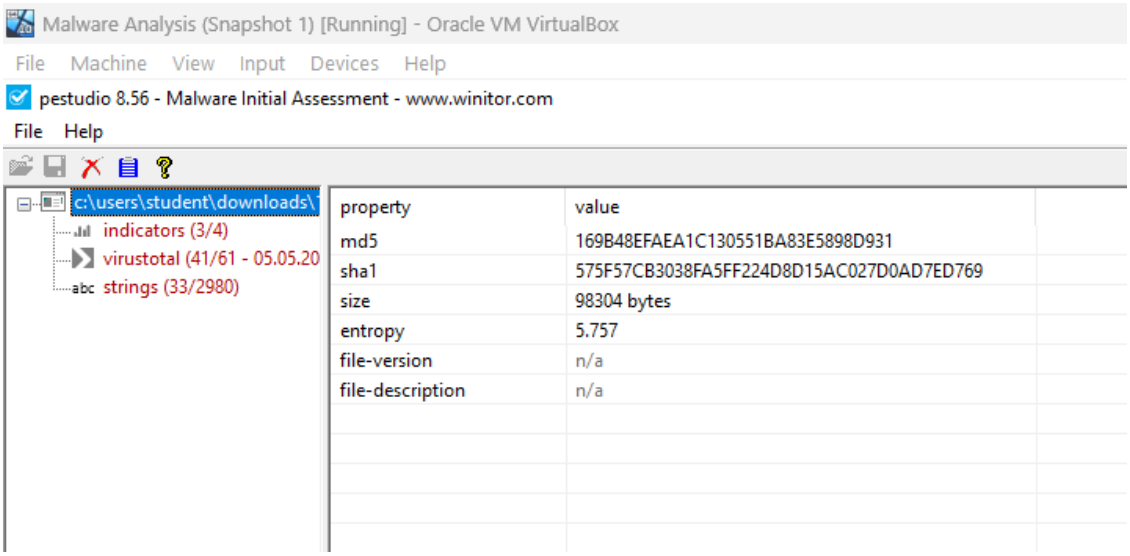


Figura 6.50. Informazioni di analisi statica sul malware Emotet ottenute da PeStudio

- **Elenco degli "Imports"**: sono rilevate mediante i plugin ldrmodules e dlllist diverse dll importate dal processo WINWORD.EXE con PID 616, tra cui molte lecite di Windows. Il conteggio delle entry rileva una differenza tra le 184 dll ottenute mediante ldrmodules e le 165 rilevate da dlllist, mostrando quindi 19 casi di dll-hooking visualizzabili in Figura 6.51. Tra le dll caricato dal malware è stato rilevato come esso tenti il dll hooking, iniettando del codice al relativo indirizzo virtuale delle seguenti: ntdll.dll, kernelbase.dll, gdi32.dll, win32u.dll, kernel32.dll, user32.dll. Di particolare interesse sono la advapi32.dll, che fornisce l'accesso ai componenti avanzati del core di Windows, come il gestore dei servizi e il registro di sistema; la rpcrt4.dll, che è necessaria alla trasmissione e ricezione di Remote Procedure Call (RPC) utilizzate dal malware per collegarsi ai relativi peer. Si nota il processo sechost.dll, non indispensabile per il sistema e posto in una locazione insolita. Dalle stringhe del processo 616 si ottengono le dll principali del malware emotet, mostrate in Figura 6.53: mso20win32client.dll, mso30win32client.dll, mso40uiwin32client.dll, mso50win32client.dll, mso98win32client.dll, mso.dll, VCRUNTIME140.dll, MSVCP140.dll, AppVIsvSubsystems64.dll.

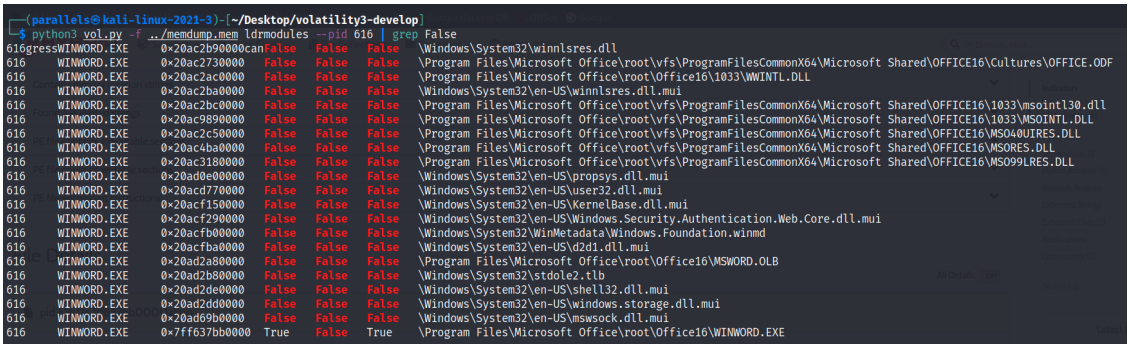


Figura 6.51. Artefatti di DLL-hooking relativi al processo con PID 616

- **Elenco delle "Strings"**: si identificano 33 stringhe in formato human-readable e "blacklisted" passando il file .docx in ingresso a PeStudio. Eseguendo il comando "Strings" sul dump del processo 616 si rilevano le dll principali su cui opera il malware e i metodi utilizzati per ottenere informazioni sul sistema infetto, come "GetStartupInfoW", "GetSystemInfo", "GetSystemDirectoryW", "GetCurrentProcessId". In particolare si riscontra la presenza

```
┌──(parallels㉿kali-linux-2021-3)-[~/Desktop/volatility3-develop]
└─$ strings pid.616.0×7ff637bb0000.dmp | grep dll
mso20win32client.dll
mso30win32client.dll
mso40uiwin32client.dll
mso50win32client.dll
mso98win32client.dll
mso.dll
Mso20Win32Client.dll
api-ms-win-crt-locale-l1-1-0.dll
api-ms-win-crt-math-l1-1-0.dll
api-ms-win-crt-string-l1-1-0.dll
api-ms-win-crt-runtime-l1-1-0.dll
api-ms-win-crt-heap-l1-1-0.dll
api-ms-win-crt-time-l1-1-0.dll
api-ms-win-crt-stdio-l1-1-0.dll
api-ms-win-crt-utility-l1-1-0.dll
MSVCP140.dll
VCRUNTIME140.dll
VCRUNTIME140_1.dll
KERNEL32.dll
ADVAPI32.dll
AppVIsvSubsystems64.dll
```

Figura 6.52. DLL principali del malware Emotet riscontrate nelle Strings del processo con PID 616

di metodi di creazione e rilascio di mutex, legati a istruzioni per l'ottenimento di parametri temporali come "GetSystemTimeAsFileTime". Questi artefatti inducono al pensiero che il malware sia in grado di impostare lo stato "dormiente" sul sistema per ottenere la caratteristica di persistenza o per evadere al controllo di soluzioni anti-virus in caso di particolari situazioni. Il sospetto viene confermato analizzando le stringhe del plugin handles sul processo 616 alla ricerca di oggetti Mutant ad esso collegati, osservabili in Figura 6.53.

- **Elenco delle "Resources"**: dall'analisi della memoria volatile non sono state riscontrate particolari risorse in riferimento al malware Emotet e rilasciate sul filesystem.

L'analisi OSINT sul malware in esamina è stata effettuata passando in ingresso il file .docx del malware agli applicativi Virus Total [162], Hybrid-Analysis [163], OPSWAT Metadefender Cloud [168] e Intezer. Il file viene rilevato come malevolo da ogni soluzione sopra citata ed etichettato come "Emotet" o "Invoice", della famiglia dei "Banking Trojan" e "Spyware". Viene identificata la data di creazione a "2018-02-07" e quella di primo avvistamento in "2020-06-11". Emotet è noto per essere un virus che si diffonde mediante email di spam contenenti allegati dannosi di tipo Microsoft Word: scaricando ed aprendo il documento il malware viene iniettato del sistema mediante una macro VBA eseguita su Powershell, per poi cercare di diffonfersi come un worm in altri sistemi della rete [169]. A conferma di ciò, Virus Total [162] divulga denominazioni alternative che fanno riferimento allo stesso file: "Outstanding Invoices.doc", "Order Confirmation.doc", "Invoice Number 002357.doc", "Invoice.doc", "Awaiting for your confirmation.doc" e similari. Nella sezione "Relations" di Virus Total [162] e "Network Analysis" di Hybrid-Analysis [163], vengono confermati i tentativi di contatto verso due URL, riscontrate nell'analisi delle connessioni di rete ed attualmente non raggiungibili: "www.harboursidechurch.org/YNgfxDP/", "kanmoretail.com/lKhn5rc", "gize24.com". Viene esplicitata, nelle soluzioni di OSINT, una numerosa quantità di chiavi di registro aggiunte, modificate o eliminate dal malware in esamina, che però a parità dei file rilasciati, non trovano un corrispettivo nel memory dump acquisito. Questa evidenza, analizzata nel profondo, porta alla conclusione che il malware svolga azioni evasive in caso di esecuzione in ambiente virtualizzato, a conferma della limitata presenza di dati

```
┌──(parallels㉿kali-linux-2021-3)-[~/Desktop/volatility3-develop]
└─$ python3 vol.py -f ../memdump.mem handles --pid 616 | grep Mutant -s
616gressWINWORD.EXE     0×c48d303ee210an0×c0 finMutant  0×1f0001        SM0:616:304:WilStaging_02
616     WINWORD.EXE     0×c48d30165810  0×150   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d3280abe0  0×21c   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d2f44ca80  0×224   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d32eedb00  0×230   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d30003750  0×238   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d2fe37410  0×23c   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d3031f340  0×240   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d32e1d700  0×24c   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d2fddaec0  0×2bc   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d304e76d0  0×30c   Mutant  0×100000        _ClassificationAuditCacheMutex_
616     WINWORD.EXE     0×c48d312edfc0  0×388   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d302bd1f0  0×390   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d30933310  0×3b4   Mutant  0×1f0001        x64_10MU_ACBPIDS_S-1-5-5-0-156444
616     WINWORD.EXE     0×c48d2fc1afc0  0×3dc   Mutant  0×1f0001        BootProfiler
616     WINWORD.EXE     0×c48d30933550  0×3f8   Mutant  0×1f0001        SM0:616:120:WilError_01
616     WINWORD.EXE     0×c48d31aa1390  0×5b8   Mutant  0×100000        Office16.B1E641B5-F92B-4B82-83B7-10DC868435E8
616     WINWORD.EXE     0×c48d32d15550  0×5c4   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d314e6790  0×640   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d3241ac70  0×774   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d329e27d0  0×7c8   Mutant  0×1f0001        x64_10MU_ACB10_S-1-5-5-0-156444
616     WINWORD.EXE     0×c48d31a4fa10  0×884   Mutant  0×1f0001        SessionImmersiveColorMutex
616     WINWORD.EXE     0×c48d320c4740  0×b38   Mutant  0×1f0001        ZonesLockedCacheCounterMutex
616     WINWORD.EXE     0×c48d327bc080  0×b3c   Mutant  0×1f0001        ZonesCacheCounterMutex
616     WINWORD.EXE     0×c48d3022cb40  0×b84   Mutant  0×100000        _ClassificationPolicyCacheMutexFetchPolicy_
616     WINWORD.EXE     0×c48d2e645a90  0×b98   Mutant  0×1f0001        MTX_MSO_AdHoc1_S-1-5-21-2936109392-2283665066-63263006-1001
616     WINWORD.EXE     0×c48d3131f1f0  0×c38   Mutant  0×100000        UsageMetrics_NLMicrosoft_Office_16_5481855601190024192
616     WINWORD.EXE     0×c48d31270220  0×cd0   Mutant  0×100000        UsageMetrics_NLMicrosoft_Office_16_3787169836345545405
616     WINWORD.EXE     0×c48d2fe9bf40  0×ce8   Mutant  0×100000        Disco_Microsoft_Office_16_Catalog
616     WINWORD.EXE     0×c48d2ffd74b0  0×cf0   Mutant  0×100000        Disco_Microsoft_Office_16_MappingLock
616     WINWORD.EXE     0×c48d3030f080  0×d04   Mutant  0×100000        Disco_Shm_Microsoft_Office_16_MA_42b497c7dd71bf3e
616     WINWORD.EXE     0×c48d31cedfc0  0×f2c   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d3023ee00  0×f3c   Mutant  0×1f0001
616     WINWORD.EXE     0×c48d3035a970  0×f40   Mutant  0×100000        _ClassificationPolicyCacheMutexPolicy_
616     WINWORD.EXE     0×c48d2f45bd70  0×1150  Mutant  0×1f0001
616     WINWORD.EXE     0×c48d312ca2e0  0×1154  Mutant  0×1f0001
```

Figura 6.53. Dettaglio sugli oggetti Mutant correlati al processo con PID 616

all'interno del dump di memoria rispetto a quelli riportati delle soluzioni di OSINT. Il malware viene inoltre rilevato in 3 Suricata Alert, riportati da Hybrid-Analysis [163]: "ETPRO TROJAN W32/Emotet.v4 Checkin 2", "ETPRO TROJAN W32/Emotet.v4 Checkin 3" e "ET POLICY PE EXE or DLL Windows file download HTTP".

Data Reporting: in questa fase viene mostrato un "summary" delle caratteristiche del malware, rilevate durante l'analisi post-mortem.

Malware name: Emotet

Malware family: TROJAN W32/Emotet

Threat score: 100% malevolo (Crowdstrike Falcon), 67% malevolo (Virus Total) [163]

SHA256: efb6b3a11d1d7f55bb0696453e31a0d6602f03ae74324b502277c7a245f6e5ef

MD5: 169b48efaea1c130551ba83e5898d931

Size: 96.00 KB

File type: MS Word Document

Creation Time: 2018-02-07 08:41:00 UTC

Sistema Operativo: Windows

Memory Dump: memdump.vmem (5.50 GB)

Detonation time: 2022-11-23 14:33:50

Entrypoint: 0x7ff637bb1630

Anti-Virus Results: 41 security vendors and 1 sandbox flagged this file as malicious [162]

MITRE ATT&CK Techniques Detection: 9 tecniche di attacco(1 High, 8 Low)

MITRE ATT&CK ID: TA0002, T1059, T1064, T1203, TA0004, T1055, TA0005, T1036, T1140, T1497, T1562.001, TA0007, T1010, T1018, T1057, T1082, T1083, TA0011, T1071, T1095, T1105, T1573

Installation/Execution: il malware viene installato per mezzo di un dispositivo USB ed eseguito all'apertura di un documento .docx per mezzo di una macro VBA incorporata. La macro inietta uno script nel sistema mediante Powershell.

Persistence: il malware presenta abilità di corruzione di chiavi di registro, la cui modifica viene offuscata in modo da non essere rilevata durante l'analisi della memoria. Inoltre Emotet si propaga nel sistema generando processi malevoli, che hanno la capacità di rimanere in stato dormiente nel sistema infetto.

Evasive: il malware presenta abilità di offuscamento di file e informazioni, mascheramento del file eseguibile e process injection. Emotet riconosce se è in esecuzione su di una macchina virtuale o sandbox e, nel caso, rimane inattivo. Il virus è polimorfico, ovvero il suo codice cambia leggermente ogni volta che viene eseguito, complicando l'identificazione da parte delle soluzioni anti-virus.

Discovery: il malware è in grado di ottenere una panoramica dettagliata dell'ambiente infetto, costruita dalla correlazione delle informazioni di sistema derivate dalla lista dei processi in esecuzione, delle applicazioni windows aperte, degli host file, delle software policy, dei dispositivi collegati.

Anti-Reverse Engineering: il file eseguibile interno al documento Word malevolo non presenta l'utilizzo di Packer o altre tecniche di anti-reverse engineering

Privilege Escalation: il processo di sistema si connette alla rete, successivamente ad una iniezione di codice, alloca la memoria in altri processi e sovrascrive dati, inietta file nell'applicazione Windows e tenta la privilege escalation a sistema.

Network Communication: il malware tenta l'apertura di canali di comunicazione verso server C&C. In particolare, si rilevano 18 richieste DNS su domini distinti e richieste SSDP, volte a identificare i dispositivi disponibili nella rete locale. A tal riguardo, il malware utilizza le informazioni di rete ricavate per tentare di infettare sistemi interni alla rete locale.

Process Tree:

- svchost.exe (PID 1672)
- svchost.exe (PID 3612)

- svchost.exe (PID 7084)
- svchost.exe (PID 800)
  - OpenWith.exe (PID 6220)
    - WINWORD.EXE (PID 4892, 0 Thread)
    - WINWORD.EXE (PID 616, 23 Thread)

File Dropped: non vengono rilevati specifici file rilasciati sul sistema sul malware, ad esclusione delle dll importate in fase di esecuzione.

Resources Dropped: il malware non rilascia alcuna risorsa nel filesystem che possa essere individuata mediante metodi di Memory-Forensics.

Registry Keys Set: il malware occulta la creazione, modifica o cancellazione di 184 chiavi di registro.

Screenshot:

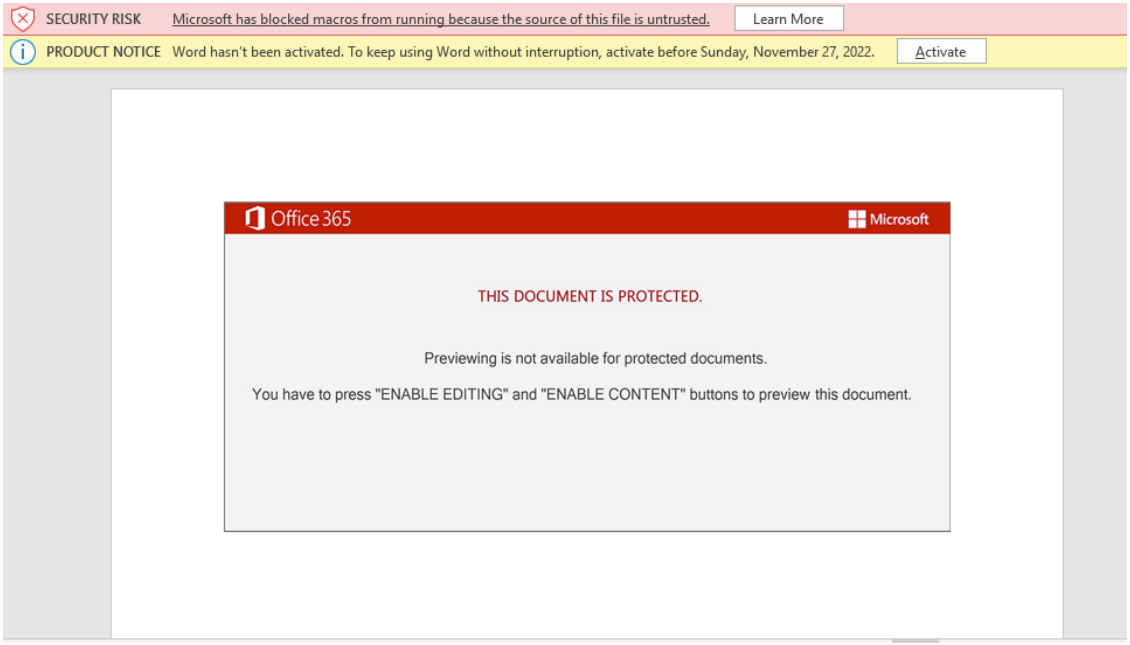


Figura 6.54. Esecuzione del malware Emotet

Analisi comportamentale: il malware in esamina, identificato come Emotet, infetta un sistema Windows andando ad iniettare una macro VBA malevola mediante il processo powershell.exe. La variante di Emotet presente nel memory dump analizzato presenta meccanismi di evasione delle soluzioni anti-virus, caso per cui la ricerca delle evidenze è complicata dallo stato dormiente dei processi malevoli principali, correlati alla detonazione del file Word sorgente. Il malware, una volta infettato il sistema, tenta l'apertura di canali di comunicazione verso server C&C, mediante Remote Procedure Call, contattando 18 domini distinti. Inoltre, effettua delle richieste SSDP per ottenere una mappatura dei dispositivi presenti nella rete al fine di diffondersi in essi come un worm. Il malware, inoltre, genera una panoramica del sistema infetto recuperando informazioni sul sistema infetto e sulle relative policy di sicurezza.

Capitolo 7

Conclusione e direzioni future

I malware di tipo "Stealth" rappresentano al giorno d'oggi un tema di grandissimo spessore, in una società fortemente immatura in termini di sicurezza informatica e inconsapevole delle reali conseguenze che un attacco informatico potrebbe comportare tanto ad un'Organizzazione quanto al singolo individuo. Questo panorama di stabile ignoranza collettiva facilita il progresso di tecniche di attacco sempre più complicate da comprendere e rilevare, ad ogni modo basate sulla sicura complicità dell'utente finale. Difatti, una delle qualità principali dei file-less malware risiede nell'infettare un sistema senza allarmare tanto le soluzioni anti-virus quanto l'utente, che contribuisce alla detonazione del file malevolo mancando di scrupolosità nei controlli basici di sicurezza. L'obiettivo del progetto di tesi è stato fissato nella produzione di una metodologia atta alla rilevazione dei malware di tipo stealth mediante tecniche di forensica applicate alla memoria volatile di un sistema. Il progetto intende concentrarsi sul punto di vista dell'analista forense e include solo marginalmente il "Reverse Engineering" del malware in esamina, discostandosi dalla prospettiva del "malware analyst". A tale scopo la definizione del framework segue ad una descrizione, tanto introduttiva quanto profonda, dei concetti di Digital Forensics, Memory Forensics e Stealthy Malware. Il framework presentato pone le proprie basi sull'estrazione di evidenze dalla memoria RAM, volte a provare la presenza di infezioni nel sistema in esamina, mediante lo strumento Volatility con l'ausilio di soluzioni di analisi statica, dinamica e di rete. Una peculiarità della metodologia proposta risiede nell'affiancamento delle analisi di Open Source Intelligence (OSINT) al processo forense, allo scopo di confermare gli artefatti riscontrati nel sistema o di spingere l'analisi più nel profondo. Il framework nella fase di testing ha riportato risultati positivi, riuscendo nella rilevazione di ogni malware presente nei dump di memoria presi in carica. Allo stesso modo il framework è riuscito nel delineare, anche parzialmente, l'analisi comportamentale dei malware rilevati nei sistemi infetti. La limitazione principale del lavoro svolto è riconosciuta nella complessità, in fase di testing, di detonare i malware in una modalità volta ad esplodere, e quindi rilevare, ogni funzionalità contenuta nel codice maligno. L'origine di questa problematica risiede nella capacità, propria a molti malware, di rilevare l'esecuzione in ambiente virtualizzato e impostarsi in stato dormiente sul sistema infetto, in modo da eludere le soluzioni anti-virus e anti-reverse engineering con pratiche di Anti-Digital Forensics. La riproduzione del framework non è riservata unicamente al personale tecnico, ma pensata per la fruizione libera, spinta dall'interesse. A tal riguardo è stato selezionato ogni strumento in modo da essere gratuito e pubblicamente accessibile, per la riproduzione dei risultati e l'utilizzo indipendente. Si esplicita la volontà futura di automatizzare il processo presentato mediante un apposito script, sviluppato in Python con l'ausilio delle API rilasciate, gratuitamente e non, dalle soluzioni software interne al framework. Il prodotto risultante potrebbe essere integrato come plugin in una soluzione di "Endpoint Detection And Response" (EDR), al fine di monitorare in modo continuo e in tempo reale la presenza di minacce all'interno del sistema e integrando efficaci e specifiche modalità di "Response" sulla base del processo malevolo in input, con l'ausilio di soluzioni di "Artificial Intelligence".

Bibliografia

[1] R. Kaur and A. Kaur, "Digital Forensics", International Journal of Computer Applications, vol. 50, July 2012, DOI 10.5120

[2] "CYBER ATTACK TRENDS Mid Year Report 2021", Checkpoint Research, July 2021

[3] M. R. Al-Mousa, "Analyzing Cyber-Attack Intention for Digital Forensics Using Case-Based Reasoning", International Journal of Advanced Trends in Computer Science and Engineering, vol. 8, November – December 2019, pp. 3243–3247, DOI 10.30534

[4] J.-P. A. Yaacoub, H. N. Noura, O. Salman, and A. Chehab, "Digital Forensics vs. Anti-Digital Forensics: Techniques, Limitations and Recommendations", April 2021, pp. 1–28, DOI 10.48550

[5] M. Pierce, "Detailed Forensic Procedure for Laptop computers", SANS Institute 2021, November 2003

[6] M. Reith, C. Carr, and G. Gunsch, "An Examination of Digital Forensic Models", International Journal of Digital Evidence, vol. 1, no. 3, 2002, DOI 10.1.1.13.9683

[7] K. Kent, S. Chevalier, T. Grance, and H. Dang, "Guide to integrating forensic techniques into incident response", Tech. Rep. NIST Special Publication (SP) 800-86, National Institute of Standards and Technology, Gaithersburg, MD, 2006

[8] W. Halboob, R. Mahmod, N. I. Udzir, and M. T. Abdullah, "Privacy levels for computer forensics: Toward a more efficient privacy-preserving investigation", Procedia Computer Science, vol. 56, July 2015, pp. 370–375, DOI 10.1016

[9] M. Rafique and M.N.A.Khan, "Exploring Static and Live Digital Forensics: Methods, Practices and Tools", International Journal of Scientific & Engineering Research, vol. 4, October 2013, pp. 1048–1056

[10] K. Awuson-David, T. Al-Hadhrami, M. Alazab, N. Shah, and A. Shalaginov, "CFL logging: An approach to acquire and preserve admissible digital forensics evidence in cloud ecosystem", Future Generation Computer Systems, vol. 122, May 2021, pp. 1–13, DOI 10.1016/j.future.2021.03.001

[11] H. Khurana, M. Hadley, N. Lu, and D. A. Frincke, "Smart-grid security issues", IEEE Security & Privacy, vol. 8, February 2010, pp. 81 – 85, DOI 10.1109/MSP.2010.49

[12] K. Conlan, I. Baggili, and F. Breitinger, "Anti-forensics: Furthering digital forensic science through anew extended, granular taxonomy", DFRWS USA 2016 Proceedings of the 16th Annual USA Digital Forensics Research Conference, Aug, 2016, pp. 66–75, DOI 10.1016/j.diin.2016.04.006

[13] M. H. Ligh, A. Case, J. Levy, and A. Walters, "The art of memory forensics: Detecting malware and threats in windows, linux, and mac memory", Wiley, 2014

[14] C. D. Alwis, "Memory Dump Formats", Forensic Focus, February 2018. https://www.forensicfocus.com/articles/memory-dump-formats/

[15] V. Mishra, S. Sutar, P. Nigam, and S. Tripathi, "RAM Forensic Framework", International Journal of Engineering Research & Technology, IJERT, vol. 3, no. 1, 2015, DOI 10.17577/IJERTCONV3IS01036

[16] F. Pagani, "Advances in memory forensics", September 2019. Sorbonne Université

[17] S. Vömel and F. C. Freiling., "Correctness, atomicity, and integrity: defining criteria for forensically-sound memory acquisition", Digital Investigation, vol. 9, November 2012, p. 125–137, DOI 10.1016/j.diin.2012.04.005

[18] B. D.Carrier and J. Grand, "A hardware-based memory acquisition procedure for digital investigations", Digital Investigation, vol. 1, 2004, pp. 50–60, DOI 10.1016/j.diin.2003.12.001

[19] L. Zhang, L. Wang, R. Zhang, S. Zhang, and Y. Zhou, "Live memory acquisition through firewire", Forensics in Telecommunications, Information, and Multimedia, Berlin, Heidelberg, 2011, pp. 159–167, DOI 10.1007/978-3-642-23602-0_14

[20] M. N. Faiz and W. A. Prabowo, "Comparison of Acquisition Software for Digital Forensics Purposes", Kinetik, vol. 4, November 2019, pp. 37–44, DOI 10.22219/kinetik.v4i1.687

[21] M. F. Martínez, "Impact of Tools on The Acquisition of RAM Memory", The International Journal of Cyber Forensics and Advanced Threat Investigations, vol. 1, 2021, pp. 3–17, DOI 10.46386/ijcfati.v1i1-3.12

[22] D. Likhar and M. Rajput, "Study of Memory Forensics: Memory Analysis Technique", International Research Journal of Engineering and Technology (IRJET), vol. 7, September 2020

[23] E. Chan, W. Wan, A. Chaugule, and R. Campbell, "A Framework for Volatile Memory Forensics", DOI 10.1.1.232.8871

[24] F. Pagani, O. Fedorov, and D. Balzarotti, "Introducing the Temporal Dimension to Memory Forensics", ACM Transactions on Privacy and Security, vol. 22, May 2019, pp. 1–21, DOI 10.1145/3310355

[25] M. Gruhn and F. C. Freiling, "Evaluating atomicity, and integrity of correct memory acquisition methods", DFRWS 2016 Europe — Proceedings of the Third Annual DFRWS Europe, March 2016, pp. 1–10, DOI 10.1016/j.diin.2016.01.003

[26] S. L. Berre, "From corrupted memory dump to rootkit detection", 2018. https://exatrack.com/public/Memdump_NDH_2018.pdf

[27] A. Case and G. Richard III, "Memory forensics: The path forward", Digital Investigation, vol. 20, March 2017, pp. 23–33, DOI 10.1016/j.diin.2016.12.004

[28] K. Gupta and A. Nisbet, "Memory forensic data recovery utilising ram cooling methods", The Proceedings of 14th Australian Digital Forensics Conference, Edith Cowan University, Perth, Australia, December 2016, pp. 11–16, DOI 10.4225/75/58a54cc3c64a2

[29] D. C. Prakoso, I. Riadi, and Y. Prayudi, "Detection of metasploit attacks using RAM Forensic on proprietary operating systems", Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control, vol. 5, May 2020, pp. 155–160, DOI 10.22219/kinetik.v5i2.1037

[30] T. Thomas, M. Piscitelli, B. A. Nahar, and I. Baggili, " Duck Hunt: Memory forensics of USB attack platforms", Forensic Science International: Digital Investigation, vol. 37, August 2021, DOI 10.1016/j.fsidi.2021.301190

[31] A. Orgah and A. Golden Richard III; Case, "MemForC: Memory Forensics Corpus Creation for Malware Analysis", International Conference on Cyber Warfare and Security, 2021, DOI 10.34190/IWS.21.032

[32] S. Jung, S. Seo, Y. Kim, and C. Lee, "Memory Layout Extraction and Verification Method for Reliable Physical Memory Acquisition", Electronics, vol. 10, June 2021, DOI 10.3390/electronics10121380

[33] A. Case and G. Richard III, "Fixing a memory forensics blind spot: Linux kernel tracing", Blackhat USA 2021, Las Vegas, July 2021

[34] M. Hari, J. J. Kizhakkethottam, and A. Madhu, "An Open Source Memory Forensics framework integreated with Blockchain Technology for Linux Operating Systems", Conference on Cyber Security and Ethical Hacking in Blockchain Technology 2021, October 2021, pp. 30–33

[35] J. Ottmanna, F. Breitingerb, and F. Freilinga, "Defining atomicity (and integrity) for snapshots of storage in forensic computing", Proceedings of the Digital Forensics Research Conference Europe (DFRWS EU), April 2022

[36] S. Singh, "The code book: The science of secrecy from ancient egypt to quantum cryptography", Fourth Estate, 1999

[37] B. Bashari Rad, M. Masrom, and S. Ibrahim, "Camouflage In Malware: From Encryption To Metamorphism", International Journal of Computer Science And Network Security (IJCSNS), vol. 12, August 2012, pp. 74–83

[38] RedscanTeam, "Memory forensics: How to detect and analyse memory-resident malware", April 2018. https://www.redscan.com/news/memory-forensics-how-to-detect-and-analyse-memory-resident-malware

[39] PonemonInstitute, "The 2017 state of endpoint security risk", Nov 2017

[40] AristaNetworks, "Fileless malware", July 2022. https://aristanetworks.force.com/AristaCommunity/s/article/Fileless-Malware
[41] D. Pattern, "The Evolution to Fileless Malware",
[42] Trellix, "What is the difference between malware and a virus?", https://www.trellix.com/en-au/security-awareness/ransomware/malware-vs-viruses.html
[43] D. Howard, K. Prince, and B. Schneier, "Security 2020: Reduce security risks this decade", Wiley, 2011
[44] Sudhakar and S. Kumar, "An emerging threat Fileless malware: a survey and research challenges", Cybersecurity, vol. 3, January 2020, DOI 10.1186/s42400-019-0043-x
[45] E. M. Rudd, A. Rozsa, M. Günther, and T. E. Boult, "A Survey of Stealth Malware: Attacks, Mitigation Measures, and Steps Toward Autonomous Open World Solutions", Dec 2016, DOI 10.48550/ARXIV.1603.06028
[46] "Polymorphic and Stealth viruses: Their Evasion and Countermeasure Techniques." shorturl.at/bdqw3
[47] V. Khushali, "A Review on Fileless Malware Analysis Techniques", International Journal of Engineering and Technical Research, vol. 9, May 2020, DOI 10.17577/IJERTV9IS050068
[48] Yiftach Keshet, "What Are LOLBins and How Do Attackers Use Them in Fileless Attacks?", August 2020. https://www.cynet.com/attack-techniques-hands-on/what-are-lolbins-and-how-do-attackers-use-them-in-fileless-attacks/
[49] Rahul Awati, "Stealth virus." https://www.techtarget.com/searchsecurity/definition/stealth-virus
[50] , "Instructions substitution." https://github.com/obfuscator-llvm/obfuscator/wiki/Instructions-Substitution
[51] C. Wueest and H. Anand, "Living off the land and fileless attack techniques", Internet Security Threat report (ISTR), July 2017
[52] Asaf Perlman, "The art of persistence", July 2022, https://www.cynet.com/attack-techniques-hands-on/the-art-of-persistence/
[53] VMray, "Hooking." https://www.vmray.com/glossary/hooking/
[54] S. Grinberg, "API Hooking – Tales from a Hacker's Hook Book", Cynet, October 2020. https://www.cynet.com/attack-techniques-hands-on/api-hooking/
[55] Microsoft, "Hooks Overview." https://docs.microsoft.com/it-it/windows/win32/winmsg/about-hooks
[56] MITRE—ATT&CK, "Input Capture: Credential API Hooking." https://attack.mitre.org/techniques/T1056/004/
[57] NytroRST, "NetRipper." https://github.com/NytroRST/NetRipper/blob/master/README.md
[58] Dejan Lukan, "API hooking and DLL injection on Windows", May 2013, https://resources.infosecinstitute.com/topic/api-hooking-and-dll-injection-on-windows/
[59] Dejan Lukan, "Using SetWindowsHookEx for DLL injection on windows", June 2013, https://resources.infosecinstitute.com/topic/using-setwindowshookex-for-dll-injection-on-windows/
[60] Dejan Lukan, "Using CreateRemoteThread for DLL injection on Windows", May 2013, https://resources.infosecinstitute.com/topic/using-createremotethread-for-dll-injection-on-windows/
[61] G. Hoglund and J. Butler, "Rootkits: Subverting the Windows Kernel", Addison Wesley Professional, July 2005
[62] Kaspersky Lab, "Gauss: Abnormal Distribution", 2012, https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/20134940/kaspersky-lab-gauss.pdf
[63] Kurt Baker, "Malware Analysis", January 2022, https://www.crowdstrike.com/cybersecurity-101/malware/malware-analysis/
[64] Greg Hoglund and James Butler, "VICE – Catch the Hookers!"", January 2004,
[65] J. Rutkowska, "System Virginity Verifier – Defining the Roadmap for Malware Detection on Windows Systems"", September 2005,
[66] J. Rutkowska, "Detecting Windows Server Compromises with Patchfinder 2"", September 2004,

[67] J. Nick L. Petroni, T. Fraser, J. Molina, and W. A. Arbaugh, "Copilota Coprocessor-based Kernel Runtime Integrity Monitor", 2004, http://usenix.org/publications/library/proceedings/sec04/tech/full_papers/petroni/petroni.pdf

[68] G. Jacob, H. Debar, and E. Filiol, "Behavioral detection of malware: From a survey towards an established taxonomy", Journal in Computer Virology, vol. 4, 08 2008, pp. 251–266, DOI 10.1007/s11416-008-0086-0

[69] S. Saad, W. Briguglio, and H. Elmiligi, "The curious case of machine learning in malware detection", 2019, DOI 10.48550/ARXIV.1905.07573

[70] S. Kilgallon, L. D. L. Rosa, and J. Cavazos, "Improving the effectiveness and efficiency of dynamic malware analysis with machine learning", 2017 Resilience Week (RWS), 2017, pp. 30–36, DOI 10.1109/RWEEK.2017.8088644

[71] Microsoft, "Managing privileges in a file system", 2021, https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/privileges

[72] A. Bertram, "How to Change Registry Permissions with PowerShell", 2020, https://www.ipswitch.com/blog/how-to-change-registry-permissions-with-powershell

[73] Microsoft Documentation, "RegistryRights Enum." https://learn.microsoft.com/en-us/dotnet/api/system.security.accesscontrol.registryrights?redirectedfrom=MSDN&view=netframework-4.8

[74] Adam Bertram, "How to Use PowerShell to Manage Folder Permissions." https://petri.com/how-to-use-powershell-to-manage-folder-permissions/

[75] A. Todd, J. Benson, G. Peterson, T. Franz, M. Stevens, and R. Raines, "Analysis of tools for detecting rootkits and hidden processes", Advances in Digital Forensics III, New York, NY, 2007, pp. 89–105, DOI 10.1007/978-0-387-73742-3_6

[76] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "MalInsight: A systematic profiling based malware detection framework", Journal of Network and Computer Applications, vol. 125, January 2019, p. 236–250, DOI 10.1016/j.jnca.2018.10.022

[77] , "Cuckoo Sanbox." https://cuckoosandbox.org/

[78] Will Koehrsen, "Neural Network Embeddings Explained", 2018, https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

[79] A. Gupta, "Data Provenance", 209, p. 608, DOI 10.1007/978-0-387-39940-9_1305

[80] , "Novelty and Outlier Detection." https://scikit-learn.org/stable/modules/outlier_detection.html#novelty-detection-with-local-outlier-factor

[81] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, and H. Chen, "You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis", 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020, 2020

[82] M. Agarwal, R. Puzis, J. Haj-Yahya, P. Zilberman, and Y. Elovici, "Anti-forensic = Suspicious: Detection of Stealthy Malware that Hides Its Network Traffic", Hal Open Science, September 2018, pp. 216–230, DOI 10.1007/978-3-319-99828-2_16

[83] P. Luckett, J. T. McDonald, W. B. Glisson, R. Benton, J. Dawson, and B. A. Doyle, "Identifying stealth malware using CPU power consumption and learning algorithms", Journal of Computer Security, vol. 26, 2018, p. 589–613, DOI 10.3233/JCS-171060

[84] C. Spensky, H. Hu, and K. Leach, "LO-PHI: Low-Observable Physical Host Instrumentation for Malware Analysis ", February 2016, DOI 10.14722/ndss.2016.23121

[85] M. Botacin, A. Grégio, and M. Alves, "Near-memory & in-memory detection of fileless malware", The International Symposium on Memory Systems, New York, 2020, pp. 22–38, DOI 10.1145/3422575.3422775

[86] Egidius Mysliwietz, "Identifying rootkit stealth strategies."

[87] R. Sihwail, K. Omar, and K. A. Zainol Ariffin, "An effective memory analysis for malware detection and classification", Computers, Materials and Continua, vol. 67, February 2021, pp. 2301–2320, DOI 10.32604/cmc.2021.014510

[88] M. Manna, A. Case, A. Ali-Gombe, and G. G. Richard, "Memory analysis of .net and .net core applications", Forensic Science International: Digital Investigation, vol. 42, 2022, p. 301404, DOI https://doi.org/10.1016/j.fsidi.2022.301404. Proceedings of the Twenty-Second Annual DFRWS USA

[89] AccessData, "FTK IMAGER VERSION 4.5." https://accessdata.com/product-download/ftk-imager-version-4-5

[90] Packtpub , " Acquiring memory using FTK Imager ." https://subscription.packtpub.com/book/networking-&-servers/9781782174905/2/ch02lvl1sec23/acquiring-memory-using-ftk-imager
[91] DFIRScience , " Forensic Memory Acquisition in Windows - FTK Imager." www.youtube.com/watch?v=1OxR4KLj-4I
[92] A. Walters and N. Petronni Jr., "Volatools: Integrating volatile memory forensics into the digital investigation process", Black Hat DC, 2007
[93] Volatility Foundation , " About The Volatility Foundation ." https://www.volatilityfoundation.org/about
[94] Volatility Foundation, " Volatility." https://github.com/volatilityfoundation/volatility
[95] Volatility Foundation , "Community ." https://github.com/volatilityfoundation/community
[96] Volatility Foundation , " Release Downloads." https://www.volatilityfoundation.org/releases
[97] Volatility Foundation, " Volatility 3 v1.0.0 (Python 3 Rewrite)." https://www.volatilityfoundation.org/releases-vol3
[98] HackTricks, " Volatility - CheatSheet ." https://book.hacktricks.xyz/generic-methodologies-and-resources/basic-forensic-methodology/memory-dump-analysis/volatility-examples
[99] Volatility Foundation, " Command Reference ." https://github.com/volatilityfoundation/volatility/wiki/Command-Reference
[100] Andreas Schuster, " YARA: An Introduction." https://www.first.org/resources/papers/conference2014/first_2014_-_schuster-_andreas_-_yara_basic_and_advanced_20140619.pdf
[101] Yara-Rules, " rules." https://github.com/Yara-Rules/rules.git
[102] Volatility Foundation , " Command Reference Mal." https://github.com/volatilityfoundation/volatility/wiki/Command-Reference-Mal#yarascan
[103] Security Ninja , " Static malware analysis." https://resources.infosecinstitute.com/topic/malware-analysis-basics-static-analysis/
[104] gchq, "Cyberchef ." https://gchq.github.io/CyberChef/
[105] BalaGanesh, "Cooking Malicious Documents with Cyberchef – Detect & Respond ." https://www.socinvestigation.com/cooking-malicious-documents-with-cyberchef-detect-respond/
[106] winitor, "pestudio free ." https://www.winitor.com/download
[107] winitor, "pestudio 9.44 features ." https://www.winitor.com/tools/pestudio/current/pestudio-features.pdf
[108] MajorGeeks.Com , " Detect It Easy 3.06." https://www.majorgeeks.com/files/details/detect_it_easy.html
[109] horsicq , " Detect-It-Easy." https://github.com/horsicq/Detect-It-Easy
[110] Pieter Arntz , "Explained: Packer, Crypter, and Protector ." https://www.malwarebytes.com/blog/news/2017/03/explained-packer-crypter-and-protector
[111] Wikipedia , " Process Explorer." https://it.wikipedia.org/wiki/Process_Explorer
[112] Microsoft Sysinternals, " Process Explorer v16.43." https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer
[113] Ionos, " Reverse engineering software." https://www.ionos.com/digitalguide/websites/web-development/reverse-engineering-of-software/
[114] hex-rays, "IDA Freeware ." https://hex-rays.com/ida-free
[115] hex-rays, "IDA Pro ." https://hex-rays.com/ida-pro/
[116] Wikipedia , "Interactive Disassembler ." https://en.wikipedia.org/wiki/Interactive_Disassembler
[117] hex-rays , "IDA Help: Graph view ." https://www.hex-rays.com/products/ida/support/idadoc/42.shtml
[118] Monnappa K A, "Learning Malware Analysis." https://www.oreilly.com/library/view/learning-malware-analysis/9781788392501/12556df2-7825-4e43-8811-c0fabeab78d8.xhtml
[119] NtQuery , " Scylla." https://github.com/NtQuery/Scylla

[120] vk.com, "How to Dump a packed executable with Scylla ." https://vk.com/@reverse_engine-how-to-dump-a-packed-executable-with-scylla
[121] Dimitar Kostadino, "Analyzing Malware Network Behavior ." https://resources.infosecinstitute.com/topic/analyzing-malware-network-behavior/
[122] Wireshark , "Wireshark About ." https://www.wireshark.org/index.html#aboutWS
[123] Wireshark, " Download Wireshark." https://www.wireshark.org/download.html
[124] Comptia, "What Is Wireshark and How Is It Used? ." shorturl.at/flpL5
[125] Mandiant, " Fakenet-ng." https://github.com/mandiant/flare-fakenet-ng
[126] CrowdStrike Falcon® Sandbox. , " hybrid-analysis." https://www.hybrid-analysis.com/
[127] Maltego , "Hybrid Analysis ." https://www.maltego.com/transform-hub/hybrid-analysis/
[128] Kamil Bojarski, " Hunting for implants - OSINT malware analysis." https://counterintelligence.pl/en/2022/03/na-tropie-implantow-osintowa-analiza-malwareu/
[129] Joe Sandbox , "Joe Sandbox Cloud Basic." https://www.joesandbox.com/#windows
[130] Joe Security, "Overview." https://www.joesecurity.org/joe-sandbox-cloud#overview
[131] Joe Security, " Key Features." https://www.joesecurity.org/joe-sandbox-cloud#key-features
[132] VirusTotal, " VirusTotal - Home." https://www.virustotal.com/gui/home/upload
[133] VirusTotal, "About us - VirusTotal ." https://support.virustotal.com/hc/en-us/categories/360000160117-About-us
[134] OPSWAT , "MetaDefender Cloud ." https://metadefender.opswat.com/
[135] Shivaji Sarkar , " OPSWAT MetaDefender Cloud As a Service ." https://www.opswat.com/blog/opswat-metadefender-cloud-as-a-service
[136] Thanursan , "Comparison of Memory Acquisition Software for Windows." https://thanursan.medium.com/comparison-of-memory-acquisition-software-for-windows-e8c6d981db23
[137] Security Ninja , " Memory Forensics: Enumeration." https://resources.infosecinstitute.com/topic/memory-forensics-power-part-2/
[138] Hybrid-Analysis, " sample-20200214.exe ." https://www.hybrid-analysis.com/sample/31f910e7cd03fb0c447c5a6764c52d799312dc0188a890214bde4d60fb5a53d8/630a2da9771a77139312dbaf
[139] Virus Total, "cell_jr.exe ." https://www.virustotal.com/gui/file/31f910e7cd03fb0c447c5a6764c52d799312dc0188a890214bde4d60fb5a53d8/behavior
[140] Dr.Web , "Trojan.Encoder.33064." https://vms.drweb-av.it/virus/?i=22623581
[141] bangdroid , "What is LockApp.exe?." https://answers.microsoft.com/en-us/windows/forum/all/what-is-lockappexe/cc075a02-7404-4c5e-ba86-6cceadb16862
[142] Johannes Ullrich, "When Windows 10 Comes to Live: The First Few Minutes in the Live of a Windows 10 System." https://isc.sans.edu/diary/When+Windows+10+Comes+to+Live%3A+The+First+Few+Minutes+in+the+Live+of+a+Windows+10+System/24838
[143] digitalf0rensics , "Digital Forensics and Incident Response, Tag: NTUSER.DAT." https://digitalf0rensics.wordpress.com/tag/ntuser-dat/
[144] Andrea Fortuna, " Amcache and Shimcache in forensic analysis." https://andreafortuna.org/2017/10/16/amcache-and-shimcache-in-forensic-analysis/
[145] Malware bazaar , "MalwareBazaar Database." https://bazaar.abuse.ch/browse/yara/with_sqlite/
[146] jipegit, "Spyeye.rar ." https://github.com/jipegit/yara-rules-public/blob/master/Banker/Spyeye.yar
[147] Yara-Rules , "MALW_LURK0.yar ." https://github.com/Yara-Rules/rules/blob/master/malware/MALW_LURK0.yar
[148] jackcr , "shylock.yar." https://github.com/jackcr/yara-memory/blob/master/shylock.yar
[149] Josh Homan, "Challenge #4 Solution." https://www.fireeye.com/content/dam/fireeye-www/global/en/blog/threat-research/flareon/2015solution4.pdf
[150] pc-facile, " GLOSSARIO INFORMATICO: LSA." http://www.pc-facile.com/glossario/lsa/

[151] Cybercrypto, "The RAT Likes To Gossip ." https://cybercrypto.net/remcos/
[152] F-secure, "Trojan:W32/VB.BKX ." https://www.f-secure.com/v-descs/trojan_w32_vb_bkx.shtml
[153] Fortiguard, "W32/Waski.B!tr ." https://www.fortiguard.com/encyclopedia/virus/6048964
[154] StackOverflow , "Need help understanding "DBWinMutex" ." https://stackoverflow.com/questions/66853562/need-help-understanding-dbwinmutex
[155] File.net, "Come eliminare l'errore lsass." https://www.file.net/it/processo/lsass.exe.html
[156] Red Hot Cyber, "Top Malware: Stuxnet e gli albori della guerra cibernetica.." https://www.redhotcyber.com/post/la-storia-dei-malware-stuxnet/
[157] Xavier, "Memory Analysis Report Stuxnet ." https://www.solomonsonya.com/cyber/memory_analysis/malware_analysts_cook_book/stuxnet/abbreviated_analysis_summary/_html/analysis_report_stuxnet.vmem.html
[158] executable.868.exe, "Hybrid-Analysis." https://hybrid-analysis.com/sample/6f293f095e960461d897b688bf582a0c9a3890935a7d443a929ef587ed911760/6379e0a1d8f1db5d17650b11
[159] executable.1928.exe, "Hybrid-Analysis." https://hybrid-analysis.com/sample/20a3c5f02b6b79bcac9adaef7ee138763054bbedc298fb2710b5adaf9b74a47d/636243042589a83f46708d66
[160] executable.940.exe, "Hybrid-Analysis." https://hybrid-analysis.com/sample/3620aab5a7a24869a7366a78189c13d993da1fa34029c88e7628e76f9f76d39b/59073b14aac2eda20b148026
[161] Symantec Security Response, "W32.Stuxnet Dossier." https://www.wired.com/images_blogs/threatlevel/2011/02/Symantec-Stuxnet-Update-Feb-2011.pdf
[162] Virus Total, "Outstanding Invoices.doc." https://www.virustotal.com/gui/file/efb6b3a11d1d7f55bb0696453e31a0d6602f03ae74324b502277c7a245f6e5ef/detection
[163] Hybrid-Analysis, "Invoice." https://www.hybrid-analysis.com/sample/efb6b3a11d1d7f55bb0696453e31a0d6602f03ae74324b502277c7a245f6e5ef/5a7ad43c7ca3e11d187cee97
[164] Revert Service, "SSDP Discovery (SSDPSRV) Service Defaults in Windows 7." http://revertservice.com/7/ssdpsrv/
[165] Securelist, "Emotet modules and recent attacks." https://securelist.com/emotet-modules-and-recent-attacks/106290/
[166] Yara-Rules, "MALW_Ponmocup.yar." https://github.com/Yara-Rules/rules/blob/master/malware/MALW_Ponmocup.yar
[167] Hackmuraz Hacking Blog, "Vulnerabilità grave scoperta su Windows 7 e sfruttata attraverso Chrome." hackmuraz.ch/2019/03/vulnerabilita-grave-scoperta-su-windows.html
[168] OPSWAT Metadefender Cloud, "169b48efaea1c130551ba83e5898d931." https://metadefender.opswat.com/results/file/efb6b3a11d1d7f55bb0696453e31a0d6602f03ae74324b502277hash/multiscan
[169] Kaspersky, "Emotet: come proteggersi al meglio dal trojan." https://www.kaspersky.it/resource-center/threats/emotet