

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica – Scienza e Ingegneria

Corso di Laurea in Ingegneria Informatica

Analisi delle performance ed espressività di  
regole per Intrusion Detection System  
generate in linguaggio LUA per scenari  
industriali

Relatore:

Prof. Dr. Marco Prandini

Presentata da:

Edoardo Marinelli

Correlatori:

Dr. Andrea Melis

Dr. Amir Al Sadi

Dr. Giacomo Gori

Dr. Lorenzo Rinieri

Sessione II

Anno Accademico 2022/2023



## Abstract

In un ambiente industriale dove l'automazione riveste crescente importanza, il **protocollo Modbus** si afferma come componente fondamentale per l'affidabilità delle comunicazioni grazie alla sua struttura **master/slave**.

Questa ricerca presenta la creazione di una rete Modbus in cui un **client (slave) simula una macchina industriale comunicando con un server (master)**, il quale svolge il compito di un PLC, offrendo un'immagine concreta delle sfide comunicative in scenari reali. Questa infrastruttura è stata creata tramite una libreria Python completa per l'implementazione del protocollo di comunicazione Modbus per l'automazione industriale, ovvero **pyModbus**.

L'enfasi, però, non è posta unicamente sulla descrizione dell'infrastruttura. Piuttosto, si focalizza sull'elaborazione e applicazione di regole **Suricata** per un'analisi dettagliata del traffico Modbus, al fine di rilevare **attività anomale o minacce**, entrambe da associare ad attacchi al sistema o malfunzionamenti causati da agenti interni come guasti.

Tale fine è stato possibile anche grazie al linguaggio di programmazione **Lua**, il quale ha permesso di ampliare l'espressività delle regole Suricata, vincolata alle keywords.

Un ulteriore aspetto cardine riguarda l'ottimizzazione dell'**espressività dei file di log**, per facilitare l'identificazione tempestiva di anomalie da parte degli analisti di sicurezza. Per fare ciò è stato utilizzato sempre Lua, data la sua forte integrazione con Suricata.

L'obiettivo finale è potenziare sia la sicurezza che la chiarezza nella gestione dei dati in contesti industriali automatizzati.



# Ringraziamenti

*"Ringrazio con tutto me stesso la mia famiglia per il supporto diretto ricevuto durante il mio percorso di studi e per avermi incoraggiato ad arrivare fino a dove sono oggi. Un grazie sincero al mio gruppo di amici per la loro presenza. Condividere risate e momenti di pausa ha reso questo percorso più dolce. La vostra amicizia è inestimabile."*



# Indice

<b>Abstract</b>	<b>i</b>
<b>Ringraziamenti</b>	<b>iii</b>
<b>Elenco delle Figure</b>	<b>vii</b>
<b>Elenco delle Tabelle</b>	<b>ix</b>
<b>1 Introduzione</b>	<b>xiii</b>
<b>2 Scenari applicativi e stato dell'arte</b>	<b>3</b>
2.1 Suricata . . . . .	3
2.1.1 IDS . . . . .	3
2.1.2 Storia di Suricata . . . . .	5
2.1.3 Installazione . . . . .	6
2.1.3.1 Repository OISF . . . . .	6
2.1.3.2 File sorgenti . . . . .	6
2.1.4 Come funziona . . . . .	8
2.1.5 Regole . . . . .	9
2.1.6 Files di Log . . . . .	12
2.1.7 Integrazione con LUA . . . . .	13
2.1.7.1 Breve introduzione . . . . .	13
2.1.7.2 Lua detection . . . . .	13
2.1.7.3 Lua output . . . . .	15
2.2 Modbus . . . . .	16

2.2.1	Origini e storia . . . . .	16
2.2.2	Tipologia di dati . . . . .	17
2.2.3	Standard usati da Modbus seriale . . . . .	18
2.2.3.1	Standard RS232 . . . . .	18
2.2.3.2	Standard RS485 . . . . .	19
2.2.4	Versioni del protocollo . . . . .	19
2.2.4.1	Modbus ASCII . . . . .	20
2.2.4.2	Modbus RTU . . . . .	21
2.2.4.3	Modbus TCP/IP . . . . .	22
<b>3</b>	<b>Analisi progettuale ed Implementazione</b>	<b>25</b>
<b>4</b>	<b>Risultati</b>	<b>37</b>
<b>5</b>	<b>Conclusioni e sviluppi futuri</b>	<b>41</b>
	<b>Bibliografia</b>	<b>45</b>



# Elenco delle Figure

2.1	Struttura di una regola Suricata . . . . .	10
2.2	Struttura standard RS232 . . . . .	18
2.3	Elenco funzioni Modbus . . . . .	20
3.1	Esempio Format CEF . . . . .	28
3.2	Oggetto della comunicazione server/client . . . . .	30
3.3	Stringhe diagnostiche . . . . .	31
3.4	Vista traffico con Wireshark . . . . .	32
4.1	Infrastruttura Vagrant . . . . .	38
4.2	Log luatest.log . . . . .	39
4.3	Log ceftest.cef . . . . .	40



# Elenco delle Tabelle

2.1	Frame Modbus ASCII . . . . .	21
2.2	Frame Modbus RTU . . . . .	22
2.3	Frame Modbus TCP/IP . . . . .	23



# Elenco dei Codici

2.1	Installazione Suricata Repository OISF Ubuntu . . . . .	6
2.2	Installazione Suricata Repository OISF Debian . . . . .	6
2.3	Download file sorgente . . . . .	7
2.4	Opzioni ./configure . . . . .	7
2.5	Funzione Init Lua Detection . . . . .	14
2.6	Funzione Match Lua Detection . . . . .	14
3.1	Funzione genera payload . . . . .	30
3.2	suricata.yaml: Interfaccia . . . . .	33
3.3	suricata.yaml: Modbus . . . . .	33
3.4	suricata.yaml: Script Lua Output . . . . .	33
3.5	Regola detection traffico Modbus . . . . .	34



# Capitolo 1

## Introduzione

L'efficacia e la sicurezza delle comunicazioni tra dispositivi diventano essenziali in un contesto industriale in cui l'automazione gioca un ruolo sempre più importante. In questo contesto, il protocollo **Modbus** emerge come uno dei componenti chiave dell'infrastruttura di automazione grazie alla sua struttura master/slave altamente affidabile. Tuttavia, insieme all'aumento della connettività e della capacità dei dispositivi, è sempre più necessario salvaguardare e tenere sotto controllo il flusso di dati attraverso questi ricettacoli.

In questo scenario, la presente ricerca descrive lo sviluppo di una rete Modbus **master/slave** in cui un client, che emula una macchina industriale, trasmette dati a un **PLC (Programmable Logic Controller)**. Questa simulazione mira a fornire un quadro realistico delle sfide e dei problemi di comunicazione negli ambienti industriali reali, piuttosto che servire solo come esercizio teorico.

Tuttavia, l'obiettivo principale di questa tesi non è la semplice descrizione dell'infrastruttura, ma piuttosto la creazione e l'applicazione delle regole Suricata. L'obiettivo primario è quello di sviluppare una metodologia che consenta un'analisi minimamente invasiva del traffico dati Modbus, identificando ed analizzando determinati **pattern o valori diagnostici** che potrebbero rivelare **attività anomale o potenzialmente dannose**, sia da associare

ad **attacchi al sistema** o semplici **malfunzionamenti casuali**.

La seconda area di interesse della ricerca è **l'espressività dei file di log** creati durante l'analisi. Un log chiaro, dettagliato e ben organizzato può facilitare notevolmente il compito degli analisti di sicurezza nell'identificare rapidamente potenziali violazioni o attività sospette. Di conseguenza, una parte consistente del lavoro è stata dedicata alla ricerca e all'offerta di metodi per migliorare l'efficacia e la chiarezza di tali file di log.





## Capitolo 2

# Scenari applicativi e stato dell'arte

### 2.1 Suricata

#### 2.1.1 IDS

Un **sistema di identificazione delle intrusioni (Intrusion Detection System)** è un programma che analizza il traffico nella rete alla ricerca di potenziali minacce o attività insidiose. Quando l'IDS rileva problemi di sicurezza o pericoli, invia avvisi ai **gruppi di IT e protezione**.

Molte soluzioni IDS si focalizzano su osservare e comunicare comportamenti e flussi di traffico malevoli quando scorgono una deviazione dalla normale attività. Ma alcuni possono agire attivamente di fronte a tali deviazioni, come interrompere traffico giudicato pericoloso. Le applicazioni IDS sono in genere software che funzionano sull'infrastruttura delle aziende o come meccanismo di tutela della rete. Un IDS esamina i dati che attraversano la rete per individuare comportamenti non ordinari e confronta l'attività di rete con una serie di regole e pattern predefiniti per riconoscere eventuali attività che potrebbero suggerire un attacco o un'intrusione. Se l'IDS identifica qualcosa che corrisponde a una di queste regole o pattern, invia una notifica all'am-

ministratore del sistema. Dopodichè quest'ultimo può quindi esaminare la notifica e intraprendere misure per evitare danni o ulteriori intrusioni.

Gli IDS possono essere classificati in **3 macrogruppi** [1]:

- **Network based intrusion detection system (NIDS)**: I sistemi di rilevamento delle intrusioni di rete sono posizionati in un **punto strategico della rete** per controllare il traffico proveniente da tutti i dispositivi connessi. Questo sistema monitora il traffico che attraversa l'intera sottorete e lo confronta con un insieme di attacchi noti. Se rileva un attacco o un comportamento inusuale, può inviare una notifica all'amministratore. Un esempio di utilizzo di un NIDS è la sua installazione sulla sottorete dove sono posizionati i firewall, per verificare se qualcuno sta tentando di violare il firewall.
- **Host based intrusion detection system (HIDS)**: I sistemi di rilevamento delle intrusioni su host operano su **singoli host o dispositivi presenti nella rete**. Un HIDS controlla i pacchetti in entrata e in uscita dal dispositivo e avvisa l'amministratore in caso di attività sospette o malevole. Questo sistema cattura una rappresentazione degli attuali file di sistema e la confronta con quella precedente. Se i file di sistema vengono modificati o eliminati, viene inviata una notifica all'amministratore per un ulteriore controllo. Un esempio dell'utilizzo di HIDS può essere trovato su macchine ad alta importanza, dove non ci si aspetta che la loro configurazione cambi.
- **Hybrid intrusion detection system (HIDS)**: Gli IDS ibridi combinano le caratteristiche sia degli IDS basati su rete (NIDS) che degli IDS basati su host (HIDS). Questa combinazione mira a **sfruttare i vantaggi di entrambi gli approcci** per fornire una protezione più completa e dettagliata contro le minacce. Integrando questi due approcci, un Hybrid IDS può monitorare il traffico di rete per rilevare attacchi esterni e comportamenti sospetti e allo stesso tempo osservare le attività a livello di host per identificare minacce come malware o atti-

vità sospette all'interno del sistema. Grazie a questa doppia capacità, un Hybrid IDS può fornire una visione più completa delle potenziali minacce e risponde in modo più efficace a una gamma più ampia di scenari di attacco.

La sicurezza informatica è diventata una priorità per le organizzazioni di ogni dimensione a causa della crescente digitalizzazione e dell'interconnessione globale. Di fronte a questa situazione, gli IDS sono diventati un componente essenziale delle risorse di sicurezza di molte aziende. Negli ultimi anni, l'aumento degli attacchi informatici e la crescente consapevolezza della necessità di soluzioni di sicurezza affidabili hanno spinto il mercato dei sistemi di rilevamento delle intrusioni.

### 2.1.2 Storia di Suricata

Le prime fasi di scrittura del codice di Suricata iniziarono nel 2007 dalle menti di **Victor Julien** e **Matt Jonkman** insieme a **William Metcalf**. Insieme alla prima beta pubblica di Suricata rilasciata nel 2009, venne fondata nel 2010 la **Open Information Security Foundation**, una fondazione non-profit con sede a Boston creata per costruire una comunità e sostenere le tecnologie di sicurezza open source. Il team di OISF è composto da esperti di sicurezza, programmatori e leader di settore dedicati alle tecnologie di sicurezza open source. La fondazione ha anche un consorzio di membri che finanziano le operazioni internazionali di OISF e il team di sviluppo di Suricata.

A partire dal 2015 si svolge il **Suricon**, una conferenza a cadenza annuale della comunità delle tecnologie open source che mette in evidenza le discussioni e gli sviluppi relativi a Suricata. Suricon offre l'opportunità di incontrare e interagire con gli sviluppatori, gli utenti e i sostenitori di Suricata, nonché di apprendere le ultime novità e le migliori pratiche sul suo utilizzo.

Suricon si tiene in diverse località ogni anno come ad esempio **Atene**, **Boston** e **Vancouver**, eccezion fatta per il 2020, dove la conferenza si è svolta in modalità virtuale causa Coronavirus.

Attualmente, **ottobre 2023**, l'ultima versione di Suricata rilasciata è la **7.0** in data **18 Luglio 2023**.

### 2.1.3 Installazione

Per quanto riguarda l'installazione di Suricata è necessario fare la distinzione tra due metodi, ovvero tramite la **repository ufficiale OISF** e i **file sorgente**.

#### 2.1.3.1 Repository OISF

Il team di sviluppo di Suricata mette a disposizione una repository dov'è possibile scaricare e avere a disposizione sempre l'ultima versione disponibile del programma.

Per Ubuntu è possibile scaricare ed installare l'ultima versione stabile dalla **PPA suricata-stable**, tramite i seguenti comandi eseguibili dal prompt:

```
1 sudo apt-get install software-properties-common
2 sudo add-apt-repository ppa:oisf/suricata-stable
3 sudo apt-get update
4 sudo apt-get install suricata
```

Listing 2.1: Installazione Suricata Repository OISF Ubuntu

Nel caso di un sistema Debian è sufficiente eseguire il comando:

```
1 sudo apt-get install suricata
```

Listing 2.2: Installazione Suricata Repository OISF Debian

#### 2.1.3.2 File sorgenti

Questo metodo permette di aggiungere un pizzico di personalizzazione all'installazione di Suricata. Prima di procedere è necessario scaricare le librerie necessarie per la compilazione [2].

A questo punto è possibile procedere con il download del file sorgente.

```
1 wget https://www.openinfosecfoundation.org/download/  
2                                     suricata-7.0.0.tar.gz  
3 tar xzvf suricata-7.0.0.tar.gz  
4 cd suricata-7.0.0
```

Listing 2.3: Download file sorgente

Per compilare correttamente il file sorgente, Suricata offre numerose opzioni di configurazione al comando `./configure`, il quale svolge il compito di preparare il sorgente di un programma per la compilazione e l'installazione su un sistema Linux.

```
1 --prefix=/usr/  
2     %Installa il binario Suricata in /usr/bin/  
3 --sysconfdir=/etc  
4     %Posiziona suricata.yaml in /etc/suricata/  
5 --localstatedir=/var  
6     %Genera i log in /var/log/suricata/  
7 --enable-lua  
8     %Abilita il supporto a Lua
```

Listing 2.4: Opzioni `./configure`

Lo script verifica la presenza delle librerie e degli strumenti necessari, e crea un file denominato `Makefile` che contiene le istruzioni per la compilazione. Infine completiamo l'installazione tramite i comandi `make` e `make install`, usati per compilare e installare il sorgente di un programma su un sistema Linux. Il comando `make` esegue le istruzioni contenute nel file `Makefile`, mentre il comando `make install` copia i file binari e le librerie nella posizione appropriata del sistema. `make` e `make install` richiedono i privilegi di scrittura nelle directory in cui si trovano i sorgenti e in quelle in cui si vogliono installare i file. Per questo motivo, spesso si usa il comando `sudo make install` per eseguire l'installazione come utente root.

### 2.1.4 Come funziona

Il funzionamento di Suricata ruota attorno a dei componenti fondamentali:

1. **Regole:** Al centro di Suricata ci sono regole che identificano schemi o comportamenti sospetti che devono essere identificati o fermati. Le regole sono scritte in un linguaggio specifico chiamato "**Suricata IDS Rule Language**". Ogni regola può specificare proprietà diverse come **indirizzi IP di origine/destinazione, porte, protocolli e stringhe o modelli specifici nei pacchetti**. Queste regole vengono utilizzate per analizzare il traffico di rete e generare avvisi o eseguire altre azioni in caso di corrispondenza.
2. **Acquisizione dei pacchetti:** Suricata può catturare pacchetti da diverse fonti come dispositivi di rete, pcap (packet capture) files o da altre fonti, a seconda della configurazione.
3. **Decodifica e analisi:** Una volta catturati, i pacchetti vengono decifrati e analizzati. Durante questa fase, Suricata suddivide il pacchetto nei suoi componenti base (header IP, TCP, ecc.) e li confronta con le regole caricate.
4. **Rilevamento:** Viene generato un avviso quando un pacchetto, una sua caratteristica o un suo comportamento corrisponde a una delle regole. I pacchetti possono anche essere bloccati se Suricata è configurato come **Intrusion Prevention System (IPS)**.
5. **File di Log:** I file di log di Suricata contengono **dettagli sul traffico di rete** e le possibili minacce identificate durante l'analisi di questo traffico come ad esempio il timestamp, il tipo di evento, l'IP e porta di sorgente e destinatario del pacchetto, protocollo e messaggio di alert della regola che ha generato il log.

Ultimo ma non meno importante, il file **suricata.yaml**, posizionato nella cartella `/etc/suricata/`, il quale funge da file di configurazione principale di Suricata, specifica quali protocolli monitorare, dove localizzare le regole di rilevamento, come registrare gli output e una serie di altre variabili operative. Ci sono sezioni per le impostazioni generali, le preferenze di registrazione, le posizioni dei file delle regole, i formati di output, le personalizzazioni specifiche del protocollo, i comportamenti del motore di rilevamento e altro ancora. Gli utenti possono modificare il comportamento di Suricata per adattarlo alle esigenze del proprio ambiente di rete e ai requisiti di sicurezza, cambiando i parametri di questo file.

### 2.1.5 Regole

Suricata utilizza delle regole che fungono da **istruzioni per monitorare e reagire al traffico di rete**. Esse specificano particolari circostanze e pattern di traffico, consentendo a Suricata di identificare eventuali attività dannose o sospette. Queste regole possono essere aggiornate o modificate per riflettere l'ambiente in evoluzione delle minacce informatiche e possono identificare un'ampia gamma di pericoli, dai malware ai comportamenti sospetti. Ogni regola è formata da componenti fisse che devono essere sempre inserite e determinano la struttura base di una regola. Tuttavia, per renderla più specifica e adattarla a particolari esigenze di analisi del traffico, si possono aggiungere ulteriori opzioni. Queste opzioni possono specificare contenuti o pattern da cercare nel traffico, riferimenti esterni, limitazioni sulla profondità e l'offset della ricerca nel payload del pacchetto, classificazioni di tipi di attività e molte altre condizioni. Insieme, queste componenti fisse e opzionali permettono a Suricata di identificare e rispondere in modo preciso e flessibile a una vasta gamma di eventi sospetti o malevoli nel traffico di rete.



Una tipica regola Suricata contiene:

```
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET
TROJAN Likely Bot
Nick in IRC (USA +..)"; flow:established,to_server;
flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK
.*USA.*[0-9]{3,}/i"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvswweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```



Figura 2.1: Struttura di una regola Suricata [3]

- **Azione (in rosso):** Quando il traffico di rete soddisfa le condizioni di una regola, le azioni delle regole determinano la reazione di Suricata. Le più comuni sono "**alert**", che segnala un'attività potenzialmente dannosa; "**drop**", che blocca e scarta silenziosamente il pacchetto; "**pass**", che permette al pacchetto di continuare senza ulteriori ispezioni; e "**reject**", che interrompe il traffico e lo notifica al mittente. Il comportamento di Suricata è definito essenzialmente dalle sue attività in risposta a particolari modelli di traffico o minacce.
- **Header (in blu):** Gli header specificano i criteri essenziali per l'esame del traffico, tra questi abbiamo:
  - PROTOCOLLO: descrive il protocollo di rete a cui si applica la regola. **tcp**, **udp**, **icmp** e **ip** sono alcuni esempi.

- IP E PORTA DI ORIGINE: indica la provenienza del traffico che la regola dovrà valutare. Si può usare un indirizzo IP specifico, un intervallo o la parola *any* per indicare tutte le fonti. La porta di origine (se pertinente) può essere una singola porta, un intervallo di porte o una qualsiasi.
  - DIREZIONE: indica la direzione del flusso di traffico che la regola prenderà in considerazione. I simboli **->** (**dalla sorgente alla destinazione**) e **<>** (**bidirezionale o in qualsiasi direzione**) sono spesso utilizzati come indicatori di direzione.
  - IP E PORTA DI DESTINAZIONE: analogamente alla sorgente, l'indirizzo IP e la porta di destinazione identificano il destinatario del traffico. Può essere un IP particolare, un intervallo o la parola chiave **any**. Lo stesso vale per la porta di destinazione.
- **Opzioni (in verde):** È possibile utilizzare le opzioni e delle keywords all'interno delle parentesi di una regola Suricata per eseguire il match su sezioni particolari di un pacchetto, categorizzare una regola o registrare messaggi personalizzati. Tra le opzioni di una regola di Suricata è richiesto il **punto e virgola (;)** e in genere hanno una struttura **chiave:valore**.

Le opzioni nelle regole di Suricata descrivono i criteri precisi per la corrispondenza del traffico e offrono ulteriori metadati relativi alla regola. Ad esempio, **"msg"** fornisce un messaggio di avviso dettagliato, **"sid"** assegna alla regola un ID univoco e **"rev"** visualizza il numero di revisione della regola. Insieme, queste alternative migliorano gli standard di rilevamento e garantiscono un'identificazione, un monitoraggio e un aggiornamento accurato delle regole.

### 2.1.6 Files di Log

I file di log di Suricata sono i luoghi in cui Suricata memorizza i dettagli sul traffico di rete che ispeziona e qualsiasi evento o anomalia che viene rilevata in base al suo set di regole predefinite. Questi log forniscono informazioni sulle attività del sistema, sui rischi potenziali e sull'attività della rete. Sono uno strumento essenziale per il monitoraggio, la diagnosi e la risposta a possibili eventi di sicurezza per analisti della sicurezza, amministratori di sistema e altri professionisti IT.

Suricata offre 2 file di log principali, ovvero **fast.log** e **eve.json**, entrambi localizzabili nella cartella `/var/log/suricata/`.

Il file **fast.log** è progettato per la registrazione rapida degli alert e fornisce una chiara cronologia degli allarmi del sistema. Ogni alert registrato in questo registro contiene informazioni importanti, tra cui il timestamp, l'ID della regola e gli indirizzi IP e le porte pertinenti, tutti visualizzati in modo da consentire un rapido esame. Per gli esperti che cercano una rapida panoramica dei pericoli scoperti senza dover passare al setaccio informazioni più dettagliate, questo file è indispensabile.

D'altro canto il file **eve.json** presenta un quadro più completo dell'attività di rete. Registra un'ampia gamma di dati in un formato JSON strutturato, dagli alert e dagli header HTTP a informazioni più complesse come gli handshake TLS, le interazioni SSH e i trasferimenti di file. Grazie al suo stile leggibile e all'abbondanza di contenuti, eve.json è particolarmente adatto per studi approfonditi. In sostanza, eve.json fornisce una panoramica completa delle interazioni di rete e dei potenziali problemi di sicurezza, mentre fast.log fornisce un'istantanea dei pericoli attuali.

## 2.1.7 Integrazione con LUA

### 2.1.7.1 Breve introduzione

Lua è un linguaggio di scripting leggero, di alto livello e incorporabile, che si è ritagliato un posto di rilievo nel mondo della programmazione. Il suo nome, che in portoghese significa "luna", proviene dal Brasile dei primi anni '90 e fa riferimento alla sua semplicità. Lua è stato creato con un'attenzione particolare al minimalismo, offrendo un numero ridotto di funzioni potenti che lo rendono semplice da imparare per i principianti, pur rimanendo adattabile per i professionisti. La sua espressività non viene intaccata dalla sua compattezza, consentendo agli sviluppatori di scrivere codice chiaro per applicazioni robuste. In particolare, Lua è stata una scelta popolare in una varietà di settori, dallo sviluppo di videogiochi ai dispositivi incorporati e alle applicazioni online, grazie alla sua capacità di integrarsi senza sforzo nei programmi, in particolare nei sistemi con limiti di risorse rigorosi.

Grazie alla sua adattabilità e semplicità d'uso, Lua è ampiamente scelto nell'industria dei videogiochi per lo scripting di interfacce utente, comportamenti dell'intelligenza artificiale e logica di gioco. Diversi giochi e piattaforme di gioco ben noti consentono agli utenti di migliorare o sviluppare nuovi sistemi di gioco utilizzando Lua.

### 2.1.7.2 Lua detection

L'integrazione di Lua con Suricata apre le porte alla creazione di regole dinamiche e all'ispezione avanzata del traffico. Grazie a questo linguaggio è possibile creare modelli di rilevamento personalizzati utilizzando degli script, anziché affidarsi solo a set di regole consolidate. Grazie al grado granulare di controllo e personalizzazione offerto da questo sistema, è possibile rispondere a rischi insoliti o di recente sviluppo che potrebbero non essere ancora affrontati dai set di regole stabiliti.

Suricata utilizza **LuaJIT**, un compilatore **Just-In-Time** per Lua, per eseguire rapidamente gli script Lua. Per inizializzare lo script Lua all'interno

di una regola Suricata è possibile utilizzare due keyword con la stessa funzione: **lua** e **lua**. In origine veniva supportato solo Lua, ma in seguito è stato inserito anche il supporto a Lua, ma l'uso di Lua o Lua dipende solo dalle impostazioni in fase di compilazione. E' possibile usare via prompt il comando `ldd <your/suricata/binary>` per sapere a quale libreria Lua è collegato Suricata, così da sapere se usare Lua o Lua.[4]

Lo script Lua finalizzato all'analisi del traffico deve essere formato da due funzioni: **init** e **match**.

La funzione **init** è usata per dichiarare i dati che lo script richiede a Suricata per eseguire la sua analisi.

In questo esempio viene deciso di analizzare il payload di un pacchetto.

```
1 function init (args)
2     local needs = {}
3     needs["payload"] = tostring(true)
4     return needs
5 end
```

Listing 2.5: Funzione Init Lua Detection

La funzione **match** viene invece utilizzata per eseguire l'analisi effettiva sui dati dichiarati nella funzione **init**, ad esempio nel seguente codice viene analizzato il payload di un pacchetto e viene verificato che il valore dopo la stringa **Analisi numero:** sia maggiore di 0. La regola a cui è collegato lo script scatta unicamente in questo caso e restituisce il valore 1, 0 altrimenti.

```
1 function match(args)
2     local a = args["payload"]
3     local x = string.sub(string.match(a,
4                                     "Analisi numero:(.*)
5     ), 1, 3)
6     local x_num = tonumber(x)
7     if x_num > 0 then
8         return 1
9     end
10    return 0
end
```

```
11 return 0
```

Listing 2.6: Funzione Match Lua Detection

### 2.1.7.3 Lua output

Oltre a creare delle regole più complesse e specifiche, Lua può essere utilizzato anche nell'ambito dei file di log.

Come spiegato in precedenza, Suricata fornisce strutture di log predefinite che registrano gli eventi di rete e gli alert in una configurazione tipica. Tuttavia, vari contesti e scenari d'uso possono richiedere diversi formati di logging o l'acquisizione di dati specializzati. Lua interviene per migliorare la situazione. Gli utenti possono scegliere il formato di ogni file di log, scegliere quali campi sono cruciali e persino usare la logica condizionale per controllare con precisione ciò che viene riportato e quando, scrivendo la logica di output del log in Lua. Quando si combinano i log di Suricata con piattaforme o strumenti esterni, questa flessibilità diventa essenziale. Ad esempio, un'azienda può utilizzare un formato di log unico come parte di una pipeline di analisi dei log proprietaria. Questo è reso più facile dallo scripting Lua, che consente di organizzare con precisione i log in modo che corrispondano al formato necessario.

Per creare questi file di log personalizzabili è necessario per prima cosa modificare il file di configurazione **suricata.yaml** in modo da abilitare i log generati da uno script lua. Si va perciò a modificare la stringa **enable: yes** e aggiungendo gli script nel direttorio principale */etc/suricata/lua-output/*. Come nel caso degli script Lua per controllare il traffico, è necessario che vi sia una struttura fissa [5], questa volta composta da 4 funzioni: **init**, **setup**, **log** e **deinit**.

La funzione **init** svolge lo stesso compito del caso Lua detection, **setup** viene generalmente utilizzato per generare il nuovo file di log, **log** si occupa di creare il formato del log, utilizzando le funzioni che Lua mette a disposizione per estrarre informazioni dal traffico di dati [6] ed infine **deinit** formatta lo script una volta terminato.

## 2.2 Modbus

I protocolli di comunicazione sono essenziali nell'automazione industriale in quanto consentono ai dispositivi e ai sistemi di scambiare dati ed eseguire azioni coordinate. Questi protocolli definiscono regole e standard per la trasmissione, la ricezione e l'interpretazione dei dati, assicurando una comunicazione senza interruzioni tra dispositivi di produttori diversi. Stabiliscono un linguaggio e un formato comune per la comunicazione, facilitando l'interoperabilità e l'integrazione in ambienti industriali complessi.

### 2.2.1 Origini e storia

Modbus è stato introdotto nel 1979 da un'azienda chiamata **Modicon**, nome che sta per "**Modular Digital Controller**". Modicon è stato il pioniere del primo **controllore logico programmabile (PLC)** nel 1968. Questa innovazione ha cambiato il panorama dell'automazione industriale, consentendo di sostituire i complessi sistemi logici a relè cablati con soluzioni programmabili.

Con il crescente utilizzo e l'adozione dei PLC, è nata l'esigenza di un protocollo di comunicazione standardizzato per facilitare lo scambio di dati tra i PLC e i vari dispositivi periferici, nonché tra i PLC stessi.

Per rispondere a questa esigenza, Modicon ha sviluppato il protocollo Modbus. Il nome Modbus deriva dalla combinazione di "**Modicon**" e "**Bus**" con quest'ultimo che indica un sistema o una rete di comunicazione. Pertanto, Modbus può essere interpretato come il bus di comunicazione progettato per i dispositivi Modicon.

### 2.2.2 Tipologia di dati

Il protocollo di comunicazione Modbus definisce 4 tipi di dati di scambio: **Discretes Input**, **Coils**, **Input Registers** e **Holding Registers**.

- I **Coils**, noti anche come **Discrete Output**, sono tipicamente utilizzati per le rappresentazioni di **dati binari**, che includono stati come **ON/OFF** o **VERO/FALSO**. Sono spesso collegate alle uscite digitali di un dispositivo e possono essere utilizzate **sia in lettura che in scrittura**.
- I **Discretes Input**, a differenza dei Coils, sono entità di **sola lettura** che spesso rappresentano gli **ingressi digitali** di un dispositivo. Possono essere utilizzati per registrare lo stato di sensori con uscite binarie. Essendo di sola lettura, l'integrità dell'ingresso viene preservata senza interferenze esterne.
- Gli **Input Registers**, oggetti a **2 bytes** che hanno un'interfaccia di **sola lettura**, sono spesso utilizzati per rappresentare i **dati di sensori** che misurano fenomeni fisici come la temperatura, l'umidità o la pressione. Poiché i dati grezzi del sensore possono talvolta essere modificati per rientrare nei limiti di un registro a 2 bytes, l'interpretazione di questi valori può occasionalmente richiedere la conoscenza dell'unità di misura utilizzata.
- Gli **Holding Registers** sono diversi dagli Input Registers in quanto sono entità **sia in lettura che in scrittura**, pur avendo un layout a **2 bytes** e funzionalità simili. Sono utili in situazioni che richiedono operazioni di uscita analogica o la memorizzazione di parametri di impostazione, grazie alla loro duplice natura.



### 2.2.3 Standard usati da Modbus seriale

#### 2.2.3.1 Standard RS232

Lo standard consente a due dispositivi di comunicare in serie asincrona attraverso **due pin RXD e TXD**.

Lo standard seriale minimo utilizza solo due linee RXD e TXD. In questo modo, possiamo utilizzare un cavo incrociato per collegare due dispositivi, **collegando il pin RX del primo dispositivo al pin TX del secondo e viceversa**.

Viene utilizzato un **connettore a vaschetta a 9 poli** con la piedinatura seguente 2.2:

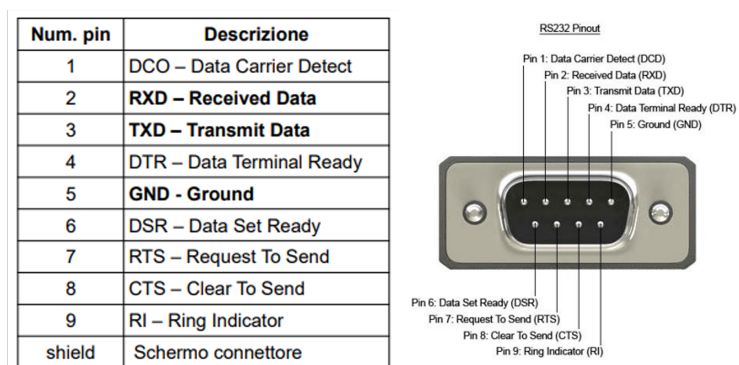


Figura 2.2: Struttura standard RS232

[7]

Il protocollo RS232 è stato sviluppato per consentire la comunicazione tra due dispositivi denominati **DTE (Data Terminal Equipment)** e **DCE (Data Communication Equipment)**. Un esempio di questo è il collegamento tra un **PC** e un **modem seriale**, dove il funzionamento dei pin RX e TX viene **invertito** nel DCE, quindi non è necessario invertire i pin RX e TX. È anche possibile utilizzare anche un **connettore a 25 pin** piuttosto che uno a 9 pin. L'**incrocio dei pin RX e TX dipende** se il collegamento avviene tra un DTE e un DCE o tra due dispositivi dello stesso tipo.

### 2.2.3.2 Standard RS485

La comunicazione RS485 trasmette i dati su un **bus composto da due collegamenti** con un valore di tensione differenziale. Ciò è diverso dalla comunicazione RS232, in cui i dati passano su **due canali distinti RX e TX** con il valore di tensione riferito allo stesso potenziale GND. Come nella RS232, ci sono due canali RX e TX oltre ad altri canali per il controllo del flusso come RTS e CTS. Anche se questo tipo di connettore viene utilizzato tipicamente per collegare due dispositivi punto-punto, il pinout di un connettore a vaschetta a 9 poli è ancora standard. Il pin 4 DI riceve il segnale digitale TX e il pin 1 R0 riceve il segnale RX. I **pin RE e DE** consentono di scegliere la direzione del dato e la scelta tra scrivere o leggere. Il vantaggio principale della comunicazione su bus RS485 è che vengono utilizzati **canali trasmissivi differenziali**, il che significa che è meno vulnerabile ai disturbi. Inoltre, questo consente di raggiungere velocità fino a **20mbps a distanze fino a 1200 metri** rispetto alla RS232. Inoltre, la comunicazione RS485 consente il **collegamento di più dispositivi** piuttosto che solo di un DTE-DCE, come accade con la comunicazione RS232. Di conseguenza, lo standard RS485 offre una struttura hardware affidabile su cui appoggiarsi nel caso di Modbus.

### 2.2.4 Versioni del protocollo

Esistono due versioni del protocollo Modbus, nel caso vengano utilizzati dei **connettori RS485 o RS232** si parla di **Modbus seriale**, a sua volta diviso in **Modbus RTU** e **Modbus ASCII**, invece su **Ethernet/Wi-Fi** viene definito **Modbus TCP/IP**.

### 2.2.4.1 Modbus ASCII

Nella modalità Modbus ASCII, i dati vengono inviati come caratteri ASCII. In particolare, un byte di dati a 8 bit viene suddiviso in due nibble a 4 bit, ciascuno dei quali è rappresentato da un carattere ASCII, 0-9 o A-F. In questo formato, all'inizio e alla fine di ogni messaggio vengono utilizzati rispettivamente **i due punti (":")** e **i caratteri di ritorno a capo e avanzamento riga (CR+LF)**.

L'**indirizzo del dispositivo** (slave) a cui è destinato il messaggio è contenuto nel campo dell'indirizzo, che occupa 1 byte. Una singola rete può supportare **247 dispositivi** con indirizzi compresi tra 1 e 247.

Il campo che definisce la funzione per ASCII contiene due caratteri. Nel momento in cui un master invia un messaggio ad uno slave, il campo funzione indica l'azione che lo slave deve svolgere, come leggere, scrivere, caricare, verificare ecc. Anche in questo caso, lo slave inserisce lo stesso codice funzione richiesto dal master nella sua risposta.

				Function Codes			
				code	Sub code	(hex)	Section
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	02		02	6.2
		Internal Bits Or Physical coils	Read Coils	01		01	6.1
			Write Single Coil	05		05	6.5
			Write Multiple Coils	15		0F	6.11
	16 bits access	Physical Input Registers	Read Input Register	04		04	6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03	6.3
			Write Single Register	06		06	6.6
			Write Multiple Registers	16		10	6.12
			Read/Write Multiple Registers	23		17	6.17
			Mask Write Register	22		16	6.16
			Read FIFO queue	24		18	6.18
			File record access	Read File record	20		14
	Write File record	21			15	6.15	
	Diagnostics		Read Exception status	07		07	6.7
			Diagnostic	08	00-18,20	08	6.8
			Get Com event counter	11		0B	6.9
			Get Com Event Log	12		0C	6.10
Report Slave ID			17		11	6.13	
Read device Identification			43	14	2B	6.21	
Other		Encapsulated Interface Transport	43	13,14	2B	6.19	

Figura 2.3: Elenco funzioni Modbus [8]

I **codici funzione Modbus** sono comandi standardizzati utilizzati per eseguire operazioni specifiche all'interno del protocollo. Questi codici funzione consentono ai dispositivi Modbus di leggere o scrivere dati, controllare il comportamento del dispositivo ed eseguire attività diagnostiche. Alcuni codici funzione comunemente utilizzati sono Read Coils (0x01), Read Input Registers (0x04), Write Single Coil (0x05) e Write Multiple Registers (0x10). 2.3 Ogni codice funzione ha uno scopo specifico ed è supportato dai dispositivi Modbus per una comunicazione efficace.

Il **Longitudinal Redundancy Check (LRC)** è una tecnica utilizzata per il controllo degli errori. Il valore LRC viene calcolato per il messaggio specificato e poi trasmesso alla fine del messaggio come stringa ASCII. La leggibilità del Modbus ASCII è una delle sue qualità principali, infatti i dati sono leggibili dall'uomo proprio perchè sono in formato ASCII.

Start	Address	Function	Data	LRC	End
:	2 Chars	2 Chars	N Chars	2 Chars	CR LF

Tabella 2.1: Frame Modbus ASCII

#### 2.2.4.2 Modbus RTU

I dati vengono trasferiti in forma binaria quando si utilizza la modalità RTU (Remote Terminal Unit). Rispetto all'ASCII, il formato binario è più efficiente dal punto di vista dello spazio e quindi è l'opzione più diffusa. Per quanto riguarda i delimitatori dei messaggi, i messaggi Modbus RTU sono unici e **non hanno caratteri di inizio e fine definiti**, a differenza di Modbus ASCII. In media, l'inizio di un messaggio è segnato da un silenzio che dura almeno **3,5 volte più di un singolo carattere**. Anche la conclusione del messaggio è segnalata da un breve intervallo di silenzio.

Il **controllo di ridondanza ciclica (CRC)**, più affidabile del controllo di

ridondanza longitudinale (LRC) utilizzato in Modbus ASCII, viene utilizzato in Modbus RTU per il controllo degli errori. Il processo CRC consiste nel calcolare un valore di 2 byte in base al contenuto del messaggio e aggiungerlo alla fine del messaggio. I messaggi Modbus RTU non sono facilmente comprensibili dall'uomo a causa della natura binaria del formato, a differenza del formato Modbus ASCII. Sono necessari strumenti o software specifici per l'interpretazione dei dati.

Start	Address	Function	Data	CRC	End
3.5 Char Time	1 Byte	1 Byte	$N * 1 \text{ Byte}$	2 Bytes	3.5 Char Time

Tabella 2.2: Frame Modbus RTU

### 2.2.4.3 Modbus TCP/IP

Poichè il Modbus TCP/IP è stato creato principalmente per la comunicazione basata su Ethernet, può funzionare senza problemi sulle reti che utilizzano lo stack di protocollo TCP/IP. Grazie a questa versatilità, i dispositivi possono connettersi facilmente tra loro attraverso le reti locali e persino a livello globale tramite Internet. L'**intestazione MBAP (Modbus Application Protocol)**, un elemento speciale della struttura dei messaggi Modbus TCP/IP, è fondamentale per gestire e instradare efficacemente i dati in un ambiente Ethernet. Per abbinare un messaggio di richiesta al messaggio di risposta associato, utilizzare l'identificatore di transazione. L'**ID del protocollo** viene sempre settato a 0, per indicare che Modbus è il protocollo usato. Il **Length Identifier** esprime il numero di byte rimanenti nel frame, escludendo l'header MBAP. Infine l'**Unit Identifier** contiene l'indirizzo del dispositivo al quale il messaggio è rivolto, la stessa funzione che l'Address svolge in Modbus RTU. [9] La parte successiva all'intestazione MBAP, ovvero il codice della funzione e i dati, rimangono simili ai casi di Modbus RTU e Modbus ASCII. Modbus TCP/IP si differenzia dai suoi fratelli seriali poichè non si affida a tecniche come il CRC per il controllo degli errori, bensì utilizza

le funzioni di controllo degli errori integrate nel protocollo TCP, controllando la consegna precisa dei pacchetti di dati, coordinando la ritrasmissione dei pacchetti persi e organizzando i pacchetti di dati in modo da preservarne l'ordine.

Transaction ID	Protocol ID	Length	Unit ID	Function	Data
2 Bytes	2 Bytes	2 Bytes	1 Byte	1 Byte	Variable

Tabella 2.3: Frame Modbus TCP/IP

Un'altra caratteristica del Modbus TCP/IP è la sua **struttura a pacchetti**, che deriva dalle sue origini Ethernet. Con questa struttura, ogni comunicazione è contenuta in un pacchetto di rete separato. Di conseguenza, questi formati di pacchetti delineano naturalmente l'inizio e la fine dei messaggi, **eliminando la necessità di delimitatori separati**, al contrario di Modbus RTU e ASCII. Modbus TCP/IP interagisce principalmente attraverso la **porta 502**, un dettaglio tecnico importante per chi configura dispositivi di rete rilevanti come firewall e router. Il protocollo può operare su grandi distanze perchè dipende dallo stack TCP/IP. Inoltre, consentendo una perfetta integrazione con varie apparecchiature di rete, come router e switch, può creare connessioni tra piccole reti locali e vaste reti internazionali.



## Capitolo 3

# Analisi progettuale ed Implementazione

L'adozione dell'**Industrial Internet of Things (IIoT)** ha portato una rivoluzione tecnologica senza precedenti nel panorama industriale. L'**Internet of Things** ha permesso alle industrie di sviluppare nuovi modelli di business, ottimizzazione delle risorse e processi produttivi più avanzati. Tuttavia, questo nuovo ambiente interconnesso ha portato con sé importanti problemi di sicurezza informatica.

La quantità di dati generata, trasmessa e analizzata è enorme in un mondo in cui ogni dispositivo, dal più piccolo sensore al più grande macchinario, è collegato alla rete. Queste informazioni, che spesso vengono inviate in tempo reale, sono essenziali per la gestione e l'ottimizzazione dei processi industriali, ma il loro valore li rende interessanti per i criminali.

L'Internet of Things (IIoT) presenta una nuova prospettiva per l'industria. Ogni nodo di questa rete, che sia un sensore, una macchina o un database, svolge un ruolo specifico mentre i dati fluiscono incessantemente attraverso essa. Sebbene abbia portato ad opportunità impreviste, questa stretta connessione ha anche portato a nuovi ostacoli. La sicurezza di un sistema così complesso non può più essere una considerazione aggiuntiva o una considerazione successiva. deve essere integrato fin dall'inizio, progettato in ogni



singolo elemento e in ogni connessione tra di loro.

Irregolarità o fluttuazioni inaspettate nei dati, come variazioni rapide o temperature fuori norma, possono indicare tentativi di intrusione, sabotaggio o attacchi mirati piuttosto che anomalie o guasti. Ad esempio, un **malware** che è stato progettato per sovraccaricare il sistema e causare danni fisici o interruzioni nella produzione potrebbe causare un aumento anomalo della temperatura in un macchinario.

Nel campo dell'Internet of Things (IIoT), il tempo è una risorsa vitale. Ogni azione, dalla rilevazione alla decisione, deve avvenire rapidamente. Il sistema deve essere in grado di reagire prontamente e in modo appropriato quando un sensore rileva un'anomalia nell'impianto di produzione. Il costo del ritardo o dell'inazione può andare oltre le perdite finanziarie, includendo danni irreversibili alle attrezzature o, peggio, rischi per la sicurezza umana.

Inoltre, osservare e gestire manualmente ogni singolo dispositivo è difficile a causa della vastità e della complessità delle reti IIoT. Sistemi di sicurezza sofisticati con capacità di apprendimento automatico e intelligenza artificiale sono necessari per analizzare e rispondere rapidamente a queste fluttuazioni. Queste soluzioni sono essenziali per identificare rapidamente comportamenti sospetti, isolare i dispositivi compromessi e proteggere l'intera infrastruttura. Il panorama tecnologico si diversifica mentre l'Internet of Things (IIoT) si espande. La difficoltà consiste nel far funzionare insieme sistemi e dispositivi che non sono stati progettati per essere compatibili. In questo momento, soluzioni come Suricata diventano essenziali in quanto non solo servono da ponte tra diversi dispositivi, ma assicurano anche la sicurezza, consentendo una comunicazione sicura e protetta tra diversi dispositivi.

Anche se non si tratta di attacchi mirati, c'è la possibilità di malfunzionamenti o incidenti imprevisti causati da configurazioni errate, aggiornamenti software incompatibili o altre vulnerabilità hardware o software esistenti.

La sfera della sicurezza informatica è in continuo cambiamento. Ogni giorno sorgono nuove minacce e nuove difese sono necessarie per affrontarle e la capacità di adattarsi rapidamente è essenziale in un ambiente in rapido

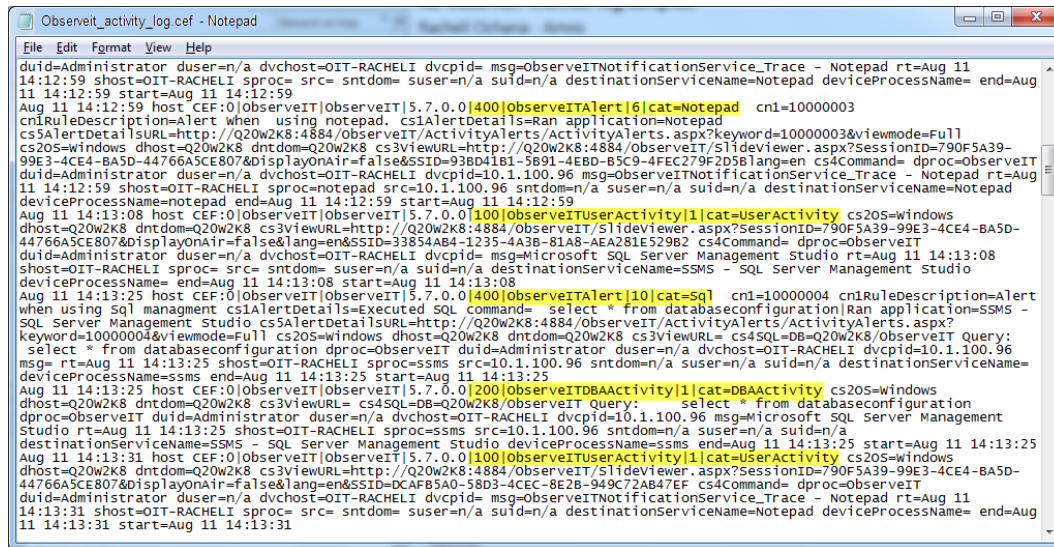
cambiamento come l'Internet of Things.

La combinazione di Suricata e Lua fornisce un framework agile e altamente personalizzabile che consente alle organizzazioni non solo di rispondere alle minacce attuali, ma anche di anticipare e prepararsi per le future.

Grazie a questa premessa, è possibile capire quanto sia necessario controllare meticolosamente il traffico e per fare questo la scelta migliore ricade su Suricata. In ambienti come l'Internet of Things (IIoT), dove le minacce possono essere tanto variegate quanto elusive, è fondamentale avere uno strumento che può essere modificato e adattato per rilevare specifiche anomalie.

Utilizzando Lua insieme a Suricata, è possibile sviluppare e distribuire script specifici per la generazione di log. Questi script possono essere progettati per acquisire informazioni specifiche, formattarle in modo univoco e persino classificarle in base a una varietà di criteri. Questo livello di personalizzazione va oltre la semplice registrazione degli eventi, trasformando i log in un potente strumento di analisi. Ad esempio, considerando un ambiente IIoT, potremmo voler monitorare in modo mirato le variazioni di temperatura dei macchinari. Con Lua è possibile creare log personalizzati che registrano solo i cambiamenti di temperatura al di fuori di altri parametri, collegando tali cambiamenti a eventi o fattori sospetti. Ciò non solo riduce il "rumore" nei log rimuovendo i dati non necessari, ma velocizza anche la diagnosi di problemi o anomalie.

Grazie alla sua capacità di generare log dettagliati e compatibili con standard come il **Common Event Format (CEF)** 3.1, Suricata non solo facilita l'integrazione con altre soluzioni di sicurezza e analisi, ma aiuta anche gli amministratori di sistema a comprendere la natura e l'origine degli attacchi, permettendo una risposta e una prevenzione ancor più mirate.



```

duid=Administrator duser=n/a dvchost=OIT-RACHELI dvcpid= msg=observeitNotificationService_Trace - Notepad rt=Aug 11
14:12:59 shost=OIT-RACHELI sproc= src= sntdom= suser=n/a suid=n/a destinationServiceName=Notepad deviceProcessName= end=Aug
11 14:12:59 start=Aug 11 14:12:59
Aug 11 14:12:59 host CEF:0|ObserveIT|ObserveIT|5.7.0.0|400|ObserveITAlert|6|cat=Notepad cn1=10000003
cn1RuleDescription=Alert when using notepad. cs1AlertDetails=Run application=Notepad
cs5AlertDetailsURL=http://Q20w2K8:4884/ObserveIT/ActivityAlerts/ActivityAlerts.aspx?keyword=10000003&viewmode=Full
cs205=windows dhost=Q20w2K8 dntdom=Q20w2K8 cs3viewURL=http://Q20w2K8:4884/ObserveIT/Slideviewer.aspx?SessionID=790F5A39-
99E3-4CE4-BA5D-44766A5CE807&DisplayOnAir=false&SSID=938D41B1-5891-4EBD-B5C9-4FEC279F2D58lang=en cs4Command= dproc=ObserveIT
duid=Administrator duser=n/a dvchost=OIT-RACHELI dvcpid=10.1.100.96 msg=observeitNotificationService_Trace - Notepad rt=Aug
11 14:12:59 shost=OIT-RACHELI sproc=notepad src=10.1.100.96 sntdom=n/a suser=n/a suid=n/a destinationServiceName=Notepad
deviceProcessName=notepad end=Aug 11 14:12:59 start=Aug 11 14:12:59
Aug 11 14:13:08 host CEF:0|ObserveIT|ObserveIT|5.7.0.0|100|ObserveITUserActivity|1|cat=UserActivity cs205=windows
dhost=Q20w2K8 dntdom=Q20w2K8 cs3viewURL=http://Q20w2K8:4884/ObserveIT/Slideviewer.aspx?SessionID=790F5A39-99E3-4CE4-BA5D-
44766A5CE807&DisplayOnAir=false&SSID=33854AB4-1235-4A3B-81A8-AEA281E52982 cs4Command= dproc=ObserveIT
duid=Administrator duser=n/a dvchost=OIT-RACHELI dvcpid= msg=Microsoft SQL Server Management Studio rt=Aug 11 14:13:08
shost=OIT-RACHELI sproc= src= sntdom= suser=n/a suid=n/a destinationServiceName=SSMS - SQL Server Management Studio
deviceProcessName= end=Aug 11 14:13:08 start=Aug 11 14:13:08
Aug 11 14:13:25 host CEF:0|ObserveIT|ObserveIT|5.7.0.0|200|ObserveITDBAActivity|1|cat=DBAActivity cn1=10000004 cn1RuleDescription=Alert
when using sql management cs1AlertDetails=Executed SQL command= select * from databaseconfiguration|Run application=SSMS -
SQL Server Management Studio cs5AlertDetailsURL=http://Q20w2K8:4884/ObserveIT/ActivityAlerts/ActivityAlerts.aspx?
keyword=10000004&viewmode=Full cs205=windows dhost=Q20w2K8 dntdom=Q20w2K8 cs3viewURL= cs4SQL=DB-Q20w2K8/ObserveIT Query:
select * from databaseconfiguration dproc=ObserveIT duid=Administrator duser=n/a dvchost=OIT-RACHELI dvcpid=10.1.100.96
msg= rt=Aug 11 14:13:25 shost=OIT-RACHELI sproc=ssms src=10.1.100.96 sntdom=n/a suser=n/a suid=n/a destinationServiceName=
deviceProcessName=ssms end=Aug 11 14:13:25 start=Aug 11 14:13:25
Aug 11 14:13:25 host CEF:0|ObserveIT|ObserveIT|5.7.0.0|200|ObserveITDBAActivity|1|cat=DBAActivity cn1=10000004 cn1RuleDescription=Alert
when using sql management cs1AlertDetails=Executed SQL command= select * from databaseconfiguration|Run application=SSMS -
SQL Server Management Studio cs5AlertDetailsURL=http://Q20w2K8:4884/ObserveIT/ActivityAlerts/ActivityAlerts.aspx?
keyword=10000004&viewmode=Full cs205=windows dhost=Q20w2K8 dntdom=Q20w2K8 cs3viewURL= cs4SQL=DB-Q20w2K8/ObserveIT Query:
select * from databaseconfiguration dproc=ObserveIT duid=Administrator duser=n/a dvchost=OIT-RACHELI dvcpid=10.1.100.96
msg= rt=Aug 11 14:13:25 shost=OIT-RACHELI sproc=ssms src=10.1.100.96 sntdom=n/a suser=n/a suid=n/a destinationServiceName=
deviceProcessName=ssms end=Aug 11 14:13:25 start=Aug 11 14:13:25
Aug 11 14:13:31 host CEF:0|ObserveIT|ObserveIT|5.7.0.0|100|ObserveITUserActivity|1|cat=UserActivity cs205=windows
dhost=Q20w2K8 dntdom=Q20w2K8 cs3viewURL=http://Q20w2K8:4884/ObserveIT/Slideviewer.aspx?SessionID=790F5A39-99E3-4CE4-BA5D-
44766A5CE807&DisplayOnAir=false&SSID=DCAF85A0-58D3-4CEC-8E2B-949C72AB47EF cs4Command= dproc=ObserveIT
duid=Administrator duser=n/a dvchost=OIT-RACHELI dvcpid= msg=observeitNotificationService_Trace - Notepad rt=Aug 11
14:13:31 shost=OIT-RACHELI sproc= src= sntdom= suser=n/a suid=n/a destinationServiceName=Notepad deviceProcessName= end=Aug
11 14:13:31 start=Aug 11 14:13:31

```

Figura 3.1: Esempio Format CEF [10]

In questa tesi verrà simulata una comunicazione tra un **server (master)** e **client (slave)** Modbus dove lo slave invierà dati diagnostici al master come **temperatura, uso della CPU e pressione**. Nel frattempo il suddetto payload verrà analizzato per controllare che non vi siano comportamenti sospetti e, in caso di anomalia, avvertire l'utente.

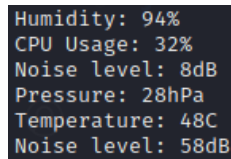
Per aumentare l'espressività dei file di log e plasmarli attorno alle necessità di questa tesi, sarà necessario utilizzare sempre l'integrazione che Suricata offre verso il linguaggio di programmazione **Lua** ed apprendere le funzioni che quest'ultimo offre per fare ciò.

Il primo passo di questo progetto è quello di creare una simulazione di un'infrastruttura del protocollo Modbus. Sarà quindi necessario creare un **client (slave)** che invia dati diagnostici ad un **server (master)**. Per rendere il tutto più verosimile, i dati comunicati saranno informazioni riguardanti temperatura, tempo di utilizzo o pressione riguardanti un certo macchinario industriale. Tramite Suricata poi, si andrà a posizionarsi nel mezzo di master/slave ed analizzare il traffico alla ricerca di anomalie significative che potrebbero essere sinonimo di malfunzionamenti o di attacchi mirati.

Per realizzare questo primo passo la scelta è ricaduta sulla libreria per il linguaggio Python chiamata **pyModbus** [11]. Nonostante ciò esistono una serie di opzioni quando si considera come il protocollo Modbus viene implementato in diversi linguaggi informatici. **Libmodbus** è un'opzione importante per gli ambienti che utilizzano C o C++. **J2Mod** si distingue nei contesti Java. Opzioni come **node-modbus** e **jsmodbus** possono essere utilizzate rispettivamente con NodeJs e Javascript. **EasyModbusTCP** e **NModbus4** ricevono spesso attenzione nella comunità .NET. **Gomodbus** è una possibile alternativa per le soluzioni incentrate sul linguaggio Go e **tokio-modbus** è suggerito per le implementazioni in Rust. In questo caso la scelta è ricaduta su **pymodbus**. Questa scelta è data dalla sua facilità d'uso e dal suo vasto supporto della community, ciò lo rende affidabile grazie agli aggiornamenti regolari e al sostegno completo di Modbus. Grazie alla sua adattabilità, può essere utilizzato su diverse piattaforme e l'ampio ecosistema di Python rende semplici le integrazioni. Un altro fattore determinante riguardo questa scelta è la curiosità di approfondire il linguaggio Python, in quanto mai affrontato in precedenza. È necessario sottolineare inoltre che la documentazione fornita dal team di **pymodbus** [11] è estremamente esaustiva e durante lo svolgimento del progetto è stata di grande aiuto per chiarire dubbi e risolvere problemi.

Il punto di partenza è stato la pagina che la documentazione offre inerente a degli esempi con diverse funzionalità, come la comunicazione sincrona e asincrona, il client e il server, i dispositivi seriali e TCP/IP, e altro.

In questo caso specifico verranno utilizzati come punti di partenza gli esempi Modbus payload Server/Client. Ciò è stato fatto in quanto l'oggetto scelto per la comunicazione è per l'appunto un payload, contenente un insieme di informazioni diagnostiche mischiate con stringhe casuali.



```
Humidity: 94%  
CPU Usage: 32%  
Noise level: 8dB  
Pressure: 28hPa  
Temperature: 48C  
Noise level: 58dB
```

Figura 3.2: Oggetto della comunicazione server/client

Nell'immagine precedente 3.2 tutte le stringhe rappresentano dati diagnostici, ma nella realtà non è detto che tutto il payload sia composto da tali, perciò con una frequenza a scelta verranno generate delle **stringhe casuali a lunghezza compresa tra 5 e 15 caratteri**. Per fare ciò verrà usata una funzione che prende in input un vettore contenente le stringhe sensibili e restituisce un mix tra stringhe sensate e casuali.

```
1 def random_payload(payloads):  
2     length = random.randint(5, 15)  
3     random_string = ''.join(random.choices(string.  
4         ascii_lowercase, k=length))  
5     if random.random() < 0.5:  
6         random_payload = random.choice(payloads)  
7     return random_payload  
8     return random_string
```

Listing 3.1: Funzione genera payload

Nel codice il valore 0.5 dopo `random.random()` indica la probabilità con cui le stringhe diagnostiche compariranno all'interno del payload. Impostando il valore 0 verranno generate solo stringhe casuali, al contrario con 0.5 ci sarà una probabilità del 50% tra stringhe sensibili e non.

```
payloads = [  
    "CPU Usage: " + str(random.randint(0,100)) + "%",  
    "Temperature: " + str(random.randint(0,100)) + "C",  
    "Humidity: " + str(random.randint(0,100)) + "%",  
    "Pressure: " + str(random.randint(0,100)) + "hPa",  
    "Noise level: " + str(random.randint(0,100)) + "dB",  
    "Air Quality Index: " + str(random.randint(0,100))  
]
```

Figura 3.3: Stringhe diagnostiche

I valori inerenti alle stringhe diagnostiche 3.3 saranno generati casualmente e per comodità potranno tutti assumere valori compresi tra 0 e 100.

Ovviamente prima di iniziare sarà necessario installare Python sulle macchine coinvolte, dopodichè, ponendo la dicitatura `python3` davanti allo script, sarà possibile avviarlo (Il 3 sta per la versione installata di Python, la 3.9.2).

Il server deve essere avviato prima del client tramite il comando:

```
python3 server_payloads.py -c tcp -p 502,
```

mentre il client viene lanciato con:

```
python3 client_payloads.py -c tcp -p 502 -host 192.168.1.61.
```

La voce **-c** indica il **protocollo da utilizzare**, **-p** e **-host** indicano rispettivamente la **porta** e l'**indirizzo IP** dove insiste il master.

Ora è utile osservare cosa succede nel traffico dei pacchetti quando il client invia dati al server.

Per fare ciò si ricorrerà a **Wireshark** [12], un software **open-source packet-sniffer** utile per monitorare la rete.

No.	Time	Source	Destination	Protocol	Length	Info
5693	4.745260948	192.168.1.201	192.168.1.201	Modbus...	80	Response: Trans: 797; Unit: 1, Fu
5694	4.745312119	127.0.0.1	127.0.0.1	TCP	104	2222 → 60520 [PSH, ACK] Seq=109173 Ac
5695	4.745724943	127.0.0.1	127.0.0.1	TCP	104	2222 → 60520 [PSH, ACK] Seq=109209 Ac
5696	4.745740497	192.168.1.201	192.168.1.201	Modbus...	80	Query: Trans: 798; Unit: 1, Fu
5697	4.745741005	127.0.0.1	127.0.0.1	TCP	68	60520 → 2222 [ACK] Seq=325 Ack=109245
5698	4.745800750	192.168.1.201	192.168.1.201	Modbus...	193	Response: Trans: 798; Unit: 1, Fu
5699	4.746238789	127.0.0.1	127.0.0.1	TCP	104	2222 → 60520 [PSH, ACK] Seq=109245 Ac
5700	4.746314655	127.0.0.1	127.0.0.1	TCP	104	2222 → 60520 [PSH, ACK] Seq=109281 Ac
5701	4.746329851	127.0.0.1	127.0.0.1	TCP	68	60520 → 2222 [ACK] Seq=325 Ack=109317
5702	4.746461258	127.0.0.1	127.0.0.1	TCP	128	2222 → 60520 [PSH, ACK] Seq=109317 Ac
5703	4.746627811	127.0.0.1	127.0.0.1	TCP	112	2222 → 60520 [PSH, ACK] Seq=109377 Ac
5704	4.746641540	127.0.0.1	127.0.0.1	TCP	68	60520 → 2222 [ACK] Seq=325 Ack=109421
5705	4.746676213	192.168.1.201	192.168.1.201	Modbus...	211	Query: Trans: 799; Unit: 1, Fu
5706	4.746741088	192.168.1.201	192.168.1.201	Modbus...	80	Response: Trans: 799; Unit: 1, Fu
5707	4.746855330	127.0.0.1	127.0.0.1	TCP	104	2222 → 60520 [PSH, ACK] Seq=109421 Ac
5708	4.747141389	127.0.0.1	127.0.0.1	TCP	104	2222 → 60520 [PSH, ACK] Seq=109457 Ac
5709	4.747155800	127.0.0.1	127.0.0.1	TCP	68	60520 → 2222 [ACK] Seq=325 Ack=109493
5710	4.747157324	192.168.1.201	192.168.1.201	Modbus...	80	Query: Trans: 800; Unit: 1, Fu
5711	4.747218413	192.168.1.201	192.168.1.201	Modbus...	207	Response: Trans: 800; Unit: 1, Fu
5712	4.747684149	127.0.0.1	127.0.0.1	TCP	104	2222 → 60520 [PSH, ACK] Seq=109493 Ac
5713	4.747834325	127.0.0.1	127.0.0.1	TCP	104	2222 → 60520 [PSH, ACK] Seq=109529 Ac
5714	4.747849052	127.0.0.1	127.0.0.1	TCP	68	60520 → 2222 [ACK] Seq=325 Ack=109565
5715	4.747973462	127.0.0.1	127.0.0.1	TCP	128	2222 → 60520 [PSH, ACK] Seq=109565 Ac

Register 39 (UINT16): 2627	0000	00 00 03 04 00 06	00 00 00 00 00 26 21 08 00	.....&!
Register 40 (UINT16): 20565	0010	45 00 00 b1 74 bd 40 00	40 06 40 a7 c0 a8 01 c9	E...t @ @ @
Register 41 (UINT16): 8277	0020	c0 a8 01 c9 01 f6 e8 d0	5b 88 0f ad 53 bd 1b 55	.....[ o S U
Register 42 (UINT16): 29537	0030	80 18 02 00 85 86 00 00	01 01 08 0a c2 a9 4f 90	.....0
Register 43 (UINT16): 26469	0040	c2 a9 4f 90 03 1e 00 00	00 77 01 03 74 0a 41 09	...0.....w t A l
Register 44 (UINT16): 14880	0050	72 20 51 75 61 6c 69 74	79 20 49 6e 64 65 78 3a	r Qualit y Index:
Register 45 (UINT16): 12857	0060	20 36 34 0a 48 75 6d 69	64 69 74 79 3a 20 30 33	64 Humi dity: 63
Register 46 (UINT16): 9482	0070	25 0a 48 75 6d 69 64 69	74 79 3a 20 38 34 25 0a	% Humi dity: 84%
Register 47 (UINT16): 20594	0080	4c 6f 69 73 65 20 6c 65	76 65 6c 3a 20 31 37 64	Noise le vel: 17d
Register 48 (UINT16): 25971	0090	42 0a 43 50 55 20 55 73	61 67 65 3a 20 32 39 25	B CPU Us age: 29%
Register 49 (UINT16): 29557	00a0	0a 50 72 65 73 75 72	65 3a 20 38 38 68 50 61	Pressur e: 88hPa
Register 50 (UINT16): 29285	00b0	0a 43 50 55 20 55 73 61	67 65 3a 20 37 35 25 0a	CPU Usa ge: 75%
Register 51 (UINT16): 14880	00c0	00		

Figura 3.4: Vista traffico con Wireshark

Come si evince dall'immagine precedente 3.4, il payload viene distribuito di volta in volta all'interno dei registri, i quali hanno una capacità massima di 2 bytes. Esaurito lo spazio in un determinato registro, la parte rimanente della stringa verrà scritta in un altro e così via fino alla fine dei dati. Durante i vari test effettuati è sorto subito un problema riguardante la quantità di registri su cui è possibile leggere e/o scrivere contemporaneamente tramite i metodi della libreria pyModbus, `write_registers` e `read_holding_registers`. Secondo la documentazione ufficiale Modbus, è possibile **scrivere su 123 registri e leggerne 125 per ogni richiesta** [13], ciò rende impossibile trattare più di 8-9 stringhe contemporaneamente.

Nonostante questo, è possibile inserire un **ciclo for** in modo da riuscire a simulare un sample di dati potenzialmente infinito, anche se il massimo numero di righe è sempre limitato.

A questo punto è necessario apportare delle modifiche al file `suricata.yaml` in modo da rendere l'IDS compatibile a pieno con le operazioni che verranno effettuate successivamente:

- Raggiungere la sezione dedicata all'interfaccia di rete ed impostare **eth1**.

```
1  af-packet :  
2    - interface: eth1
```

Listing 3.2: `suricata.yaml`: Interfaccia

- Abilitare Modbus ed impostare come porta predefinita dove agirà il protocollo la **502**.

```
1  modbus :  
2    enabled: yes  
3    detection-ports :  
4      dp: 502
```

Listing 3.3: `suricata.yaml`: Modbus

- Abilitare la creazione di log tramite Lua ed impostare una directory dove mettere gli script lua, ovvero `/etc/suricata/rules/lua-outputs/`.

```
1  - lua :  
2    enabled: yes  
3    scripts-dir: /etc/suricata/lua-output/  
4    scripts :  
5      - luatest.lua
```

Listing 3.4: `suricata.yaml`: Script Lua Output

- Aggiungere la regola **modbus\_detect.lua** nella directory predefinita `/etc/suricata/rules/`.



La regola, `modbus__detect.lua`, è la seguente:

```
1 alert modbus any any -> any any (msg:"Too much  
2 CPU usage/ high temperature/high humidity.  
3 Possible attack!!!"; luajit:prova1.lua;  
4 threshold: type threshold, track by_src,  
5 count 2, seconds 180; sid:103; rev:1;)
```

Listing 3.5: Regola detection traffico Modbus

Lo script Lua annesso, `prova1.lua`, legge i valori numerici successivi alle stringhe da controllare, nello specifico **temperatura, umidità e utilizzo della CPU**, e fa "scattare" la regola Suricata nel caso superino delle soglie predefinite (rispettivamente **60C, 50% e 70%**). Inoltre è anche necessario che le condizioni stabilite in precedenza si verifichino **più di 2 volte in 180 secondi (3 minuti)**. In questo particolare caso Lua si dimostra di gran lunga migliore rispetto alle normali keywords Suricata. Tramite l'utilizzo di **content**, non è possibile analizzare degli specifici valori numeri successivi a delle stringhe, rendendo l'analisi del payload poco funzionale per un utilizzo specifico, come nell'ambito industriale. Lo script Lua, quindi, diventa indispensabile per questo progetto.

A questo punto è necessario sottolineare come i due file di log, **fast.log** e **eve.json**, siano ottimi per l'analisi delle attività sospette ma non perfetti. Fast.log non riporta dati importanti tranne per il messaggio che caratterizza la regola che l'ha generato, mentre eve.json, nonostante offra la possibilità di mostrare il payload del pacchetto abilitandolo dal file **suricata.yaml**, è più completo ma è espresso in un formato piuttosto scomodo da leggere.

Per questo motivo c'è stata la necessità di creare un file di log nuovo e più funzionale rispetto ai due citati in precedenza tramite il supporto che Suricata offre al linguaggio Lua. Infatti sono disponibili dei metodi che permettono, a seconda dell'utilizzo che si vuole, di aggiungere informazioni ai file di log come ad esempio **SCPacketTuple()**, il quale restituisce una tupla composta nell'ordine la **versione del protocollo IP, indirizzo IP della sorgente e del destinatario, porta della sorgente e del destinatario**. Il proto-

collo restituito da `SCPaketeTuple` non è rappresentato dal suo nome ma dal numero del suo identificativo univoco. Sarà quindi necessario convertire tale valore numerico nel nome del protocollo [14].

Il nuovo script Lua finalizzato a generare un nuovo file di log verrà inserito nella directory `/etc/suricata/luat-output/` ed il file di log generato **luat-test.log** sarà posizionato in `/var/log/suricata/`, la stessa cartella dove si trovano i log standard.

Il formato del file di log **luat-test.log** contiene una sezione dedicata all'identificazione della regola che ha contribuito a generare l'alert composta dal **messaggio**, l'**identificativo** e la **versione della regola**. In seguito viene riportato il numero di volte in cui la regola si è verificata insieme al **timestamp** nel formato **mese/giorno/anno-ora:minuti:secondi.millisecondi**, la **versione del protocollo IP (IPv4 o IPv6)**, il **protocollo** e l'**indirizzo/porta sorgente e destinatario**.

Infine viene mostrato il payload completo del pacchetto e la parte specifica che ha generato l'alert. in questo caso la parte sospetta sarà formata da 3 stringhe con i valori associati all'uso della CPU, la temperatura e l'umidità, con i rispettivi valori.



# Capitolo 4

## Risultati

Per effettuare i seguenti test, è stata creata un' infrastruttura di sistema fittizia composta da **un client, 2 router e un server**.

Lo strumento utilizzato per creare quanto citato pocanzi è **Vagrant** [15]. Si tratta di uno strumento di **HashiCorp** che permette di creare e gestire ambienti di sviluppo virtualizzati in modo riproducibile e portabile. Con Vagrant, gli sviluppatori possono definire e configurare macchine virtuali specifiche attraverso file di configurazione chiamati '**Vagrantfile**'.

Queste macchine possono poi essere avviate, fermate e distrutte facilmente. Vagrant si integra con vari provider di virtualizzazione come **VirtualBox**, **VMware** e **AWS**, offrendo agli sviluppatori un ambiente consistente su diverse piattaforme e sistemi.

È fondamentale chiarire e specificare l'architettura e la disposizione dei componenti all'interno della topologia virtualizzata creata con Vagrant. In particolare, è essenziale delineare la posizione e le funzioni delle componenti nel client e server. Nel **server** è stato avviato il **master Modbus**, mentre nel **client** è stato avviato lo **slave pyModbus** e configurato **Suricata**. In entrambi è stato installato **Python** nella versione **3.9.2**, requisito indispensabile per avviare gli script pyModbus.



Si va perciò ad avviare il master e lo slave con il demone di Suricata sempre attivo, mentre contemporaneamente si controlla il file di log **luatest.log** tramite il comando `sudo tail -f /var/log/suricata/luatest.log`, così da osservare live i dati.

```
ALERT DETAILS
Too much CPU usage/high temperature/high humidity. Possible attack!!!
sid: 103, rev: 1, gid: 1, Priority: 3
N:4 | Timestamp: 08/31/2023-17:11:57.713252 | IPv4 | Protocol: TCP
| Source/Destination: 192.168.10.10:54334 → 192.168.20.20:502
| CRITICAL PAYLOAD
CPU Usage: 77%
Temperature: 64°
Humidity: 53%

TOTAL PAYLOAD:

O{:t
CPU Usage: 77%
Pressure: 1hPa
Humidity: 71%
Temperature: 64C
gljoaqngs
Humidity: 1%
Humidity: 53%
bagicjhulcvjbkt
```

Figura 4.2: Log luatest.log

Come si evince dall'immagine 4.2 il test è andato a buon fine.

Da notare come lo script Lua analizza sempre l'ultimo valore rilevato, infatti nonostante siano presenti molteplici occorrenze della stringa "Humidity" viene analizzato sempre l'**ultimo in ordine cronologico**, dato che un macchinario può inviare più dati riguardo un suo stato anche a distanza di poco tempo.

È possibile, per rendere il test più verosimile, impostare un timer tra ogni richiesta ad esempio di **2 secondi**, per simulare al meglio la comunicazione che un macchinario può intraprendere con un PLC. Nel test il timer viene omesso per rendere il processo più rapido.

La seconda fase consiste nell'analizzare anche il secondo file di log nel formato CEF, **cefctest.cef**, sempre tramite il comando **tail -f**.

Grazie al file di configurazione **suricata.yaml** è possibile infatti avere contemporaneamente due script Lua che generano file di log separati, ma inerenti allo stesso evento sospetto rilevato.

```
vagrant@client:~$ sudo tail -f /var/log/suricata/cefctest.cef
CEF:0|OISF|Suricata|6.0.1|1:103:1|Modbus Communication|3| rt=09/01/2023-10:24:54.644893 act=alert proto=TCP src=192.168.10.10 spt=60414 dst=192.168.20.20 dpt= 502 msg=Too much CPU usage/high temperature/high humidity. Possible attack!!!
```

Figura 4.3: Log cefctest.cef

Il risultato segue il formato standard del **Common Event Format**, ovvero **CEF:Versione|Fornitore Programma|Programma|Versione Programma|Id Regola|Nome Evento|Priorità|Estensioni**.

## Capitolo 5

### Conclusioni e sviluppi futuri

Il focus di questa tesi è incentrato sullo studio di regole Suricata per il monitoraggio del traffico di dati in protocolli industriali, nello specifico Modbus. È stato fondamentale trovare dei metodi per aumentare l'espressività di tali regole, in quanto spesso l'utilizzo esclusivo delle keywords si è rivelato insufficiente. Uno dei modi per ovviare a questi problemi è stato l'uso del linguaggio di programmazione Lua, il quale è supportato nativamente da Suricata.

Per eseguire gli adeguati test è stata necessaria la creazione di un infrastruttura del protocollo Modbus, tramite la libreria Python **pyModbus**, che simula la comunicazione tra un macchinario ed un PLC rispettivamente rappresentati da un client ed un server.

Il primo obiettivo raggiunto riguarda la **creazione delle regole Suricata per il rilevamento di dati o pattern sospetti** e l'ampliamento dell'espressività di quest'ultime. Tale miglioramento è stato effettuato tramite il linguaggio di programmazione Lua, il quale ha permesso di manipolare al meglio il payload dei pacchetti con i messaggi Modbus. L'espressività ottenuta con Lua si è rivelata un arma estremamente utile che rende obsolete le keyword native di Suricata nei casi di detection più specifici.

Il linguaggio **Lua**, oltre all'integrazione con le regole Suricata, ha consentito con successo la **creazione di nuovi file di log** con informazioni specifiche



per il caso d'uso di questa tesi. Ancora una volta le funzioni e metodi integrati in Suricata hanno permesso di personalizzare i parametri nei log e di focalizzarsi unicamente sulle informazioni di cui si necessita, superando in comodità i due file di log già forniti da Suricata **eve.json** e **fast.log**.

Successivamente, creando script Lua che generano file di log in formati specifici come il **CEF (Common Event Format)**, si è facilitata l'**integrazione di Suricata con piattaforme di analisi e gestione degli eventi di sicurezza** come **ArcSight**, **Splunk** e molte altre. Questa interoperabilità non solo migliora la visibilità degli eventi di sicurezza, ma anche accelera il processo decisionale, consentendo alle organizzazioni di rispondere prontamente alle minacce emergenti. Per continuare la tesi, sarebbe utile creare nuovi script Lua per generare log in diversi formati come ad esempio **LEEF** o **ECS**, per rimanere nell'ambito industriale. Se invece ci si vuole spostare anche su logging nell'ambito web è possibile supportare **NCSA Combined Log Format**, **ELF (Extended Log Format)** o **W3C Extended Log File Format**.

# Acronimi

**AWS** Amazon Web Services. 37

**CEF** Common Event Format. 40, 42

**CPU** Central Processing Unit. 35

**CR** Carriage Return. 20

**CRC** Cyclic Redundancy Check. 21, 22

**CTS** Clear To Send. 19

**DCE** Data Communication Equipment. 18, 19

**DTE** Data Terminal Equipment. 18, 19

**ECS** Elastic Common Schema. 42

**ELF** Extended Log Format. 42

**HIDS** Hybrid intrusion detection system. 4

**IDS** Intrusion Detection System. v, 3, 4, 8, 33

**IIoT** Industrial Internet Of Things. 25, 26

**IP** Internet Protocol. 8, 11, 19, 23, 29, 34, 35

**IPS** Intrusion Prevention System. 8

- IT** Information Technology. 3
- JIT** Just In Time. 13
- JSON** JavaScript Object Notation. 12
- LEEF** Log Event Extended Format. 42
- LF** Line Feed. 20
- LRC** Longitudinal Redundancy Check. 21, 22
- mbps** MegaBit Per Second. 19
- NIDS** Network-based intrusion detection system. 4
- OISF** Open Information Security Foundation. v, 5, 6
- PLC** Programmable Logic Controller. i, xiii, 16, 39, 41
- PPA** Personal Package Archive. 6
- RTS** Ready To Send. 19
- RTU** Remote Terminal Unit. 19, 21, 22
- SSH** Secure Shell. 12
- TCP** Transmission Control Protocol. 8, 19, 23, 29
- TLS** Transport Layer Security. 12
- W3C** World Wide Web Consortium. 42
- Wi-Fi** Wireless Fidelity. 19

# Bibliografia

- [1] “Ids,” Mar 2023. [Online]. Available: <https://www.geeksforgeeks.org/intrusion-detection-system-ids/>
- [2] “Installation - suricata 7.0.2-dev documentation.” [Online]. Available: <https://docs.suricata.io/en/latest/install.html>
- [3] “Suricata rules example.” [Online]. Available: [https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata\\_Rules](https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Rules)
- [4] V. Julien and J. Fajardini, “Lua and luajit keyword,” Sep 2021. [Online]. Available: <https://forum.suricata.io/t/lua-and-lualjit-keyword/1687>
- [5] “Lua output - suricata 7.0.1-dev documentation.” [Online]. Available: <https://docs.suricata.io/en/latest/output/lua-output.html>
- [6] “Lua functions supported in suricata.” [Online]. Available: <https://docs.suricata.io/en/latest/lua/lua-functions.html>
- [7] D. Postacchini. [Online]. Available: <https://www.danielepostacchini.it/wp-content/uploads/2020/08/TUTORIAL-MODBUS.pdf>
- [8] “Modbus application protocol specification.” [Online]. Available: [https://www.modbus.org/docs/Modbus\\_Application\\_Protocol\\_V1\\_1b.pdf](https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf)
- [9] “Modbus rtu vs tcp.” [Online]. Available: <https://www.wevolver.com/article/modbus-rtu-vs-tcp-a-comprehensive-comparison-of-industrial-protocols>

- 
- [10] “Format cef example.” [Online]. Available: [https://prod.docs.oit.proofpoint.com/configuration\\_guide/integration\\_using\\_cef\\_logs.htm](https://prod.docs.oit.proofpoint.com/configuration_guide/integration_using_cef_logs.htm)
  - [11] “Pymodbus documentation.” [Online]. Available: <https://pymodbus.readthedocs.io/en/latest/>
  - [12] “Wireshark informations.” [Online]. Available: <https://www.wireshark.org/>
  - [13] “Modbus write/read register limitations.” [Online]. Available: <https://www.mesulog.fr/help/modbus/index.html?page=write-multiple-registers-f16.html>
  - [14] “List of ip protocol numbers,” Jul 2023. [Online]. Available: [https://en.wikipedia.org/wiki/List\\_of\\_IP\\_protocol\\_numbers](https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers)
  - [15] “Vagrant by hashicorp2023,” Aug 2023. [Online]. Available: <https://www.vagrantup.com/>
  - [16] “Github repository.” [Online]. Available: [https://github.com/UniboSecurityResearch/Marinelli\\_Edoardo\\_BT/tree/main/project](https://github.com/UniboSecurityResearch/Marinelli_Edoardo_BT/tree/main/project)