Università del Piemonte Orientale

Dipartimento di Scienze e Innovazione Tecnologica

Corso di Laurea in Informatica

Relazione per la prova finale

Progettazione ed implementazione di un solver di SecuriDN per l'analisi di grafi d'attacco

Tutore interno:

Prof. Davide Cerotti

Candidato:

Nicolò Vittorio Marino

Anno Accademico 2023/2024

Abstract

Questo lavoro si concentra sulla progettazione e implementazione di un solver per il tool SecuriDN [1], uno strumento utilizzato per modellare e analizzare attacchi informatici all'interno di infrastrutture energetiche. Il solver sviluppato traduce i grafi di attacco in script Octave [2], che consentono di analizzare la corrispondente Rete Bayesiana Dinamica (DBN). Questa analisi permette di calcolare la probabilità di successo di un attacco da parte di un soggetto malevole su un nodo della rete e valutare la propagazione dell'attacco verso altri nodi. Il solver è stato testato su configurazioni di rete di diversa complessità, dimostrando flessibilità e accuratezza nell'analisi. Lo script creato sarà in seguito unito ad un modulo di rilevamento di cyber-attacchi.

Indice

Elenco delle figure

Listings

Acronimi

Bayesian Network BN. 4–6

Conditional Probability Table CPT. 3–5, 8, 10, 11, 16, 20, 22–24, 27, 28

Draw-Net Modeling System DMS. iv, 12–14

Dynamic Bayesian Network DBN. i, v, 2, 3, 6–9, 11, 14–16, 18–25, 27–29, 31

Grafo Diretto Aciclico DAG. 4

Intelligenza Artificiale IA. 3, 14

Interfaccia grafica GUI. 12, 14

Mean Time to Compromise MTTC. v, 20, 21, 24, 27

Probability of Failure PrF. v, 20, 21, 24, 27, 28

Capitolo 1

Introduzione

Negli ultimi anni la sicurezza informatica è diventata un argomento fondamentale, la motivazione dell'importanza la otteniamo dai dati; nel 2023 ci sono stati più di 160000 incidenti e di questi più di 5000 sono data breaches[1] [4]. Ci permette di difenderci da possibili minacce che possono portare, ad esempio, a danni economici. Si potrebbe pensare di andare a sviluppare applicazioni complesse in modo tale da rendere più difficile l'attacco a esse, ma questo non conviene in quanto più un sistema è complesso e più sarà difficile verificarne la correttezza d'implementazione, gestione e funzionamento. Infatti tipicamente il numero di bachi [2] è proporzionale al numero di righe di codice, quindi più righe ci sono nel codice e più sarà alto il numero di possibili bachi. Nello sviluppo software si cerca di attenersi al principio Keep It Simple Stupid (KISS), dove si suggerisce di creare soluzioni semplici in modo tale da rendere più facile la comprensione, affidabilità e la manutenibilità.
Gli attacchi informatici seguono una cyber-kill-chain [3] e l'obiettivo finale del mio stage è il rilevamento precoce di un attacco su una determinata componente della rete. Ci concentreremo principalmente sulla sicurezza informatica presente nelle infrastrutture energetiche per evitare che ci siano incidenti, come ad esempio quelli provocati da malware come Stuxnet [5] o Black Energy [6].
Questo lavoro è un modulo di un progetto più grande che si concentra sullo sviluppo di una piattaforma per il monitoraggio e il rilevamento di cyber-attacchi sulle reti di controllo delle infrastrutture energetiche.
Per facilitare la comprensione di questa piattaforma e del modello di rilevamento, si presentano di seguito alcune immagini che descrivono l'integrazione e l'architettura della piattaforma. La Fig. 1.1 mostra l'architettura della rete divisa in diverse zone: la *rete OT* (Operational Technology), la *zona DMZ* (Demilitarized Zone) e la *rete*

[1] Un data breach è un incidente che espone dati sensibili.
[2] I bachi sono conosciuti anche come "bug" in inglese.
[3] La Cyber Kill Chain è un modello che descrive le fasi con il quale un soggetto malevole attacca un sistema

corporate. I dati di monitoraggio provenienti da tutta la rete vengono in seguito inviati ad *Opensearch* (piattaforma per l'analisi e la ricerca di dati) tramite *Kafka* (piattaforma per la comunicazione tra vari modelli di rilevamento).

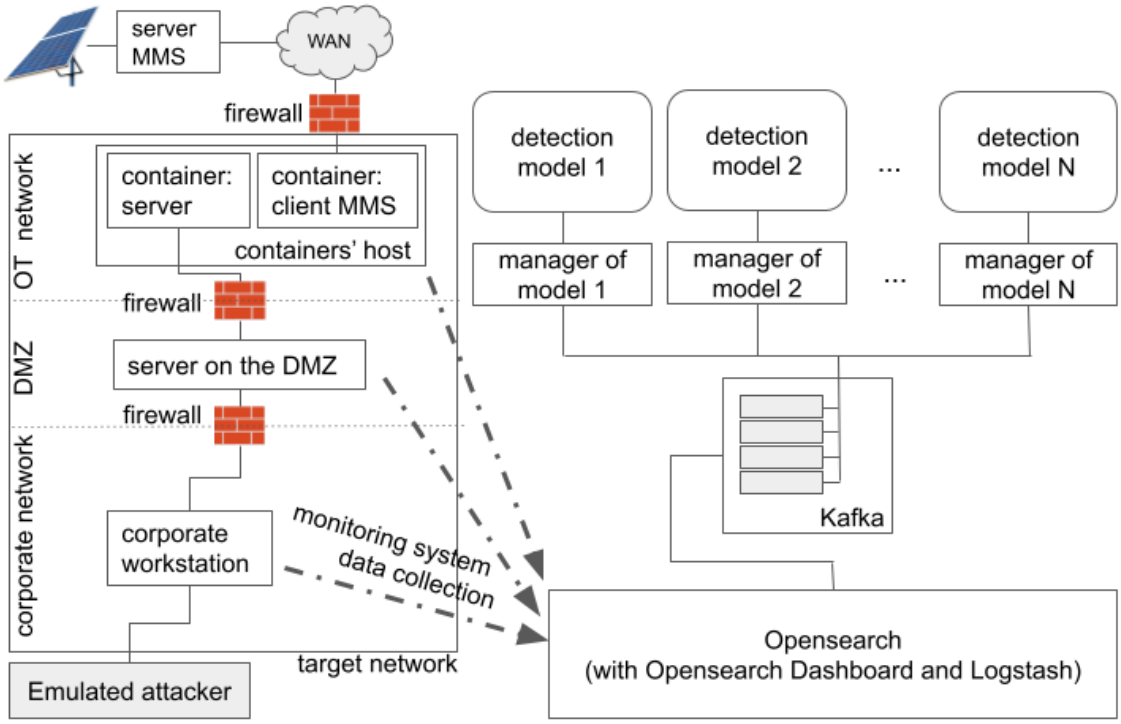

Figura 1.1: Architettura della rete

Per quanto riguarda la Fig. 1.2, viene illustrato il flusso operativo dal punto di vista dell'analista di sicurezza. L'analista utilizza l'interfaccia grafica di SecuriDN per definire l'architettura. Da questa viene generato un grafo di attacco (GA) il quale viene infine convertito in una DBN. Rappresenta anche l'integrazione tra il flusso operativo e l'architettura. Vengono presentati diversi output dei vari modelli di rilevamento e tra questi c'è lo script Octave generato dal solver descritto in questo lavoro.

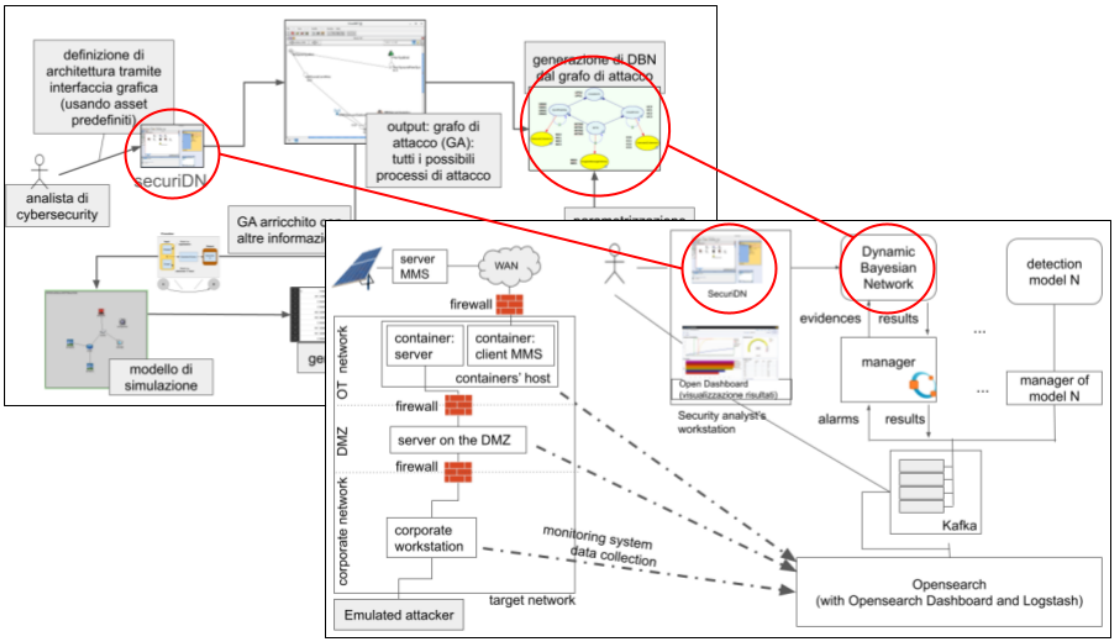

Figura 1.2: Integrazione generale della piattaforma di rilevamento con SecuriDN

In [1] è stato presentato SecuriDN, uno strumento grafico progettato per aiutare gli analisti della sicurezza a creare modelli di rilevamento delle minacce basati sull'Intelligenza Artificiale (IA). Questo avviene attraverso un'interfaccia grafica che consente di definire l'architettura di rete d'interesse. Ogni risorsa nell'architettura contiene informazioni sui possibili attacchi che può subire e le loro relazioni mostrano come tali attacchi possono propagarsi ad altre risorse. In questo, invece, viene modificato un solver di tale tool in modo tale che vada a creare uno script Octave contenente le informazioni della Rete Bayesiana Dinamica (DBN) di riferimento per poi inviarlo a un modulo di rilevamento degli attacchi.
Il documento è così organizzato:

- Nel capitolo 2 viene introdotto il dominio del discorso discutendo il funzionamento delle DBN e della Conditional Probability Table (CPT).
- Nel capitolo 3 vengono definiti i framework, strumenti e librerie usate.
- Nel capitolo 4 viene descritto lo scopo del lavoro.
- Nel capitolo 5 si tratta la modifica del solver per SecuriDN e spiegato lo script Octave.
- Nel capitolo 6 verranno analizzati i risultati.

Capitolo 2

Nozioni preliminari

2.1 Bayesian Network

Una Rete Bayesiana (BN) è un modello probabilistico usato per rappresentare in modo compatto la probabilità congiunta $\Pr(X_1, X_2, ..., X_n)$ di un insieme di variabili aleatorie $\{X_1, X_2, ..., X_n\}$, sfruttando le relazioni di dipendenza condizionale tra di loro. Si basa su un grafo diretto senza cicli (DAG), dove ogni nodo rappresenta una variabile e ogni collegamento tra i nodi descrive una dipendenza condizionale. Le probabilità associate a ciascuna variabile sono organizzate in Tabelle delle Probabilità Condizionate (CPT), che specificano il valore di una variabile dipendente dal valore delle altre. Consideriamo una semplice BN, la quale mette in relazione il tempo atmosferico con le attività in un laboratorio informatico [3].

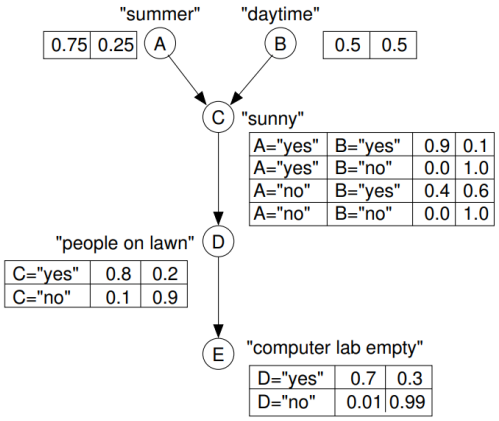

Figura 2.1: Esempio Bayesian Network [3]

Nella Fig. 2.1 vediamo la struttura della rete composta da diverse variabili e le CPT vengono calcolate basandosi solo sulla configurazione del valore dei genitori. Le BN sono utili per la loro capacità di rappresentare situazioni incerte ma esse sono statiche e non modellano i cambiamenti delle variabili nel tempo, il che limita la loro

applicazione in contesti dinamici ossia dove, ad esempio, l'evoluzione temporale di un nodo potrebbe dipendere sia dal valore delle variabili dei genitori che dal proprio valore all'istante temporale precedente.

2.2 Conditional Probability Table

La probabilità condizionata associata a ogni variabile è descritta tramite la Tabella delle Probabilità Condizionate (CPT). I valori di probabilità nella CPT vengono determinati tramite osservazioni, dati empirici o modelli. Facendo riferimento alla Fig. 2.1 possiamo considerare il fatto che ci siano tante persone nel prato dato che è soleggiato. I parametri li otteniamo dalla CPT del nodo D mostrati nella tabella 2.1:

	D = "yes"	D = "no"
C = "yes"	0.8	0.2
C = "no"	0.1	0.9

Tabella 2.1: CPT del nodo D

2.3 Formula di Bayes

La formula di Bayes [7] è un'equazione utilizzata per aggiornare la probabilità di un evento o condizione in base alle nuove informazioni, o evidenze. Ad esempio, per calcolare la probabilità condizionata di una variabile X dato Y, si usa questa formula:

$$P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)} \tag{2.1}$$

dove:

- $P(X|Y)$ è la probabilità che si verifichi X sapendo che è avvenuto Y.
- $P(Y|X)$ è la probabilità che si verifichi Y sapendo che è avvenuto X.
- $P(X)$ è la probabilità di X.
- $P(Y)$ è la probabilità di Y.

Considerando la BN presente nella Fig. 2.1, possiamo calcolare la probabilità che il laboratorio di informatica sia vuoto ($E = yes$) dato che le persone sono nel giardino ($D = yes$). Per fare questo, utilizziamo la formula di Bayes. In particolare, avremo bisogno di:

- $P(D = yes|E = yes)$ è la probabilità che le persone siano nel giardino dato che il laboratorio di informatico è vuoto.

- $P(E = yes)$ è la probabilità che il laboratorio sia vuoto.
- $P(D = yes)$ è la probabilità che ci siano persone nel giardino.

e quindi:

$$P(E = yes|D = yes) = \frac{P(D = yes|E = yes) \cdot P(E = yes)}{P(D = yes)} \tag{2.2}$$

2.4 Dynamic Bayesian Network

Una DBN è un'estensione della BN classica, in questo caso vengono modellate relazioni probabilistiche tra variabili, i cui valori cambiano dinamiche nel tempo. Il tempo viene considerato in modo discreto[1], quindi una DBN rappresenta processi stocastici a tempo discreto che evolvono e modellano l'evoluzione delle variabili attraverso slice temporali. Più precisamente, una DBN può essere rappresentata tramite processi stocastici semi-markoviani [8] di ordine $k - 1$. Un processo semi-Markoviano è un tipico processo stocastico in cui la probabilità dello stato dipende non solo dallo stato attuale ma anche dagli stati passati. Quando, ad esempio, stiamo andando a utilizzare un ordine $k = 2$ stiamo praticamente dicendo che la probabilità di uno stato dipenderà da solo due slice temporali: lo slice al tempo t e al tempo $t - \Delta t$. Questo ci permette di andare a rappresentare e analizzare l'evoluzione nel tempo del sistema.
Per comprendere al meglio la struttura di una DBN, consideriamo la Fig. 2.2, la quale illustra una semplice rete con due variabili, X_1 e X_2, osservate in due istanti di tempo consecutivi: t e $t - \Delta$. Gli archi continui indicano le dipendenze probabilistiche all'interno dello slice temporale (intraslice), mentre gli archi tratteggiati rappresentano dipendenze tra slice consecutivi (interslice).

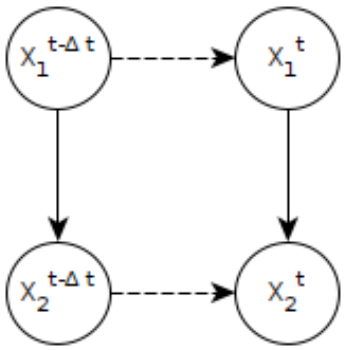

Figura 2.2: Esempio Dynamic Bayesian Network

[1]Rappresentare il tempo in modo discreto significa suddividere il tempo in istanti o intervalli distinti.

In una DBN, la formula di Bayes viene usata per aggiornare le probabilità delle variabili nel tempo: ad esempio, al tempo $t+1$ si ricalcolano le probabilità in base alle evidenze osservate al tempo t. Supponiamo di avere una variabile X_t al tempo t e la stessa variabile X_{t+1} al tempo $t+1$. La probabilità di X_{t+1} dato che abbiamo osservato un'evidenza E_t può essere calcolata così:

$$P(X_{t+1}|E_t) = \frac{P(E_t|X_{t+1}) \cdot P(X_{t+1})}{P(E_t)} \tag{2.3}$$

Questo approccio permette di aggiornare continuamente le probabilità man mano che si ottengono nuove informazioni.

2.4.1 Inferenze

Le inferenze sono delle procedure tramite cui vengono calcolate le probabilità dalle variabili interessate attraverso le evidenze osservate (osservazioni concrete e misurabili che riflettono lo stato del sistema). La loro implementazione permette il monitoraggio, la previsione e la diagnostica degli stati del sistema modellato che nel nostro caso faranno riferimento alla sicurezza della rete. Ad esempio, immaginiamo di monitorare un sistema informatico che registra i tentativi di login da parte degli utenti. Se osserviamo un picco nel numero di tentativi di accesso in un breve periodo di tempo, potrebbe esserci la possibilità che qualcuno stia tentando di accedere al sistema tramite una tecnica di attacco a forza bruta, nel quale vengono provate tutte le password possibili fino ad ottenere quella corretta. Nel caso il sistema utilizzi tecniche di inferenze, questa evidenza può essere utilizza per calcolare la probabilità che effettivamente si stia verificando un attacco a forza bruta. Un'analisi probabilistica tramite una DBN potrebbe indicare che, basandosi sulle osservazione dei dati, si stia verificando un attacco a forza bruta.
Le inferenze nella DBN sono guidate da diversi parametri, che definiscono il tipo di analisi e il metodo da applicare. Il primo di questi è il tipo di inferenza da eseguire. Le due modalità principali sono:

- **Filtering**: aggiorna la probabilità sui nodi basandosi su tutte le osservazioni raccolte fino a quel momento. In pratica, ci permette di stimare lo stato attuale del sistema considerando le informazioni passate e le osservazioni presenti. Nel codice, viene indicato con *filtering=1*.

- **Smoothing**: include sia le osservazioni passate che future per stimare lo stato di un nodo in un momento passato, migliorando la precisione grazie ai dati successivi. Nel codice, questa modalità è indicata con *filtering=0*.

In breve, il *filtering* ci aiuta a capire cosa sta accadendo nel presente, mentre il *smoothing* fornisce una visione più accurata di uno stato passato, sfruttando tutte

le osservazioni.
Un altro parametro chiave è l'algoritmo da utilizzare per l'inferenza sui nodi:

- **BK (Boyen-Koller)**: sfrutta il fatto che lo spazio degli stati può essere suddiviso, semplificando i calcoli necessari ma approssimando le probabilità.
- **JT (Junction Tree)**: più preciso ma anche più complesso, questo algoritmo trasforma la DBN in una struttura chiamata "albero di giunzione" per fare inferenze esatte. È ideale per contesti in cui serve precisione e la rete non è troppo complessa.

C'è anche un parametro booleano che indica se il modello è completamente fattorizzato. In tal caso, le CPT sono rappresentate come prodotti di distribuzioni più semplici, riducendo i calcoli ma a scapito della precisione.
Infine, il parametro che indica ogni quanto tempo viene eseguita l'inferenza tra due istanti successivi (o "slice" temporali). In una DBN, ogni slice rappresenta uno stato del sistema in un certo momento: un *time_step* breve implica uno sguardo più dettagliato sugli stati intermedi, consentendo un'analisi temporale più accurata tra il momento iniziale e quello finale.
Le tecniche di inferenza possono essere un ottimo strumento per identificare potenziali vulnerabilità e attacchi, facilitando una risposta tempestiva e informata.

Capitolo 3

Framework, Strumenti e Librerie utilizzati

3.1 Ambiente di lavoro e Librerie

Prima di descrivere i framework e i software utilizzati, è importante presentare l'ambiente di lavoro e le librerie necessarie. Per garantire il corretto funzionamento il software è stato sviluppato sulla versione 18.04 di Ubuntu [9], che risulta compatibile con le librerie richieste. Per quanto riguarda le librerie, ce ne sono diverse.

BNT-1.0.7 La libreria Bayes Net Toolbox (BNT) [10] è stata originariamente sviluppata in MATLAB, ma oggi non è più mantenuta e ha requisiti ormai superati. Nonostante ciò, è una delle poche librerie non commerciali che implementa sofisticati algoritmi di inferenza capaci di soddisfare i requisiti di progetto. Pertanto si è deciso di utilizzare una versione modificata della libreria, resa compatibile con Octave grazie agli interventi del Prof. Davide Cerotti[1], del Prof. Daniele Codetta Raiteri[2] e del Prof. Luigi Portinale[3].
Nella libreria, è importante considerare alcune funzioni chiave, come ad esempio *mk_dbn*, che permette di creare una Rete Bayesiana Dinamica (DBN). Viene definita come:

```
1 bnet = mk_bnet(intra, inter, node_sizes, varargin)
```

Listing 3.1: Funzione mk_bnet

I parametri sono:

[1] davide.cerotti@uniupo.it (D. Cerotti)
[2] daniele.codetta@uniupo.it (D. Codetta-Raiteri)
[3] luigi.portinale@uniupo.it (L. Portinale)

- *intra*: rappresenta una matrice di archi tra i nodi all'interno dello stesso slice temporale[4].
- *inter*: rappresenta una matrice di archi tra i nodi di slice temporali consecutivi.
- *node_sizes*: rappresenta un vettore che specifica il numero di valori che ciascun nodo può assumere (nel caso di un nodo binario si avrà [2]).
- *varargin*: rappresenta dei parametri opzionali.

Funzione *tabular_CPD* tramite cui viene creata la Conditional Probability Table (CPT). Viene definita come:

```
CPD = tabular_CPD(bnet, self, varargin)
```

Listing 3.2: Funzione tabular_CPD

I parametri sono:

- *bnet*: rappresenta la rete bayesiana dove appartiene il node.
- *self*: rappresenta il nodo per cui si crea la CPT.
- *varargin*: rappresenta dei parametri opzionali.

Funzione *mk_named_CPT* tramite cui viene cambiato l'ordine delle dimensioni della CPT per far sì che corrispondano all'ordine dei nodi. Viene definita come:

```
CPT = mk_named_CPT(family_names, names, dag, CPT1);
```

Listing 3.3: Funzione mk_named_CPT

I parametri sono:

- *family_names*: i nomi dei nodi per cui stiamo creando la CPT di un certo ordine.
- *names*: nomi di tutti i nodi nella rete.
- *dag*: la rete è rappresentata tramite matrice di adiacenza (bnet.dag).
- *CPT1*: la CPT originale.

Funzione *mk_named_CPT_inter* che è una versione modificata della funzione precedente. Fa lo stesso lavoro solo che in questo caso si tiene conto di nodi che appartengono a slice temporali diversi. Viene definita come:

[4]uno slice temporale rappresenta lo stato del sistema in uno specifico momento

```
1 CPT = mk_named_CPT_inter(family_names, names, dag, CPT1, idx_2);
```

Listing 3.4: Funzione mk_named_CPT_inter

I parametri sono:

- *family_names*: nomi dei nodi per cui stiamo creando la CPT di un certo ordine.
- *names*: i nomi di tutti i nodi nella rete.
- *dag*: la rete è rappresentata tramite matrice di adiacenza (bnet.dag).
- *CPT1*: la CPT originale.
- *dx_2*: gli indici dei nodi appartenenti al secondo slice.

Funzione *boolean_CPD* permette di creare una CPT che rappresenta una funzione booleana. Viene definita come:

```
1 CPD = boolean_CPD(bnet, self, ftype, fname);
```

Listing 3.5: Funzione boolean_CPD

I parametri sono:

- *bnet*: rappresenta la rete bayesiana dove appartiene il node.
- *self*: rappresenta il nodo per cui si crea la CPT.
- *ftype*: il tipo di funzione booleana che vogliamo usare.
- *fname*: la funzione booleana che vogliamo usare ('all' per AND oppure 'any' per OR).

Tramite questa libreria possiamo gestire le DBN e le CPT in modo tale da ottenere le inferenze e i valori degli slice temporali.

3.2 Octave

In questo lavoro si utilizzerà GNU Octave, un linguaggio di programmazione ad alto livello pensato per eseguire calcoli numerici [2]. Questo linguaggio è molto compatibile con Matlab ed è inoltre open source. Include diverse funzioni per risolvere i problemi legati ai calcoli numerici, anche complessi, ma li useremo principalmente per verificare la correttezza dello script creato dal solver. Verranno utilizzate le librerie liboctave-dev e io-2.6.3.tar.gz.

liboctave-dev liboctave-dev è un pacchetto necessario per la libreria *liboctave* che permette la compilazione di script Octave.

io-2.6.3.tar.gz io-2.6.3.tar.gz è un pacchetto per Octave necessario per la lettura e scrittura di file come ad esempio quelli CSV, Excel, ecc...

3.3 SecuriDN

SecuriDN [1] è una personalizzazione dello strumento DrawNet Modeling System (DMS), progettato per rappresentare processi d'attacco nei sistemi energetici. Questo strumento fornisce un'interfaccia grafica (GUI) per la progettazione di modelli basati sui grafi, definendo la grammatica che ne descrive la struttura ed elementi. Gli elementi di base e le relative proprietà sono specificati dai formalismi, mentre la GUI consente di adattare il modello in conformità alla grammatica. SecuriDN è un formalismo definito specificatamente per modellare processi d'attacco nei sistemi energetici, questo è possibile grazie alla GUI che consente di rappresentare e analizzare la propagazione delle minacce all'interno dei sistemi.

3.3.1 DrawNet Modeling System

Draw-Net Modeling System (DMS) è un framework che supporta alla progettazione e soluzione di modelli espressi in qualsiasi formalismo a grafo. Lo strumento Draw-Net è un'interfaccia grafica basata su Java e utilizza la libreria DNlib [11] [12]. Il sistema si basa su un'architettura multilivello [13], descritta in Fig. 3.1, ed è suddiviso in:

- **Livello del formalismo**: definisce le primitive utilizzate per progettare un modello ed è definito dall'insieme degli elementi come nodi e archi, insieme degli attributi associati a un elemento e insieme dei vincoli che descrivono le relazioni richieste tra gli attributi.

- **Livello del model**: utilizza le primitive definite nel formalismo ed è formato dall'insieme delle istanze degli elementi e l'elemento principale del modello.

- **Livello del solver**: si occupa dell'analisi e della simulazione del modello. In questa fase, il modello progettato viene esportato in un file XML per l'elaborazione da parte del solver

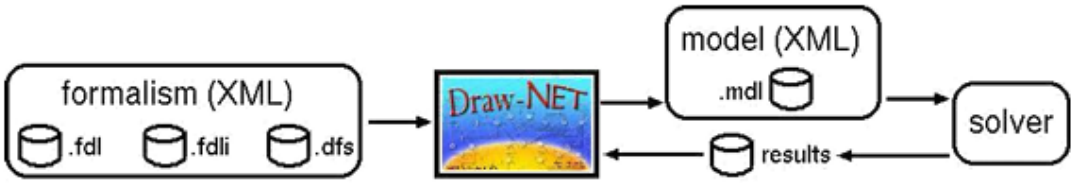

Figura 3.1: Architettura DMS [1]

DMS utilizza la libreria DNlib, descritta nella sezione 3.3.2, per la gestione degli elementi del modello e delle proprietà.

3.3.2 DNlib

Di questa libreria ci interessa particolarmente le classi:

- **Model**: la classe gestisce una struttura del modello utilizzato. I metodi principali sono *getMainElement()*, il quale restituisce l'elemento principale del modello e *getSubElement(String id)*, il quale restituisce un sotto elemento specifico del modello dato il suo identificativo.

- **ElementInstance**: la classe memorizza un'istanza di un elemento del modello. I metodi principali sono *getElementType()*, il quale restituisce il tipo dell'elemento e *subElementsEnum()*, il quale restituisce un enum contenente tutti i sottoelementi.

- **ElementType**: la classe memorizza un elemento in base alla struttura del formalismo. Il metodo principale è *getId()* tramite cui otteniamo l'Id del tipo come ad esempio: NodeAND, Arc, DBN, ecc...

- **PropertyValue**: è una classe astratta per ottenere il valore di una proprietà. Ogni PropertyValue deve derivare da questa, come ad esempio: *StringPropertyValue*, *IntegerPropertyValue*, ecc...

È fondamentale avere una buona conoscenza di questa libreria per apportare modifiche al solver, poiché la gestione del modello richiederà l'uso di queste classi. Il software richiede Java JDK 1.8 per funzionare.

JDK 1.8 Java Development Kit (JDK) 1.8 è un insieme di strumenti utilizzato per sviluppare applicazioni in Java.

3.3.3 Architettura di SecuriDN

Come descritto nell'introduzione (1), SecuriDN è uno strumento grafico progettato per supportare gli analisti della sicurezza nella costruzione di modelli di rilevamento delle minacce basati sull'Intelligenza Artificiale (IA). Attraverso un'interfaccia grafica, è possibile definire l'architettura di rete di interesse, assegnando a ciascuna risorsa informazioni sui possibili attacchi che può subire. Le relazioni tra risorse mostrano inoltre come questi attacchi possano propagarsi da una risorsa all'altra. Dopo aver completato l'architettura, SecuriDN combina le informazioni raccolte in un grafo di attacco che rappresenta i vari processi di attacco possibili. Da questo si ottiene una DBN, la quale viene utilizzata per calcolare la probabilità che un componente della rete sia compromesso. Caratteristica fondamentale di questo strumento è che ha un approccio multi formalismo il quale consente di scegliere il formalismo più appropriato in base al problema da affrontare. SecuriDN si basa sull'architettura descritta precedentemente DMS, che fornisce l'interfaccia grafica (GUI) e la libreria DNlib per la costruzione e la manipolazione dei modelli.

Capitolo 4

Introduzione al lavoro

Lo scopo del lavoro è lo sviluppo di un solver di SecuriDN capace di produrre uno script Octave, che esegue inferenze su una Dynamic Bayesian Network (DBN) di riferimento tramite la modifica di un solver. Le inferenze sono dei processi tramite cui vengono estratte delle informazioni dalla DBN e saranno utili per un modulo successivo in grado di calcolare le probabilità di avere un cyber attacco. L'attività prevista è stata lo studio della libreria DNLib per la manipolazione degli oggetti interni a DrawNet, studio del linguaggio di scripting di Octave/Matlab e dei principi delle DBN. Il punto di partenza di questo lavoro sarà lo strumento SecuriDN tramite cui potremo lavorare con la DBN. Lo strumento SecuriDN include un filtro chiamato *SolverFilterDBN.java*, che è parte integrante della sua architettura come visto nella sotto sezione 3.3.1. Questo filtro può essere eseguito tramite la funzionalità rappresentata dalla lampadina presente nell'interfaccia grafica. Quando si attiva questa funzionalità, viene invocato il metodo *execute*. Il comportamento originale era una visita generale della DBN, con l'obiettivo di stampare i dati del modello. Il punto di partenza del lavoro è stato dunque la modifica di questo filtro, trasformandolo in modo da estendere le sue funzionalità. L'obiettivo principale è stato rendere il filtro non solo in grado di raccogliere i dati, ma anche di esportarli in uno script Octave. Questo script sarà utilizzabile per ulteriori analisi e inferenze basate sul modello della DBN. Questo processo comporterà la comprensione e la modifica del codice Java esistente, nonché l'integrazione con il linguaggio Octave per l'elaborazione successiva. Una volta modificato il filtro, verranno eseguite delle prove per garantire che i dati esportati siano corretti e completi. Infine, lo script Octave sarà testato per assicurarsi che le inferenze possano essere eseguite correttamente e che i risultati siano coerenti con le aspettative. Questo progetto fornirà uno strumento utile per analizzare le DBN in ambito di sicurezza informatica, contribuendo a rilevare la presenza di un attacco cyber attraverso una comprensione approfondita delle relazioni e delle dinamiche all'interno della rete.

4.1 Formalismo della DBN

La struttura della DBN è espressa attraverso un formalismo che definisce le primitive utilizzate per progettare un modello. Visto che per l'estrazione delle informazioni lo citeremo spesso, vediamo i suoi elementi fondamentali.
Il primo di questi è la DBN stessa che rappresenterà il modello tramite cui potremmo accedere agli altri tipi di elementi. Essa ha delle proprietà importanti, descritte nella sotto sezione 2.4.1, che saranno utili nella fase d'analisi, illustrate nella Fig. 4.1. Tra le più rilevanti troviamo:

- **inf_type**: tipo di inferenza da eseguire. Il valore predefinito è "filtering" ma potrebbe essere anche "smoothing".
- **inf_algorithm**: tipo di algoritmo di inferenza da utilizzare. Il valore prefefinito è "BK" ma potrebbe essere anche "JT".
- **fully_factorized**: proprietà booleana che indica se il modello è completamente fattorizzato.
- **time_step**: definisce il passo temporale tra i slice.

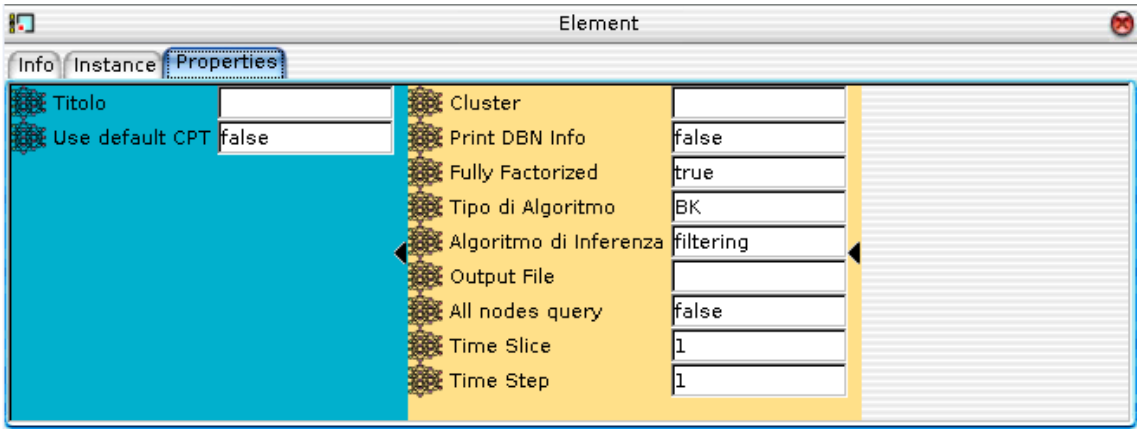

Figura 4.1: Proprietà della Dynamic Bayesian Network

In seguito abbiamo i nodi, i quali avranno proprietà differenti nel calcolo della CPT in base al tipo, ma in generale le proprietà saranno:

- **size**: rappresenta il numero di righe della CPT del nodo, dipende dal numero di nodi genitori e delle configurazioni possibili.
- **CPT**: serve per il calcolo della tabella delle probabilità condizionate.
- **parent**: i nodi genitori del nodo.

- **observed**: indica se il nodo è osservabile ossia se i suoi valori sono noti oppure dobbiamo stimare i valori.
- **obsValues**: contiene i valori osservabili del nodo.
- **query**: definisce i nodi da visualizzare nel grafico, permettendo di concentrarsi su quelli di interesse.

Possiamo vedere un esempio nella Fig. 4.2. Alcuni tipi di nodi, come *NodeAND* e *NodeOR*, si comportano secondo logiche specifiche corrispondenti a quelle degli operandi logici *AND* e *OR*. L'unica differenza è che i nodi *NodeAND* e *NodeOR* si comportano rispettivamente secondo la logica dell'operatore *AND* e dell'operatore *OR*.

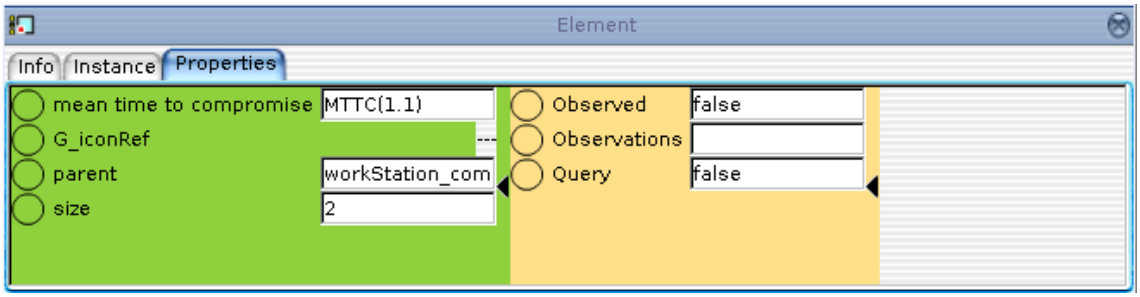

Figura 4.2: Proprietà dei nodi

Per quanto riguarda gli archi, dobbiamo differenziarli in base allo slice temporale a cui fanno riferimento. Nel caso del primo slice utilizzeremo *Arc* mentre nel secondo *TemporalArc*. In entrambi i casi, le proprietà degli archi sono le stesse illustrate nella Fig. 4.3, ovvero:

- **from**: nodo di partenza dell'arco.
- **to**: nodo di arrivo dell'arco.

Figura 4.3: Proprietà degli archi

Capitolo 5

Implementazione del Solver per SecuriDN

5.1 Introduzione

Il principale requisito del lavoro consiste nel garantire la flessibilità della traduzione, ovvero sviluppare un codice capace di generare uno script Octave sempre corretto, indipendentemente dalle diverse Reti Bayesiane Dinamiche (DBN) che si desidera modellare. Questo significa che il codice dovrà essere sufficientemente flessibile da adattarsi a una vasta gamma di configurazioni, dalle più semplici, come quella nella Fig. 5.1, a quelle estremamente complesse, come quella nella Fig. 5.2. Si dovrà garantire sempre un risultato coerente, privo di errori e ottenere uno script completo. In seguito, verranno trattate prima le descrizioni e i problemi relativi alla traduzione del modello (5.2) e successivamente gli aspetti implementativi (5.3).

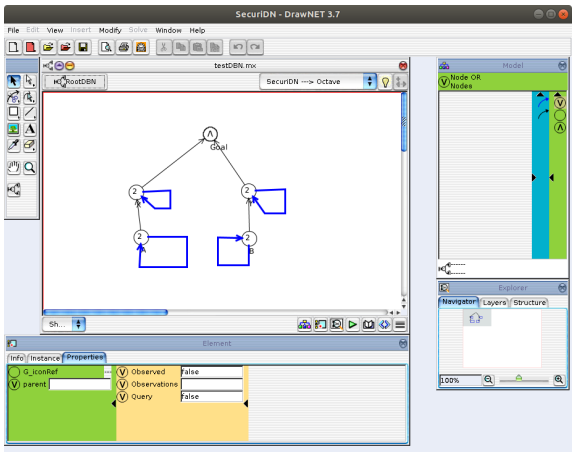

Figura 5.1: Modello di base in SecuriDN

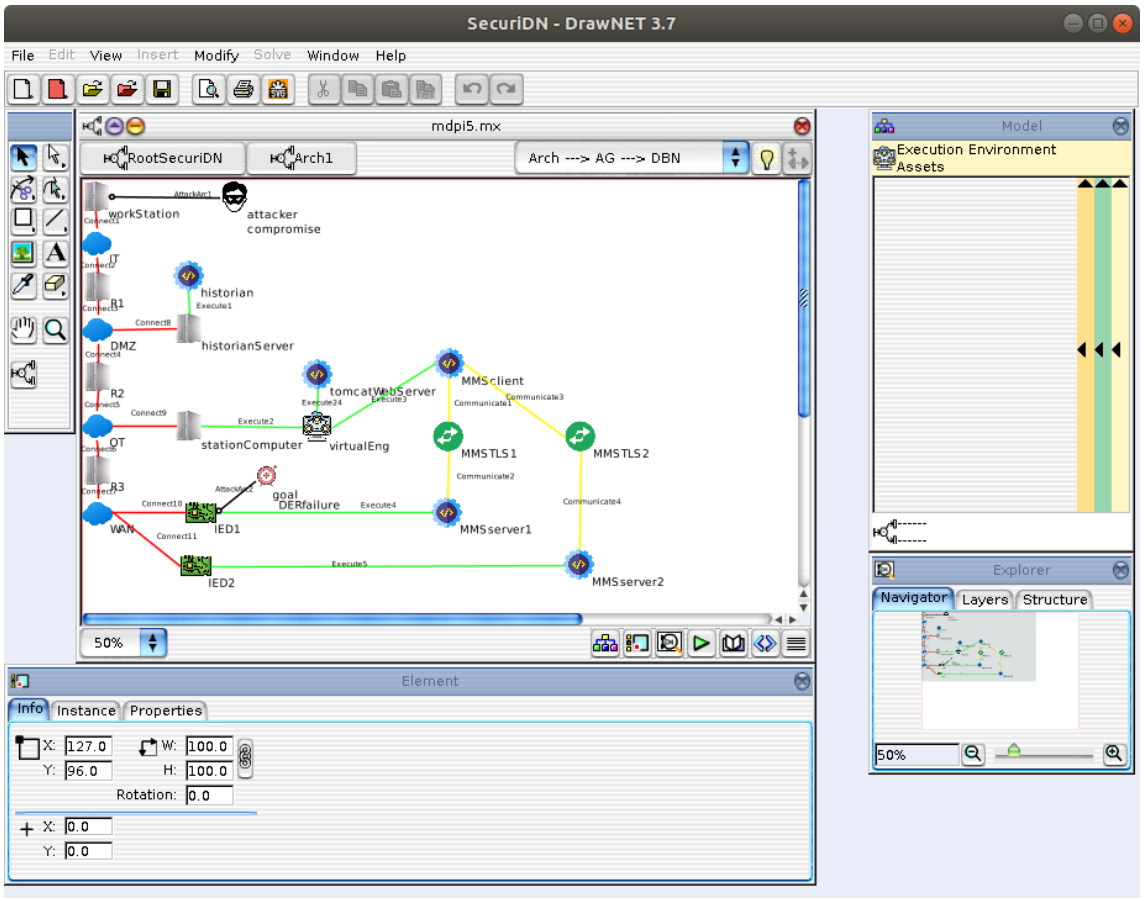

Figura 5.2: Modello completo di un processo d'attacco in SecuriDN

5.2 Problemi di traduzione del modello DBN

La traduzione di un modello DBN si articola in diverse componenti. Queste comprendono sia la rappresentazione dei nodi e delle connessioni tra di essi, sia il calcolo delle probabilità di attacco e le inferenze relative al modello. In questa sezione vengono affrontati i principali problemi teorici che ci aiuteranno per la fase di implementazione.

5.2.1 Definizione degli Stati e Nomi dei Nodi

È importante distinguere i diversi nodi che compongono una DBN in quanto a ciascuno di loro è associata una variabili del sistema che possono modellare la dinamica temporale di un fenomeno. Possiamo avere:

- **Nodi nascosti**: sono noti anche come hidden states e sono variabili non direttamente osservabili. Giocano un ruolo importante perchè rappresentano stati interni che influenzano il sistema tra i diversi slice temporali. Un

esempio di nodo nascosto può essere rappresentato dalle tecniche usate dall'attaccante, questo perchè non si ha la possibilità di sapere con certezza il suo comportamento.

- **Nodi osservabili**: sono variabili che possono essere misurate direttamente o osservate nel sistema. Anch'esse hanno un ruolo importante in quanto forniscono dei dati veri e propri. Sempre nel contesto dalla cyber sicurezza, questi possono essere rappresentati da eventuali allarmi attivati da un sistema di monitoraggio (dovuti ad esempio alla modifica di certi file di configurazione o all'esecuzione di comandi sospetti).

Queste differenze tra nodi osservabili e nodi nascosti influiscono direttamente su come viene costruito e utilizzato il modello di riferimento. I nodi osservabili forniscono dati empirici, che sono alla base delle stime probabilistiche, mentre i nodi nascosti consentono una rappresentazione più completa e complessa del sistema reale, perchè modellano variabili interne non osservabili che influenzano la dinamica del sistema. Oltre alla distinzione tra nodi nascosti e osservabili, il formalismo della DBN in SecuriDN(4.1) prevede la presenza di tre tipi di nodi:

- **Nodo Tecnica**: rappresenta una variabile di una specifica tecnica di attacco. La probabilità associata a un nodo tecnica può essere calcolata tramite il parametro *Mean Time To Compromise* (MTTC), che misura il tempo medio necessario per compromettere il nodo.

- **Nodo Analitico**: rappresenta una variabile legata a processi di analisi o misurazione. Nel modello di cyber sicurezza corrisponde a meccanismi di segnalazione di eventi sospetti. Tuttavia, tali meccanismi possono presentare un margine d'errore: un evento sospetto potrebbe essere segnalato erroneamente anche quando non lo è (falso positivo) oppure non venire segnalato nonostante sia effettivamente sospetto (falso negativo). Questa probabilità d'errore è descritta dal parametro *Probability of Failure* (PrF), assumendo che le probabilità di falsi positivi e negativi siano uguali e pari al valore di PrF. Questo parametro è fondamentale per la costruzione della CPT del nodo, in quanto tiene conto di entrambe le possibilità di errore.

- **Nodo AND/OR**: rappresenta una variabile per le condizioni logiche complesse che collegano diversi stati. Un nodo AND richiede che tutte le condizioni siano soddisfatte simultaneamente, mentre un nodo OR permette una transizione se almeno una delle condizioni è vera.

5.2.2 Definizione Connessioni Intra e Inter Slice

Nella costruzione della DBN, oltre alla distinzione tra i nodi, è importante anche comprendere come essi siano collegati tra loro. Gli archi tra i nodi definiscono le dipendenze probabilistiche, ovvero indicano se la probabilità di un nodo dipenda da un altro. In particolare, la presenza di uno o più archi che connettono nodi sorgente a un nodo destinazione stabilisce una dipendenza condizionale tra i nodi. Questo significa che la probabilità che il nodo destinazione assuma determinati valori è condizionata dai valori dei nodi sorgente. I collegamenti tra i nodi possono essere suddivisi in due categorie principali: archi intra-slice e archi inter-slice. Si hanno principalmente due tipi: archi intra-slice e archi inter-slice.
Un arco intra-slice rappresenta un collegamento tra due nodi all'interno dello stesso slice temporale ossia lo stato del sistema nello stesso momento. Questo indica che il valore di una variabile dipenderà direttamente dal valore di un'altra nello stesso slice di tempo.
Un arco inter-slice rappresenta una dipendenza tra due nodi in diversi slice temporale ossia lo stato del sistema in due momenti diversi. Ad esempio, un arco inter-slice tra un nodo e se stesso, indica che lo stato al tempo $t + 1$ dipenderà dal suo stato nel tempo t.

5.2.3 DBN, MTTC e PrF

Una volta tradotti gli elementi costitutivi della rete (nodi, archi, ...) possiamo definire la DBN. Per farlo ci servirà sapere anche quanti valori può assumere un nodo; nel nostro caso sarà sempre due (attivo o inattivo). Questa informazione, insieme ai parametri *Mean time to Compromise* (MTTC) e *Probabily of Failure* (PrF), è importante per completare la traduzione del nodo della DBN, in particolare per quanto riguarda i suoi aspetti stocastici.
Per ogni nodo tecnica dobbiamo considerare il MTTC, che rappresenta il tempo medio necessario per eseguire con successo una specifica tecnica. Questo tempo indica la frequenza relativa con cui un attacco (o evento critico) è atteso entro un determinato intervallo temporale. E' fondamentale conoscere il valore in quanto sarà necessario nel calcolo della probabilità corrente di successo di un attacco.
Per quanto riguarda i nodi analitici, dobbiamo considerare il PrF, che riflette la probabilità che un processo di analisi o monitoraggio associato al nodo possa produrre un risultato errato.
Infine, viene calcolato il tempo medio di transizione di una qualunque delle transizioni del modello, assumendo che possano avvenire in modo concorrente, questo grazie alla formula $PrAttDef = DeltaT./TMeanTech.$ Le ragioni di questa formula sono

complesse ed articolate e maggiori dettagli si possono trovare nel articolo [1]. Da questo dato è possibile calcolare, per tutti i nodi tecnica, la probabilità di avere un attacco.

5.2.4 Slice Temporali

Come menzionato in precedenza, il principale vantaggio nell'utilizzo delle DBN risiede nella loro capacità di rappresentare e analizzare l'evoluzione temporale di un sistema. Ogni slice temporale può essere interpretato come una fotografia del sistema in un determinato istante. Considerando che stiamo lavorando con DBN che incorporano processi semi-Markoviano di ordine $k = 2$, possiamo affermare che la probabilità condizionata che una variabile assuma un determinato valore (rappresentata attraverso la CPT) dipenderà non solo dallo slice corrente, ma anche da quello immediatamente precedente. Questo consente di modellare in modo più accurato le dinamiche temporali del sistema, tenendo conto delle influenze passate sulle condizioni attuali.
Nel primo slice temporale, inizializziamo le CPT per ogni nodo, che stabiliscono le probabilità condizionate dei nodi in funzione dei loro genitori all'interno dello stesso slice. In questo contesto, un nodo può dipendere da altri nello stesso slice o, se non ha genitori, solo da se stesso. La dimensione della matrice CPT dipenderà dal numero di genitori: per esempio, nel caso si avesse solo un genitore la dimensione sarà $2X2$, mentre nel caso di due genitori sarà $2X2X2$. Per convenzione, al tempo $t = 0$, tutti i nodi saranno inizializzati a inattivi/falsi, il che significa che la probabilità di un evento A sarà $Pr(A = 0) = 1$ e $Pr(A = 1) = 0$; in altre parole, per tutti i nodi è certo che al tempo $t = 0$ siano nello stato 0(inattivi) e perciò è impossibile che siano allo stesso tempo nello stato 1 (attivi).
Nel secondo slice temporale, le probabilità condizionate della CPT dei nodi non dipenderanno solo dai valori degli stati allo stesso istante ma eventualmente anche dai valori all'istante precedente. In particolare, ogni nodo al tempo $t = 1$ può dipendere dallo stato del nodo corrispondente nel tempo $t = 0$ e, se applicabile, anche dallo stato di altri nodi dello stesso istante. Si andrà ad utilizzare la variabile *CurPrAtt* rappresentante la probabilità di transizione, e quindi di successo di una tecnica, associata a un nodo da uno stato inattivo a uno attivo tra due slice temporali. Questa è definita grazie alla variabile calcolata precedentemente *PrAttDef* (5.3.4), che calcola la probabilità in base ai valori del modello.

5.3 Implementazione della traduzione

Come detto nell'introduzione al lavoro (4), una volta cliccata la lampadina verrà eseguito il metodo *execute*. Questo metodo richiede all'utente di inserire un intervallo di tempo, che rappresenta il periodo su cui verrà effettuata l'analisi della DBN. Successivamente, verrà richiesta la selezione della cartella contenente la libreria FullBNT, necessaria per utilizzare Octave. Dopo aver inserito questi dati, il metodo *createScriptOctave* viene invocato per generare uno script Octave basato sulla configurazione del modello. Inizialmente, lo script viene preparato tramite una chiamata al metodo *initializeOctave*, che ne definisce l'intestazione e carica le librerie necessarie. Successivamente, le proprietà dei nodi della rete vengono estratte e memorizzate in un attributo denominato *nodes* tramite il metodo *getNodes*. In questa fase, il metodo aggiunge allo script le informazioni dettagliate, che saranno illustrate in seguito. Infine, il contenuto dello script viene scritto su file tramite il metodo *writeFile*, completando così il processo di generazione.

5.3.1 Inizializzazione dell'Ambiente e delle Librerie

La prima sezione dello script Octave generato definisce l'inizializzazione dell'ambiente e il caricamento delle librerie necessarie. Questo passaggio è fondamentale soprattutto per il caricamento della libreria *BNT-1.0.7* tramite cui si hanno tutte le funzioni per la manipolazione delle tabelle di probabilità condizionata (CPT) e DBN.
Dobbiamo garantire che Octave possa accedere ai pacchetti e alle librerie richieste e inoltre che l'ambiente di esecuzione sia configurato correttamente per evitare errori durante l'elaborazione dei dati e l'esecuzione delle funzioni.
Viene richiamato il metodo *initializeOctave*, che genera una stringa contenente tutte le informazioni precedentemente menzionate, come quelle presenti nel Listing 5.1.

```
1  #!/usr/bin/octave -qf
2  clear;
3
4  oldpwd=pwd;
5  cd /home/nicomariv/FullBNT-1.0.7-Bis;
6  addpath(genpath(pwd));
7  cd(oldpwd);
8
9  pkg load io
10
11 args={"mdpi5.m.png", "mdpi5.m.csv"};
```

Listing 5.1: Porzione script Octave: Inizializzazione e Librerie

5.3.2 Estrazione dei nodi della rete

Come detto precedentemente, viene utilizzata una struttura chiamata nodes, che viene popolata tramite il metodo *getNodes* (vedi Listing 5.2). All'interno di questo metodo, si itera su tutti gli elementi della DBN, verificando che il tipo di ciascun elemento (*ElementType*) corrisponda a un nodo. Una volta verificato che si tratti di un nodo, tutte le proprietà descritte nel formalismo (4.1) vengono salvate.
L'unica differenza da tenere in considerazione è che, tramite la proprietà *CPT*, possiamo avere due situazioni. Se il nodo è di tipo tecnica, si dovrà estrarre il MTTC; nel caso di nodo analitico, invece, si dovrà estrarre il PrF. Questa struttura verrà poi utilizzata ogni volta che sarà necessario fare riferimento ai nodi.
Ogni nodo del formalismo utilizzato sarà caratterizzato da un campo *Observed*. Questo campo ci permetterà di differenziare i nodi osservabili da quelli non osservabili.

```
if (elementType.getId().contains("Node")) {
    Matcher matcher;
    Pattern PatternPrF = Pattern.compile("PrF\\((\\d+(\\.\\d+)?)\\)");
    Pattern PatternMTTC = Pattern.compile("MTTC\\((\\d+(\\.\\d+)?)\\)");
    Map<String, Object> nodeProperties = new HashMap<>();

    StringPropertyValue query = (StringPropertyValue) elementInstance.getPropertyValue("query");
    StringPropertyValue observed = (StringPropertyValue) elementInstance.getPropertyValue("observed");
    StringPropertyValue parent = (StringPropertyValue) elementInstance.getPropertyValue("parent");
    IntegerPropertyValue size = (IntegerPropertyValue)elementInstance.getPropertyValue("size");

    nodeProperties.put("observerd",observed.toString());
    nodeProperties.put("query", query.toString());
    nodeProperties.put("parent", parent.toString());
    nodeProperties.put("type", observed.toString().equals("true") ? "obs" : "hstate");
    nodeProperties.put("size",size != null ? size.getValue(): 2);

    if(elementType.getId().equals("NodeAND") || elementType.getId().equals("NodeOR")){
        nodeProperties.put("node_type", elementType.getId().equals("NodeAND") ? "and" : "or");
    }else{
        StringPropertyValue cptParameter = (StringPropertyValue)elementInstance.getPropertyValue("CPT");

        if(cptParameter.toString().contains("MTTC")){
            matcher = PatternMTTC.matcher(cptParameter.toString());
            if (matcher.find()) {
                nodeProperties.put("node_type", "tech");
                nodeProperties.put("MTTC", Double.parseDouble(matcher.group(1)));
            }
        }else if(cptParameter.toString().contains("PrF")){
            matcher = PatternPrF.matcher(cptParameter.toString());
            if (matcher.find()) {
                nodeProperties.put("node_type", "analytical");
                nodeProperties.put("PrF", Double.parseDouble(matcher.group(1)));
            }
        }
    }
    nodes.put(elementInstance.getId(),nodeProperties);
}
```

Listing 5.2: SolverFilterDBN: getNodes()

Nel codice viene invocato il metodo *extracNode*, che per ogni elemento, controlla se il nodo è di tipo nascosto oppure osservabile e lo traduce nel modo appropriato come mostrato in Listing 5.3.

```
h_states = {'MMSTLS1_unauthCmdMsg', ... , 'MMSclient_credAcc'};

names=[h_states];
n=length(names);
```

Listing 5.3: Porzione script Octave: Nodi della rete

Nel vettore *names* verranno salvati tutti gli *Id* dei nodi che ci serviranno in seguito per la creazione della DBN.

5.3.3 Gestione delle connessioni

Per tradurre gli archi intra-slice, verrà utilizzato il metodo *extractIntraSlice* (vedi Listing 5.4); facendo riferimento al formalismo (4.1) sappiamo che questi archi avranno l'*ElemenType* uguale ad *Arc* e per ottenere gli *Id* dei nodi collegati ci servirà la proprietà *ElementRefPropertyValue* tramite cui otterremo il nodo di partenza e quello di arrivo.

```
if(elementType.getId().equals("Arc")){
    ElementRefPropertyValue fromRef = (ElementRefPropertyValue)elementInstance.getPropertyValue("from");
    ElementRefPropertyValue toRef = (ElementRefPropertyValue)elementInstance.getPropertyValue("to");
    if((fromRef == null || toRef == null) || (fromRef.toString().length() < 1 || toRef.toString().length() < 1)
         )
        continue;
    ElementInstance fromInstance = dbn.getSubElementByPath(fromRef.toString());
    ElementInstance toInstance = dbn.getSubElementByPath(toRef.toString());
    String from = fromInstance.getId();
    String to = toInstance.getId();

    intracBuilder.append("\t'").append(from).append("','").append(to).append("';\n");
}
```

Listing 5.4: SolverFilterDBN: extractIntraSlice()

Una volta eseguito il metodo, verrà generata una lista di coppie come quella presente nel Listing 5.5, dove il primo elemento sarà il nodo di partenza mentre il secondo quello di arrivo.

```
intrac = {
   'MMSTLS1_unauthCmdMsg','IED1_DERfailureOR';
   'workStation_compromise','DMZ_scanIP';
   ...
   'tomcatWebServer_bruteForce','virtualEng_escapeHost';
   'workStation_compromise','historianServer_remoteShellOR'
};
```

Listing 5.5: Porzione script Octave: Archi intra-slice

Il procedimento di traduzione per gli archi inter-slice è analogo a quello visto per gli archi intra-slice: viene invocato il metodo *extractInterSlice* (vedi Listing 5.6) solo che questa volta il controllo degli *ElementType* deve essere uguale a *TemporalArc*.

```
if(elementType.getId().equals("TemporalArc")){
    ElementRefPropertyValue fromRef = (ElementRefPropertyValue)elementInstance.getPropertyValue("from");
    ElementRefPropertyValue toRef = (ElementRefPropertyValue)elementInstance.getPropertyValue("to");
    if((fromRef == null || toRef == null) || (fromRef.toString().length() < 1 || toRef.toString().length() < 1)
         )
        continue;
    ElementInstance fromInstance = dbn.getSubElementByPath(fromRef.toString());
    ElementInstance toInstance = dbn.getSubElementByPath(toRef.toString());
    String from = fromInstance.getId();
    String to = toInstance.getId();

    intercBuilder.append("\t'").append(from).append("','").append(to).append("';\n");
}
```

Listing 5.6: SolverFilterDBN: extractInterSlice()

Anche in questo caso, verrà generata una lista di coppie contenente gli *Id* dei nodi che fanno parte degli archi inter-slice come, ad esempio, quella presente nel Listing 5.7.

```
interc = {
    'IED1_DERfailure','IED1_DERfailure';
    'historianServer_remoteShell','historianServer_remoteShell';
    ...
    'historianServer_addSSHkey','historianServer_addSSHkey';
    'MMSclient_modAuthProc','MMSclient_modAuthProc'
};
```

Listing 5.7: Porzione script Octave: Archi inter-slice

Questi sono molto utili poiché ci permettono di definire le dipendenze probabilistiche di un nodo.

5.3.4 Costruzione della DBN e calcolo della Probabilità di Attacco

Il metodo *createBnetAttackProbability* (vedi Listing 5.8) esegue tutte le operazioni necessarie: inizialmente costruisce la DBN, utilizza la proprietà CPT per determinare il MTTC oppure il PrF e infine procede al calcolo delle probabilità di attacco.

```
for (Map.Entry<String, Map<String, Object>> entry : nodes.entrySet()) {
    String nodeId = entry.getKey();
    Map<String, Object> nodeProperties = entry.getValue();
    String nodeType = (String) nodeProperties.get("node_type");
    if("and".equals(nodeType) || "or".equals(nodeType)){
        sizeNodeBuilder.append("2 ");
    }else{
        Integer size = (Integer) nodeProperties.get("size");
        sizeNodeBuilder.append(size).append(" ");
    }
    if("tech".equals(nodeType)){
        double mttc = (Double) nodeProperties.get("MTTC");
        attackBuilder.append(nodeId).append("=bnet.names('").append(nodeId).append("');\n");
        attackBuilder.append("TMeanTech(").append(nodeId).append(")= ").append(mttc).append(";\n\n");
    }else if("analytical".equals(nodeType)){
        double prf = (Double) nodeProperties.get("PrF");
        attackBuilder.append(nodeId).append("=bnet.names('").append(nodeId).append("');\n");
        attackBuilder.append("PrTech(").append(nodeId).append(")= ").append(prf).append(";\n\n");
    }else{
        attackBuilder.append(nodeId).append("=bnet.names('").append(nodeId).append("');\n\n");
    }
}
sizeNodeBuilder.deleteCharAt(sizeNodeBuilder.length()-1).append("];\n\n");
attackBuilder.append("DeltaT=1/(sum(1./TMeanTech(TMeanTech>0)));\nPrAttDef = DeltaT./TMeanTech; \n\n");
```

Listing 5.8: SolverFilterDBN: createBnetAttackProbability()

Dopo aver eseguito il metodo, otterremo una porzione di script simile a quella mostrata nel Listing 5.9. Inoltre, possiamo notare che il nodo *historianServer_remoteShellOR* non ha nessun valore associato, poiché è un nodo OR (che potrebbe anche essere un nodo AND). Questo perché la probabilità di un attacco dipende dai nodi collegati.

```
[intra, names] = mk_adj_mat(intrac, names, 1);
inter = mk_adj_mat(interc, names, 0);
ns = [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2];

bnet = mk_dbn(intra, inter, ns, 'names', names);

MMSTLS1_unauthCmdMsg=bnet.names('MMSTLS1_unauthCmdMsg');
TMeanTech(MMSTLS1_unauthCmdMsg)= 1.1;
...

historianServer_remoteShellOR=bnet.names('historianServer_remoteShellOR');
...

virtualEng_suspiciousCmd=bnet.names('virtualEng_suspiciousCmd');
PrTech(virtualEng_suspiciousCmd)= 0.2;
...

DeltaT=1/(sum(1./TMeanTech(TMeanTech>0)));
PrAttDef = DeltaT./TMeanTech;
```

Listing 5.9: Porzione script Octave: DBN e MTTC/PrF

5.3.5 Definizione CPT per Primo Slice Temporale

Il metodo *extractSliceOne* calcola il primo slice che corrisponde ai valori della CPT allo slice $t = 0$. Si può notare che dati n genitori, avremo 2^n configurazioni. Tra queste, la prima configurazione, in cui tutti i nodi sono inattivi, garantirà sempre che l'evento non si verifichi, mentre per le altre configurazioni la probabilità sarà diversa. Per i nodi analitici dovremo andare a considerare il parametro PrF e quindi costruire la CPT di conseguenza. Nella configurazione iniziale, la probabilità che il nodo funzioni correttamente sarà rappresentata da $1 - PrF$, mentre la probabilità di fallimento sarà data da PrF. In questo modo, la CPT rifletterà accuratamente la possibilità di errore del nodo analitico, tenendo conto della probabilità di fallimento e di successo del processo di analisi o monitoraggio associato. Per quanto riguarda i nodi logici useremo un funzione che calcolerà la CPT basandosi sui genitori e la tabella di verità dell'operazione logica utilizzata.
Eseguendo il metodo, si otterrà una porzione di script come quello nel Listing 5.10 dove viene inizializzata la CPT per poi assegnarla alla DBN in relazione al nodo preso in considerazione. Una volta assegnati i valori all'interno della DBN possiamo azzerare la CPT per poterla nuovamente usare per i nodi successivi.

```
%%%%%%%% ------- slice 1 --------

%node MMSTLS1_unauthCmdMsg(id=MMSTLS1_unauthCmdMsg) slice 1
%parent order:{'MMSTLS1_AITM'}
cpt(1,:)=[1.0, 0.0];
cpt(2,:)=[0.0, 1.0];
cpt1=mk_named_CPT({'MMSTLS1_AITM','MMSTLS1_unauthCmdMsg'},names, bnet.dag, cpt);
bnet.CPD{bnet.names('MMSTLS1_unauthCmdMsg')}=tabular_CPD(bnet,bnet.names('MMSTLS1_unauthCmdMsg'),'CPT',cpt1);
clear cpt;clear cpt1;

...

%node historianServer_remoteShellOR(id=historianServer_remoteShellOR) slice 1
%parent order:{'historianServer_addSSHkey','workStation_compromise'}
bnet.CPD{bnet.names('historianServer_remoteShellOR')}=boolean_CPD(bnet,bnet.names('historianServer_remoteShellOR'),'named', 'any');

...

%node virtualEng_suspiciousCmd(id=virtualEng_suspiciousCmd) slice 1
%parent order:{'virtualEng_escapeHost'}
cpt(1,:)=[0.8, 0.2];
cpt(2,:)=[0.2, 0.8];
cpt1=mk_named_CPT({'virtualEng_escapeHost','virtualEng_suspiciousCmd'},names, bnet.dag, cpt);
bnet.CPD{bnet.names('virtualEng_suspiciousCmd')}=tabular_CPD(bnet,bnet.names('virtualEng_suspiciousCmd'),'CPT',cpt1);
clear cpt;clear cpt1;
```

Listing 5.10: Porzione script Octave: Primo Slice

5.3.6 Definizione CPT per Secondo Slice Temporale

Il metodo *extractSliceTwo* calcola il secondo slice, che rappresenta la transizione di uno stato verso uno stato successivo. Occorre considerare come ciascun nodo possa dipendere sia dal suo stato al passo precedente sia da altri nodi presenti nella rete andando ad ottenere una porzione di script come quello nel Listing 5.11.

```
%%%%%%%%% ------- slice 2 --------

%node MMSTLS1_unauthCmdMsg(id=MMSTLS1_unauthCmdMsg) slice 2
%parent order:{MMSTLS1_AITM;MMSTLS1_unauthCmdMsg}
CurPrAtt = PrAttDef(MMSTLS1_unauthCmdMsg);
cpt(1,1,:)=[1.0, 0.0];
cpt(1,2,:)=[1-CurPrAtt, CurPrAtt];
cpt(2,1,:)=[0.0, 1.0];
cpt(2,2,:)=[0.0, 1.0];
cpt1=mk_named_CPT_inter({'MMSTLS1_unauthCmdMsg', 'MMSTLS1_AITM','MMSTLS1_unauthCmdMsg'},names, bnet.dag, cpt, [2]);
bnet.CPD{bnet.eclass2(bnet.names('MMSTLS1_unauthCmdMsg'))}=tabular_CPD(bnet,n+bnet.names('MMSTLS1_unauthCmdMsg'),'CPT',cpt1);
clear cpt;clear cpt1;

...

%node workStation_compromise(id=workStation_compromise) slice 2
%parent order:{workStation_compromise}
CurPrAtt = PrAttDef(workStation_compromise);
cpt(1,:)=[1-CurPrAtt, CurPrAtt];
cpt(2,:)=[0.0, 1.0];
bnet.CPD{bnet.eclass2(bnet.names('workStation_compromise'))}=tabular_CPD(bnet,n+bnet.names('workStation_compromise'),'CPT',cpt);
clear cpt;

...
```

Listing 5.11: Porzione script Octave: Secondo Slice

5.4 Analisi della DBN

Durante l'analisi della DBN, sfruttiamo diverse proprietà del formalismo (4.1) per impostare correttamente l'inferenza. Il metodo *extractAnalysis* (vedi Listing 5.12) avvia la funzione che esegue l'inferenza sul modello appena definito e aggiunge una parte finale che prepara i dati per il grafico dei nodi di interesse, rendendo più semplice visualizzare i risultati dell'inferenza.

```
if (elementType.getId().contains("DBN")){
    var fully_factorizedDBN = (StringPropertyValue)elementInstance.getPropertyValue("fully_factorized");
    var inf_algorithmDBN = (StringPropertyValue)elementInstance.getPropertyValue("inf_algorithm");
    var inf_typeDBN = (StringPropertyValue)elementInstance.getPropertyValue("inf_type");
    var time_stepDBN = (IntegerPropertyValue)elementInstance.getPropertyValue("time_step");

    inf_algorithm = inf_algorithmDBN.toString();
    fully_factorize = fully_factorizedDBN.equals("false") ? 0 : 1;
    inf_type = inf_typeDBN.equals("smoothing") ? 0 : 1;
    time_step = Integer.parseInt(time_stepDBN.toString());
    queryNode = this.getQueryNode();

    ...
```

Listing 5.12: SolverFilterDBN: extractAnalysis()

5.5 Risultati dell'analisi

Il solver sviluppato permette di generare uno script in Octave che, una volta eseguito, produce un file CSV contenente le probabilità di successo di un cyber attacco su ciascun nodo della rete per ogni istante di tempo analizzato. Oltre al file CSV, lo script produce un grafico che mostra l'andamento di tali probabilità per i nodi di

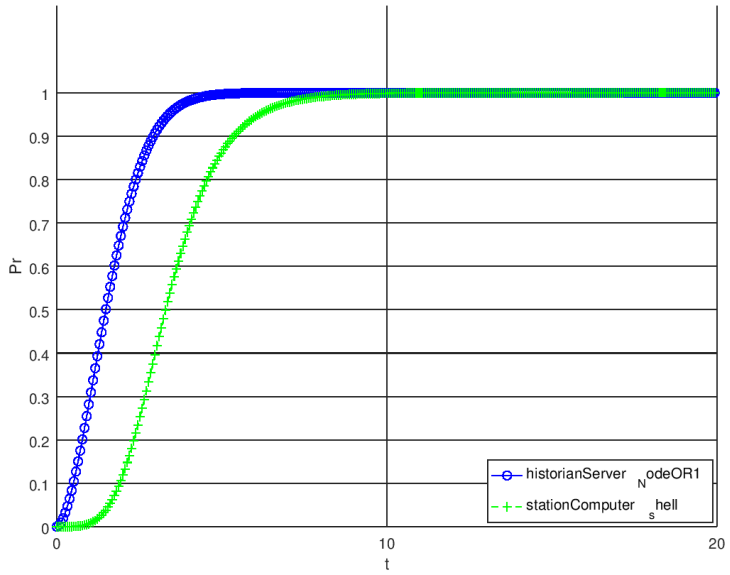

Figura 5.3: Grafico generato dallo script Octave relativo al modello mdpi5.md

interesse, come illustrato nella Fig. 5.3. Il file CSV è quindi inviato a un modulo di rilevamento di cyber attacchi come mostrato nella Fig. 1.2. Questo modulo permette di valutare l'esito dell'attacco e di analizzare la dinamica di esso sulla rete. I risultati vengono visualizzati su una dashboard per permettere ad un operatore di valutare le probabilità di attacco sui diversi componenti della rete informatica e la loro evoluzione nel tempo. Questo approccio permette di monitorare e studiare l'impatto degli attacchi sulla rete, fornendo così una base per sviluppare contromisure efficaci.

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Conclusioni sullo studio guidato

In questo lavoro è stato sviluppato un solver che permette di tradurre grafi di attacco in script Octave, facilitando l'analisi delle Reti Bayesiane Dinamiche (DBN) in ambito di sicurezza informatica per l'infrastruttura energetica. Il lavoro ha richiesto una comprensione del funzionamento delle DBN e l'integrazione con il linguaggio Octave per calcolare la probabilità di successo di un attacco. Il risultato finale è uno strumento flessibile e dinamico, capace di adattarsi a diverse configurazioni di rete, offrendo un supporto significativo per il rilevamento preventivo di minacce informatiche.

6.2 Limitazioni e sviluppi futuri

Un problema significativo che potrebbe verificarsi è l'esaurimento della memoria durante l'esecuzione dello script. Ad esempio, prendendo in considerazione il modello *mdpi5*, con nodi dipendenti tra di loro (*ff=0*) e utilizzando l'algoritmo di inferenza junction tree (*ec='JT'*), si rischia di avere la saturazione della memoria, rendendo impossibile portare a termine l'analisi per modelli di grandi dimensioni. Una soluzione potrebbe essere l'uso dell'algoritmo approssimato BK, che mira proprio a limitare il consumo della memoria, anche se non sempre è efficace.
Questo lavoro può avere ulteriori sviluppi futuri: un'estensione interessante potrebbe essere l'integrazione della gestione delle evidenze, che consentirebbe di arricchire l'analisi della DBN con esperimenti condotti su dati input variabili, simulando scenari più complessi o realistici.

Bibliografia

[1] Davide Cerotti, Daniele Codetta Raiteri, Giovanna Dondossola, Lavinia Egidi, Giuliana Franceschinis, Luigi Portinale, Davide Savarro, Roberta Terruggia, et al. Securidn: a customizable gui generating cybersecurity models for der control architectures. In *CEUR WORKSHOP PROCEEDINGS*, 2024.

[2] J. W. Eaton. Octave, 2022. Accesso nel luglio 2024. Disponibile all'indirizzo `https://www.gnu.org/software/octave/`.

[3] Dimitris Margaritis et al. *Learning Bayesian network model structure from data*. PhD thesis, School of Computer Science, Carnegie Mellon University Pittsburgh, PA, USA, 2003.

[4] Verizon. 2023 data breach investigations report: Utilities snapshot. Verizon, 2023. Accesso nel luglio 2024. Disponibile all'indirizzo `https://www.verizon.com/business/resources/reports/2023-dbir-utilities-snapshot.pdf`.

[5] Junxiao Shi, Sara Saleem, Mathias Gibbens, Harsha vardhan Rajendran, TK Balaji Prasad, Nupur Maheshwari, Paul Mueller, Babak Yadegari, Matthew Ward, Paul Jennas II, et al. Csc 566, computer security research reports. 2012.

[6] Rafiullah Khan, Peter Maynard, Kieran McLaughlin, David Laverty, and Sakir Sezer. Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid. In *4th International Symposium for ICS & SCADA Cyber Security Research 2016*, pages 53–63. BCS, 2016.

[7] Kishor S Trivedi. *Probability & statistics with reliability, queuing and computer science applications*. John Wiley & Sons, 2008.

[8] Kishor S Trivedi and Andrea Bobbio. *Reliability and availability engineering: modeling, analysis, and applications*. Cambridge University Press, 2017.

[9] Michael Kofler. *Ubuntu 18.04*. Universitätsbibliothek Tübingen, 2018.

[10] Kevin P. Murphy. The bayes net toolbox for matlab. Department of Computer Science, University of California, Berkeley, Berkeley, CA, 94720-1776, October 2001. Accesso nel luglio 2024. Disponibile all'indirizzo `https://www.cs.ubc.ca/~murphyk/Papers/bnt.pdf`.

[11] Marco Gribaudo, Daniele Codetta-Raiteri, and Giuliana Franceschinis. Draw-net, a customizable multi-formalism, multi-solution tool for the quantitative evaluation of systems. In *Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*, pages 257–258. IEEE, 2005.

[12] Daniele Codetta-Raiteri, Giuliana Franceschinis, Marco Gribaudo, et al. Defining formalisms and models in the draw-net modelling system. In *Int. Workshop on Modelling of Objects, Components and Agents*, pages 123–144, 2006.

[13] Daniele CODETTA RAITERI et al. Uml class diagrams supporting formalism definition in the draw-net modeling system. In *Technical Report, DiSIT, Istituto di Informatica, Univ. del Piemonte Orientale*. 2019.