# Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Analisi ibrida delle minacce


Relatori
prof. Riccardo Sisto
prof. Fulvio Valenza

Candidato
Enzo Maria Bisceglia

Anno accademico 2020/2021

# Sommario

Il threat modeling è un processo attraverso il quale è possibile indentificare ed enumerare potenziali minacce e proporre adeguati meccanismi di mitigazione. Questo rappresenta, nel contesto IT, un approccio strutturato alla sicurezza di una risorsa o di un insieme di risorse che possiedono un certo valore.

Al giorno d'oggi, con la rapida diffusione dei sistemi cyberfisici, la validità di questo approccio viene messa in discussione.

La causa è da ricercarsi nella sempre più numerosa partecipazione e dall' eterogenea fattura degli attori in gioco: dalle componenti fisiche responsabili dell' attuazione di determinati movimenti meccanici, ai terminali che inoltrano i comandi relativi a tali routine, fino al personale incaricato della manutenzione di questi stessi terminali e di tali mezzi fisici. Un threat model efficace deve essere in grado di riconoscere non solo le debolezze intrinseche della singola entità ma anche quelle che derivano dall' interazione, diretta e non, con le restanti componenti del sistema, indipendentemente che si tratti di relazioni di tipo umano, cyber o fisico.

Lo scopo della tesi è quello di studiare ed estendere le funzionalità di un threat model innovativo, *Tameless*, in grado di offrire una visione olistica del sistema in analisi attraverso un tipo di descrizione che soddisfa le caratteristiche elencate sopra e a un design improntato al paradigma della programmazione logica.

Per farlo è stato necessario dapprima comprendere e approfondire diversi aspetti legati alla natura di questo threat model e successivamente ragionare, un po' seguendo lo spirito iterativo cui si ispira la logica di funzionamento di quest'ultimo, su una soluzione che portasse dei benefici pratici all'utilizzatore finale.

Il risultato è un tool capace di tradurre gli output generati da *Tameless* nella forma visiva di un grafo d'attacco e fornire, ove necessario, le contromisure da mettere in campo per mitigare le criticità emerse.

# Indice

# Capitolo 1

# Introduzione

La quarta rivoluzione industriale rappresenta il contesto in cui il processo di integrazione delle nuove tecnologie nei diversi livelli della società raggiunge un punto di svolta.
Gli Smart Systems, fra le espressioni di maggiore impatto di questa era di innovazioni e colonne portanti dell'Industry 4.0, hanno raggiunto una maturità tale da rispondere positivamente alle esigenze di pressoché qualunque ambito applicativo: industria, sanità e automotive per citarne alcuni. L'Internet of Things, punto di congiunzione tra il mondo fisico e quello digitale, si è imposto come il paradigma chiave alla base dell' architettura di tali sistemi. Nascono nuovi impieghi e, nel frattempo, gli ambienti di lavoro esistenti si evolvono per adattarsi al cambiamento mentre sullo sfondo si erigono edifici e città intelligenti.
In questo contesto esistono tuttavia una serie di punti d'attenzione sotto l'aspetto della cybersecurity che, se non considerati approfonditamente, possono contribuire a creare un rischio per la sicurezza delle persone che vi stanno attorno.

A causa della complessità, della natura delle componenti e dell'ampia estensione, le falle di sicurezza all'interno di questi ecosistemi, possono manifestarsi su diversi livelli e avere un impatto e delle conseguenze molto differenti.
La sensoristica IoT così come i dispositivi di attuazione o di interfaccia con altre parti dell'infrastruttura possono rappresentare dei potenziali punti d' ingresso per un attaccante in grado di far leva su vulnerabilità hardware e software rilasciate con i firmware proprietari; l'eterogeneità negli standard di comunicazione tra le componenti di raccolta dei dati e le unità di elaborazione è un' altra criticità che può minare l'integrità e la riservatezza delle informazioni trasmesse; in base al contesto e alla potenza di calcolo richiesti dall'elaborazione, i dati collezionati possono essere processati in maniera centralizzata o decentralizzata, secondo i dettami del Cloud o

dell' Edge Computing, complicando ulteriormente lo scenario sotto esame.
Un capitolo a parte andrebbe infine speso nel considerare il ruolo del fattore umano all' interno del contesto ciberfisico, ma è fin qui già abbastanza evidente quanto sia decisivo, ai fini dell' analisi di sicurezza, descrivere in maniera esauriente scenari di questo tipo.

Oltre ad avere chiare le dinamiche legate a questi aspetti, la robustezza e la resilienza di questi sistemi dipendono anche da altri tipi di garanzie, come quelle relative alle più basilari proprietà di sicurezza.
Dall' assicurare l' integrità e la riservatezza dei dati trasmessi tra le varie componenti, all' accertare che ciascun entità coinvolta in un' interazione sia autentica ed autorizzata, alla capacità di proteggere in maniera continuativa e "distribuita" il sistema da eventuali intrusioni o ancora a quella di rispondere tempestivamente ad un attacco in maniera definitiva o anche solo contenitiva onde evitare l'"effetto cascata" sulle restanti componenti.
I punti di attenzione sono diversi e vulnerabilità intrinseche, inaffidabilità delle reti e inesperienza sono solo alcuni degli esempi di fragilità che caratterizzano la natura fisica, cyber e umana delle componenti di questi sistemi.

Il threat modeling rappresenta una possibile soluzione a queste problematiche. Esso definisce, in estrema sintesi, un approccio strutturato alla modellazione dei diversi aspetti concernenti la sicurezza di una o più risorse d' interesse. Sono molteplici i modelli descritti negli articoli dedicati a questo tema e i framework sviluppati in questa direzione. Le declinazioni di questo processo variano a seconda del dominio applicativo e della natura della componente (o delle componenti) del sistema oggetto di analisi, nello specifico: cyber, fisica e umana.
Ciò che non è stato fino ad ora documentato in letteratura, è un approccio che descriva la modellizzazione di quei sistemi in cui sono presenti contemporaneamente tutte e tre gli elementi sopra citati, fornendo un resoconto dello stato di sicurezza globale, oltre che dall' analisi puntuale delle sue componenti, anche dalle relazioni che intercorrono tra le stesse.

L'obiettivo di questa tesi è duplice: da un lato si vuole presentare il threat model che sopperisce a questa mancanza, Tameless, descrivendone il funzionamento e approfondendone gli aspetti caratterizzanti; dall' altro si vuole parlare del lavoro compiuto per estendere le funzionalità di questo threat model così da offire maggiore supporto all' analista di sicurezza. Le funzionalità aggiunte comprendono la possibilità di generare un grafo d'attacco sulla base dell' analisi condotta e di ottenere il resoconto delle mitigazioni da attuare per rimuovere le criticità individuate.

La tesi è suddivisa nel modo seguente:

- **Capitolo 2**: presenta gli aspetti più rilevanti e lo stato dell'arte per quel che riguarda le tecnologie appartenenti alla categoria degli Smart Systems, descrive alcuni degli incidenti accaduti nella storia recente ed espone le possibili cause che determinano le sfide di cybersecurity più critiche.

- **Capitolo 3**: descrive lo stato dell' arte nel campo dell'individuazione delle minacce e approfondisce i concetti chiave relativi ai due strumenti studiati durante la stesura della tesi ossia il threat modeling ed i grafi d'attacco.

- **Capitolo 4**: enuncia l'obiettivo di questa tesi, spiega le motivazioni che hanno determinato la direzione del lavoro ed introduce alla parte centrale di questo documento.

- **Capitolo 5**: è la prima delle due parti in cui è suddiviso il corpo centrale della tesi. Qui si presenta in maniera teorica il threat model originale e viene descritto a grandi linee il processo che ha condotto all' implementazione e all'integrazione delle nuove funzionalità all' interno di quest'ultimo.

- **Capitolo 6**: discute in maniera più tecnica il threat model, presenta il codice che implementa le funzionalità introdotte nel capitolo precedente, le librerie e gli strumenti di supporto allo sviluppo del tool.

- **Capitolo 7**: descrive il processo di validazione tramite due casi d'uso.

- **Capitolo 8**: presenta le conclusioni, le considerazioni sul lavoro svolto e i possibili sviluppi futuri.

# Capitolo 2

# Smart Systems

In questo capitolo verranno esposti gli aspetti chiave relativi agli Smart Systems, le tecnologie e i paradigmi che sfruttano per adattarsi ai diversi campi applicativi e le fragilità che le caratterizzano dal punto di vista della sicurezza.

## 2.1 Concetti chiave

Uno Smart System è per definizione un sistema in grado di svolgere le sue funzioni sfruttando una rappresentazione digitale del suo contesto di lavoro [1].
Tale contesto viene costruito raccogliendo, tramite una rete di sensori, i dati provenienti dal mondo esterno, integrandoli con una base di conoscenza e fornendo alle unità di controllo e attuazione le informazioni necessarie a garantire una gestione ottimale degli input e degli output del sistema.
Queste tecnologie trovano applicazione nel settore della salute, dei trasporti, della produzione di beni di consumo e nelle varie infrastrutture critiche presenti ai diversi livelli della società e la loro diffusione ha avuto un riscontro estremamente positivo nei rapporti tra fornitori e clienti in termini di efficienza, affidabilità e riduzione del costo dei servizi offerti [2].

Nel settore industriale la transizione si è concretizzata nel passaggio dalla storica "catena del valore" ad una più efficiente e connessa "rete del valore".
Qui, una riorganizzazione dei processi produttivi improntata al paradigma intelligente, ha consentito di rispondere più velocemente ed in maniera sempre più efficace alle mutevoli esigenze del mercato [3]. Le Smart Factory vedono le aziende superare i propri confini con l'ausilio di tecnologie come i sistemi cyber-fisici e l' IoT, delle

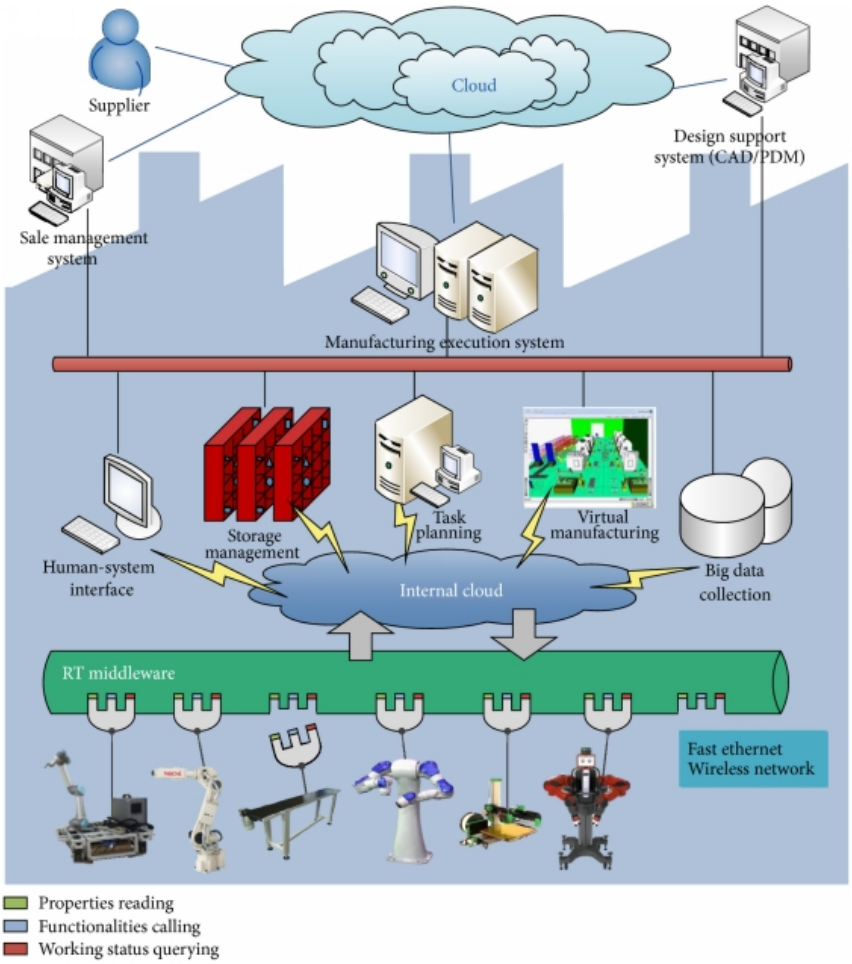



Figura 2.1: Architettura di alto livello di una Smart Factory. Fonte Wikipedia.

tecniche legate alla business intelligence e delle possibilità di confronto offerte dalla rete globale.

Nel settore sanitario è in corso una lenta ma progressiva infiltrazione delle recenti innovazioni ICT all' interno del tessuto dei macchinari ad uso medico. [4].
Gli strumenti per la diagnosi, il monitoraggio e la terapia a distanza dei malati hanno consentito ai pazienti di beneficiare di migliori trattamenti sotto gli aspetti della personalizzazione e della puntualità nella somministrazione delle cure. L' infrastruttura IoT sottostante offre la possibilità di comunicare in tempo reale le azioni da intraprendere sulla base dell'andamento istantaneo di determinate condizioni mediche o di raccogliere, tramite la sensoristica apposita, dati nel breve e lungo termine,

da sottoporre a successive analisi.

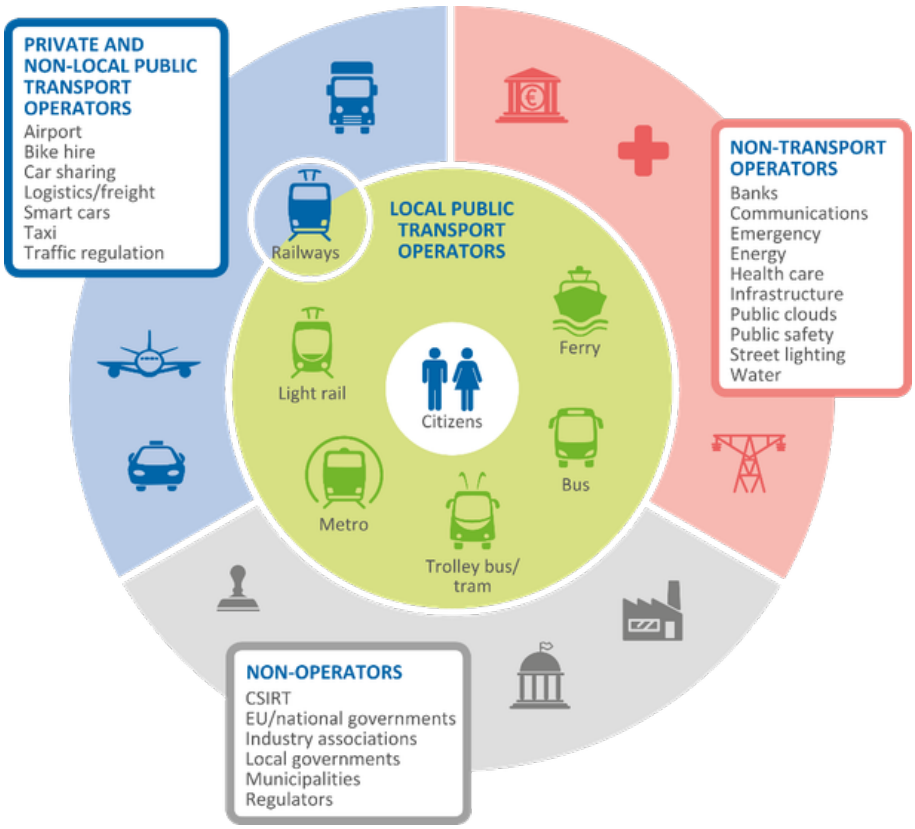



Figura 2.2: I domini applicativi chiave delle tecnologie smart. Fonte Enisa.

La democratizzazione di Internet, lo sviluppo delle reti di comunicazione ed i progressi nel campo dei meccanismi di computazione distribuita hanno stimolato anche una commercializzazione di dispositivi fatti "a misura d'uomo", il cui successo è un indicatore dell' impatto che queste innovazioni hanno avuto anche nella vita quotidiana delle persone.
Un esempio sono le Smart Home, ecosistemi che nascono per semplificare e migliorare lo stile di vita dell' individuo nell' ambito domestico. Queste, sfruttando una rete di sensori, di dispositivi ed elettrodomestici intelligenti distribuiti nei vari locali, danno vita a un' infrastruttura cosciente, in grado di reagire autonomamente a determinate situazioni, monitorare certe attività e rispondere in tempo reale attraverso app per smartphone sviluppate appositamente.
Anche in ambito automotive, i risultati degli ultimi anni nel campo della guida autonoma e semi autonoma hanno reso le Smart Cars oggetto di un crescente interesse da parte di aziende e persone. Grazie a un' infrastruttura costruita sull' intreccio di componenti veicolari, sensori e attuatori agli ordini di una centralina in grado di

eseguire compiti e prendere decisioni col supporto di algoritmi di intelligenza artificiale è stato possibile rinnovare servizi di diagnostica, connettività ed esperienza di guida [5].

Le opportunità offerte da queste infrastrutture tuttavia, vanno di pari passo con le insidie, soprattutto in materia di cybersecurity; a giustificare quest' affermazione, la storia recente riporta incidenti più o meno in ogni campo d'applicazione delle tecnologie smart e per le cause più svariate: difetti sconosciuti, inadeguatezza dell'architettura di sicurezza, complicità umana conscia o inconscia.
Spesso ad essere prese di mira sono infrastrutture critiche come quelle per l'assistenza sanitaria, le reti per la distribuzione di risorse fondamentali o quelle del trasporto pubblico. Ad esempio nel 2015, in Ucraina, un attacco hacker partito da una campagna di spear-phishing si è concluso con un' interruzione di diverse ore dell' afflusso di energia elettrica verso alcune zone della provincia di Kiev. Diversi anni prima, in Australia, un ex dipendente di un impianto per il trattamento delle acque si è vendicato dei torti subìti dal precedente datore di lavoro accedendo a distanza e senza autorizzazione ai sistemi SCADA di cui era stato responsabile, causando il versamento di tonnellate di liquame nelle aree circostanti l'impianto.

Sono oggetto di discussione anche le pratiche delle multinazionali produttrici di beni di consumo come automobili, elettrodomestici intelligenti e wearables, ree di mettere in commercio dispositivi che non soddisfano i requisiti minimi di sicurezza e anzi rappresentano un pericolo per la privacy o la salute degli utilizzatori.
In un articolo risalente al 2015, due ricercatori hanno dimostrato di poter compromettere da remoto un'automobile connessa ad Internet, assumerne il controllo e manipolare alcune funzionalità della vettura in corsa, tutto all' insaputa del guidatore [6]. In un altro approfondimento sono stati descritti i passi per compromettere un robot per le pulizie prima fisicamente e poi distanza e ottenere informazioni sensibili come la pianta dell' ambiente domestico o le credenziali d'accesso al Wi-Fi [7].
Questi episodi evidenziano, oltre che gli scarsi risultati nell' applicazione delle pratiche di sicurezza tradizionali, l'imprevedibilità e la pericolosità dei vari fattori che agiscono all' interno e all'esterno di un ecosistema intelligente e quanto sia necessario tenere in considerazione ognuno di essi durante le varie fasi del ciclo di vita di un prodotto.

Uno studio condotto da ENISA (European Union Agency for Cybersecurity) con l'obiettivo di analizzare il processo di adozione delle misure di sicurezza nell'Industry 4.0 e nell'applicazione dell' IoT per scopi industriali ha evidenziato diverse criticità [8]. I risultati hanno dimostrato come le problematiche legate alla cybersecurity siano spesso sottostimate, con le governance d' azienda riluttanti a finanziare il reclutamento di personale specializzato o a diffondere awareness sul tema tra i

livelli operativi. L' introduzione di tecnologie nuove e complesse sconvolge infatti le abitudini e i modi di lavorare consolidati negli anni, forzando un adattamento che, senza un adeguato supporto, non potrebbe essere sufficiente a rispondere a tutte le casistiche. Ad aggravare questo scenario concorrono le difficoltà legate al garantire l'interoperabilità non solo tra i nuovi dispositivi e le infrastrutture legacy ma anche tra un dispositivo smart e l'altro. Tra le cause principali vi è la mancanza di uno standard applicativo che regoli l'interazione sicura tra dispositivi a prescindere da produttore, luogo di fabbricazione o protocollo di comunicazione.
La consapevolezza della dimensione del problema viene raggiunta solo quando è ormai troppo tardi e una falla nell'infrastruttura di sicurezza ha determinato una perdita considerevole al portafogli dell' azienda.

L' eterogeneità e la vasta superficie d'attacco contribuiscono a rendere gli ecosistemi smart estremamente vulnerabili ad attacchi di cybersecurity.
Le cause di queste vulnerabilità vanno ricercate anche nelle diverse tecnologie che contribuiscono al raggiungimento degli obiettivi che uno specifico campo di applicazione intende perseguire.
Tra queste è possibile individuare:

- **Cyber-Physical Systems**, il frutto dell' integrazione di meccanismi di computazione e networking e dei processi industriali. L' espressione di maggiore rilievo di questa categoria sono gli SCADA (Supervisory Control And Data Acquisition), sistemi informatici adibiti al monitoraggio ed alla supervisione di sistemi fisici.
  La complessità raggiunta dall' insieme di queste tecnologie non è bilanciata da un altrettanto consolidato corredo di pratiche e standard in materia di cybersecurity ed interoperabilità tra componenti. Questa debolezza, unita alla sempre più sofisticata varietà di attacchi indirizzati a questi sistemi, aggiungono tra le cause di incidenti, oltre alle cause fisiche, anche quelle determinate dalle vulnerabilità cyber dell'infrastruttura [9];

- **IoT (Internet Of Things)**, un paradigma che consente l' interazione e lo scambio di dati tra oggetti "intelligenti", dotati di sensori, capacità computazionali e connessi ad Internet.
  Le criticità all' interno di questi ecosistemi dipendono da diversi fattori: capacità dell' hardware e robustezza del firmware dei dispositivi oppure varietà di mezzi trasmissivi e protocolli nello stack di comunicazione tra dispositivo e centri di raccolta dei dati. Queste limitazioni a volte non consentono la portabilità di algoritmi che implementino vincoli di sicurezza basilari, mentre troppe libertà nel design precludono dall' aderire a standard riconosciuti.
  Condurre attacchi a dispositivi smart diventa sempre più semplice e la lo-

ro diffusione determina una sempre maggiore probabilità che questi abbiano successo [10];

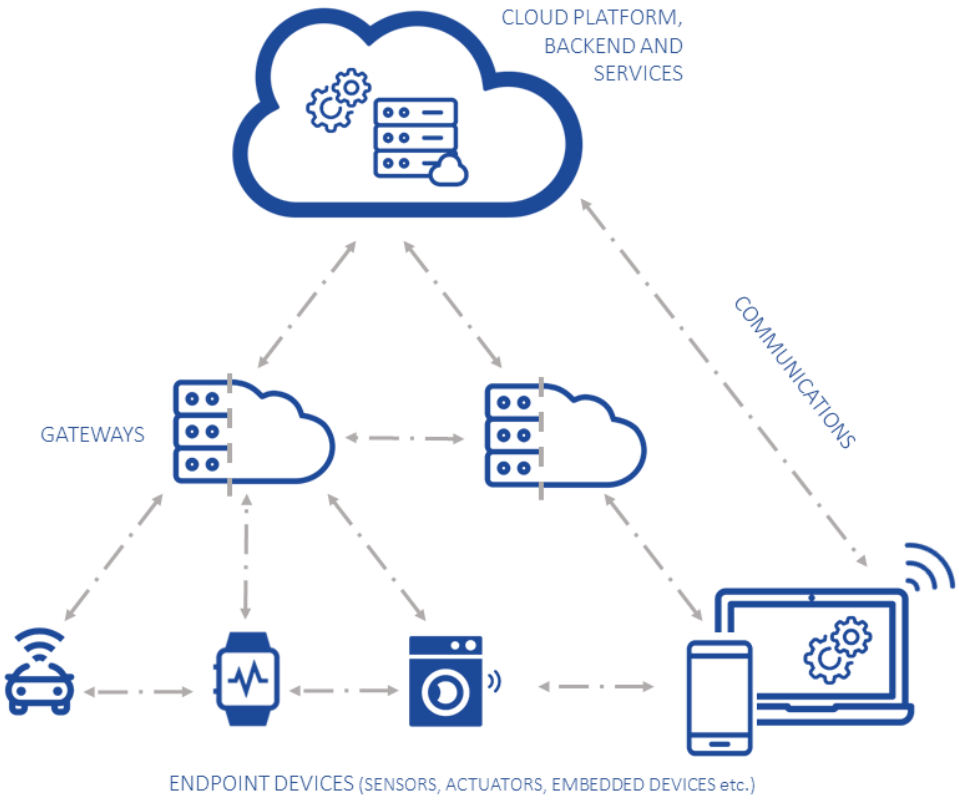



Figura 2.3: Architettura di un ecosistema basato su IoT e del relativo cloud [11].

- **Cloud, Fog and Edge Computing**: sono modelli di calcolo distribuito che si differenziano per la tipologia, le modalità e il luogo geografico in cui la computazione avviene rispetto ai punti in cui l'informazione viene raccolta. Nel primo caso, i dati collezionati (ad esempio tramite sensoristica) vengono trasmessi in rete ad un punto di calcolo centralizzato per il processamento. Nel secondo caso, uno strato dedito ad un parziale processamento o alla preparazione dei dati si frappone tra i livelli esistenti, così da diminuire il carico all' interno della rete. Nell' Edge Computing infine, questa parte dell' elaborazione avviene nelle vicinanze del luogo in cui i dati sono raccolti.
  Anche in questo caso le sfide di sicurezza sono diverse e riguardano indipendentemente le fasi del trasferimento dati, dell' analisi o del collegamento di

queste tecnologie con le infrastrutture esistenti. Le informazioni inviate dai punti di raccolta, verso i gateway e fino ai nodi in cui vengono aggregate, sono soggette a varie trasformazioni a causa dei diversi protocolli di comunicazione e metodi di connessione applicati durante il percorso. Gli stessi dati sono acceduti e ritrasmessi da applicazioni e servizi in esecuzione su dispositivi che devono garantirne integrità e confidenzialità e a loro volta devono dimostrare necessariamente la rispettiva autorità ed autenticità [11].

Un ecosistema intelligente comprendente entità umane, digitali e fisiche viene ribattezzato "ibrido" e rappresenta una categoria di sistemi che possiedono delle esigenze dal punto di vista della sicurezza differenti, più stringenti rispetto ai semplici sistemi ciberfisici, che non possono essere soddisfatte senza concepire degli approcci nuovi e che tengano conto delle peculiarità di ognuno degli agenti coinvolti.
Sia che si tratti di un macchinario responsabile del monitoraggio di complicati e rischiosi processi industriali oppure delle semplici procedure di configurazione dei dispositivi di cui si compone una casa intelligente, quando c'è di mezzo il fattore umano, nulla può essere lasciato al caso.
Come dimostrato anche dalla figura seguente, l'esito di un attacco è solo in parte determinato dalle capacità tecniche e dalla tenacia di chi intende perpetrarlo. La propensione all'errore, la predisposizione alla fiducia, la cura esclusiva degli interessi personali sono tutti aspetti che un attaccante può sfruttare per ottenere una serie di informazioni utili a compiere il resto del lavoro [12].

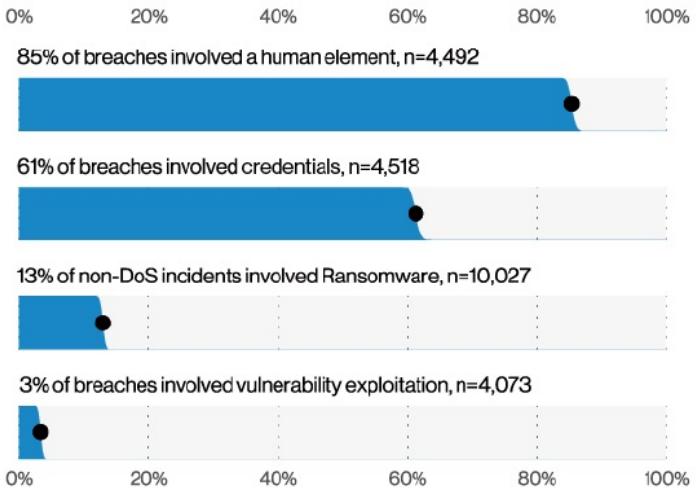



Figura 2.4: Influenza del fattore umano negli incidenti in riferimento al report [12].

Gli Smart Systems sono una risorsa fondamentale nel panorama dell' Industry 4.0 ma presentano una serie di problematiche dal punto di vista della sicurezza che vanno analizzate sotto ogni aspetto e affrontate in maniera sistematica.
Queste problematiche sono frutto delle debolezze intrinseche che caratterizzano le tecnologie alla base degli ecosistemi intelligenti e vanno approfondite non solo puntualmente ma anche e soprattutto valutando il contesto ed i possibili rischi che scaturiscono dall' interazione reciproca degli elementi costituenti l'infrastruttura cyberfisica.

# Capitolo 3

# Identificazione delle minacce

In questo capitolo verranno presentati i concetti fondamentali nel campo dell' identificazione delle minacce, del threat modeling e degli attack graph nonché lo stato dell' arte riguardo questi strumenti di supporto al processo di valutazione dei rischi di un sistema informatico.

## 3.1 Concetti chiave

Quella dell' analisi di sicurezza in ambito IT è una disciplina ricca, popolata da una moltitudine di strumenti e metodologie dalla caratterizzazione complessa ed in cui, a causa della vasta terminologia e dei vari domini di applicazione, risulta difficile orientarsi.
Essa può considerarsi come un processo, in atto continuamente, che ha l'obiettivo di valutare i rischi e stimare l'impatto che l'evenienza di determinate (o inattese) circostanze negative possono costituire per un sistema informatico e fornire allo stesso tempo sia i mezzi per proteggersi che i criteri per valutare la bontà di queste protezioni.

Si definisce "minaccia" una qualsiasi situazione potenzialmente in grado di danneggiare un sistema, provocando ad esempio la distruzione delle risorse da esso possedute, la divulgazione o l'accesso non autorizzato ad informazioni riservate o una discontinuità nella fornitura di uno o più servizi. Una minaccia può essere frutto della natura di una certa entità - e.g. un incendio rappresenta una minaccia per un edificio - ma può anche scaturire dall' abilità di un attaccante di sfruttare una o più debolezze, o "vulnerabilità", del sistema.

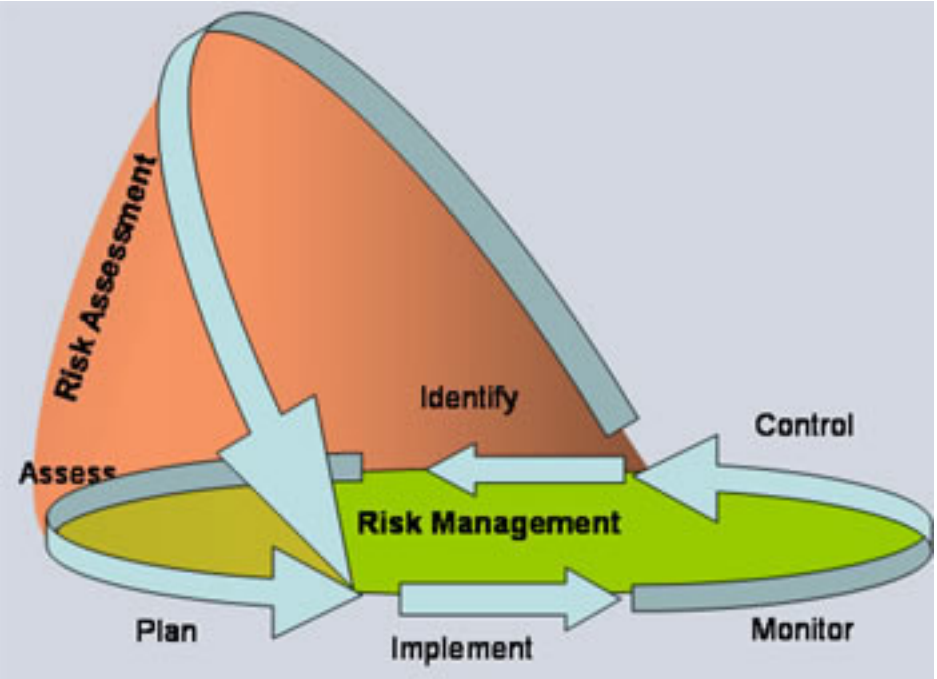



Figura 3.1: Il processo di risk assessment da un punto di vista vicino ad OCTAVE.

Una chiara comprensione della natura e delle interazioni tra vulnerabilità e minacce, interne ed esterne al sistema, è pertanto alla base di una corretta stima dei rischi e di un efficace dispiegamento delle contromisure volte ad eliminarli, un processo noto come "analisi dei rischi" o "risk assessment".

L'identificazione delle minacce è dunque uno degli step fondamentali dell' approccio strutturato alla valutazione di sicurezza, ma questo passaggio è tutt'altro che una mera enumerazione delle criticità di un sistema IT o dei vettori d'attacco più plausibili.
Spesso è necessario interrogarsi, oltre che sulla natura dei diversi fattori in gioco, anche sulle relazioni dirette o indirette che intercorrono tra gli stessi, al fine di individuare catene di azione o percorsi d'attacco che non emergerebbero altrimenti. Esistono inoltre situazioni estremamente domain-specific, come ad esempio l'analisi di una porzione di codice alla ricerca di possibili bug di sicurezza, nelle quali risulta difficile intervenire senza prima aver introdotto un livello di astrazione che consenta di reinterpretare attraverso un lessico familiare insiemi di informazioni eterogenee. Queste esigenze hanno fatto sì che, in alcuni casi, il processo di identificazione si elevasse a tal punto dalle restanti attività da assumere una valenza a sé stante, inglobando discorsi su altre parti dell' analisi di sicurezza.

A supportare tale processo esistono dei tool, come la knowledge base ATT&CK di MITRE[1] in grado di offrire una efficace classificazione di attori e avversità.
L'accezione più ampia di minaccia infatti include una possibile congiunzione di avvenimenti negativi capaci di mettere in discussione la sicurezza di un certo sistema. Spesso, ciò che determina una tale coincidenza di eventi, al di là della sfortuna, è uno scopo, condiviso da uno o più agenti esterni intenzionati a perpetrare un danno di cybersecurity. Tuttavia, per quanto i progressi fatti nel campo IT possano motivare e stimolare l'ingegno di questi gruppi è possibile riconoscere una certa ciclicità negli attacchi, negli obiettivi e nelle tecniche applicate per perseguirli.
Le possibilità offerte da questi database di conoscenza in materia di comportamenti, metodi e tecnologie d'attacco sono ben sfruttate in una disciplina, quella del threat modeling, il cui concetto di fondo, secondo il glossario NIST, è strettamente legato a quello di "risk assessment" [2].

## 3.2 Threat Modeling

Dare una definizione univoca del termine "threat modeling" è difficile per diversi motivi. Tra questi c'è il fatto che in una disciplina così vasta ed in continua evoluzione come l'analisi di sicurezza, spesso il confine tra metodologie diverse risulta molto labile. Un' altra motivazione, forse più importante, riguarda la semantica delle parole che danno il nome a questa disciplina, ovvero "threat" e "model", di cui bisogna chiarire significato e destinazioni per comprenderne al meglio il ruolo all' interno del processo dell' assessment di sicurezza.

Precedentemente si è affermato che può considerarsi una minaccia una qualsiasi eventualità capace di arrecare danno a una o più risorse d'interesse.
Nel caso più semplice tale evenienza può scaturire dalla natura della risorsa stessa mentre nel caso più complesso può risultare dall'azione di fattori molto diversi e distanti tra loro. Una chiara visione delle potenziali fonti di pericolo prescinde da una corretta valutazione di questi elementi, delle relazioni e delle dinamiche che li governano, da una caratterizzazione degli avversari e dei punti di contatto tra loro e il sistema in esame, siano questi diretti, e.g. una vulnerabilità immediatamente sfruttabile, o indiretti, e.g. la compromissione di un nodo a successiva ad azioni di movimento laterale.
Lo scopo di un modello è invece quello di dare un senso alle informazioni disseminate nel contesto osservato, categorizzandole attraverso una sintassi predefinita. Esso è caratterizzato da un linguaggio in grado di unire, grazie anche all' espressività dell'

[1] `https://attack.mitre.org/`
[2] `https://csrc.nist.gov/glossary/term/threat_modeling`

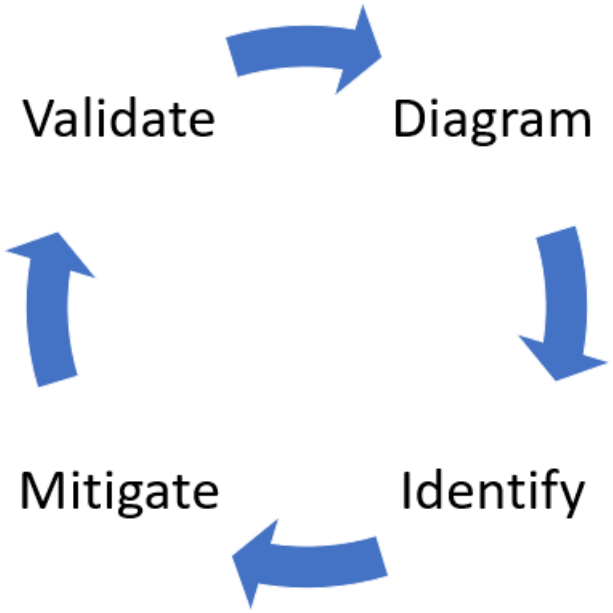



Figura 3.2: Passi di alto livello del processo di threat modeling.

insieme di regole su cui si basa, al livello di dettaglio e di profondità richiesta da un' analisi di questo tipo l'immediatezza e le capacità di sintesi dei formalismi che introduce.

Un threat model è dunque un' astrazione, frutto di un processo detto di "threat modeling", che studia i possibili modi con cui un attaccante può danneggiare gli asset di un sistema, evidenziando le potenziali fonti di minaccia, le cause e le ripercussioni che avrebbero sullo scenario, valutandone i rischi e soppesandone le contromisure, in una sequenza di operazioni solitamente riconducibili a quattro fasi ben distinte, come elencate di seguito:

1. **Descrizione** Il primo passo consiste nell' individuare le componenti più rilevanti dello scenario analizzato e rappresentarle utilizzando una sintassi o dei diagrammi che ne riassumano le proprietà principali.
   Queste componenti possono includere nodi del sistema, connessioni, persone e in generale qualunque entità interna o esterna in grado di relazionarsi e di influenzare lo scenario stesso.
   Ovviamente maggiore è l'attenzione al dettaglio durante questa fase più numerosi saranno gli spunti per quelle successive.

2. **Identificazione** A partire dal resoconto ottenuto dal punto precedente si inizia a riflettere, nodo per nodo e collegamento per collegamento, su quelle situazioni

che potrebbero rappresentare condizioni di pericolo nei confronti degli asset del sistema, eventualmente etichettando ciascuna minaccia secondo le proprietà di sicurezza che violerebbe oppure profilando l'avversario sulla base delle relative caratteristiche.

3. **Contromisure** Per ogni possibile minaccia individuata vengono proposte delle contromisure in grado di contrastare le criticità che le alimentano.
   In questo caso, avere una visione globale dell' ossatura del sistema e una comprensione profonda delle dinamiche che caratterizzano l' infrastruttura è essenziale per la buona riuscita del processo di valutazione delle soluzioni a disposizione.
   Tali soluzioni possono a loro volta essere mirate a risolvere puntualmente un problema o a descrivere un insieme di azioni volte a mitigare più problematiche interdipendenti.

4. **Validazione** Lo step di validazione infine è volto ad analizzare la ricchezza e la precisione nella rappresentazione generata, la coerenza nell' enumerazione delle minacce e l'efficacia nella somministrazione delle contromisure applicando il threat model ottenuto ad uno o più scenari noti o predeterminati.
   Le mancanze individuate durante l'esecuzione di questo passaggio offrono gli spunti per eventuali modifiche o raffinamenti di uno qualsiasi dei task precedenti.

L'applicazione di questo strumento offre al responsabile di sicurezza diverse possibilità: durante le fasi iniziali dello sviluppo di un artefatto hardware o software consente una maggiore comprensione delle criticità nella sicurezza della relativa infrastruttura, permettendo di agire di conseguenza sul processo di design (comportamento proattivo); quando il sistema è a regime, in caso di attacco, offre punti di vista differenti in merito alle soluzioni destinate a contrastarlo (comportamento reattivo).
I risultati ottenuti dipendono da diversi fattori. La dimensione e la complessità del sistema da descrivere, il dominio applicativo e l'esperienza posseduta dalla figura (o dalle figure) responsabili di eseguire l'analisi di sicurezza sono aspetti determinanti per una buona riuscita di questo processo.

### 3.2.1 Related Work

Un ottima introduzione allo stato dell' arte nell' ambito del threat modeling è il resoconto in [13], un lavoro che rappresenta il primo approccio sistematico alla letteratura riguardante questo argomento. Gli autori hanno analizzato un' ampia varietà

**Table 6 – System types addressed by articles in C1.**

| Category | | References |
|---|---|---|
| General | Software application | Xu and Nygard (2006); Xu et al. (2012); Chen et al. (2012); Al-Fedaghi and Alkandari (2011); Bedi et al. (2013) |
| | ICT system | Lavrova and Pechenkin (2015); Musman and Turner (2018); Dahbul et al. (2017); Meszaros and Buchalcevova (2017); Pei et al. (2004) |
| | Smart grid system | Kalinin and Konoplev (2014) |
| | Cloud computing | Idziorek and Tannian (2012); Paladi et al. (2016) |
| | Security protocol | Arsac et al. (2011); Martina et al. (2015) |
| Specific | AMI | Jiang et al. (2014); Liu et al. (2015) |
| | Vehicular ad hoc network (VANET) | Yan et al. (2014) |
| | SCADA network | Cardenas et al. (2009); James and Prabakaran (2015) |
| | CPS architecture for healthcare | Seifert and Reza (2016) |
| | Software-defined network (SDN) | Wu and Wei (2017) |
| | Optical cross-connect systems | Hofmann and Kasseckert (2011) |
| | Network mobility protocol | Bauer (2013) |
| | Unmanned aerial systems | Baquero et al. (2015) |
| | Telemedicine application | Pendergrass et al. (2014) |
| | Smart home environments | Olawumi et al. (2017) |
| | Electronic Health Record Systems | Almulhem (2012) |
| | Mobile telecommunication network | Dimitriadis (2013) |

**Table 7 – System types addressed by articles in C2.**

| Category | | References |
|---|---|---|
| General | Software application | Deng et al. (2011); Li et al. (2009); Li et al. (2014); Sabbagh and Kowalski (2015); Bryant and Saiedian (2017) |
| | ICT system | Frydman et al. (2014); Al-Fedaghi and Moein (2014); Magklaras et al. (2006) |
| | Cyber human systems | Kammüller and Probst (2015) |
| | Cyberspace | Sharma et al. (2013); Marback et al. (2013) |
| | Distributed systems | Uzunov and Fernandez (2014) |
| | CPS | Burmester et al. (2012) |
| | Smart grid system | Suleiman et al. (2015) |
| | Control system | Huang et al. (2009) |
| Specific | Terrorist network | Singh et al. (2004) |
| | Mobile device management system | Rhee et al. (2013) |
| | Power supply system | Brændeland et al. (2010) |
| | Socio-technical physical system | Lenzini et al. (2015) |
| | Windows operating system | Pan and Zhuang (2017) |

Figura 3.3: Metodologie nuove nel campo del threat modeling, suddivise in base al target.

di documenti sul tema, predisponendo le metodologie ivi descritte ad una classificazione basata su similitudini e distanze tra ognuna.
Ciascun articolo viene in prima istanza categorizzato sulla base del contributo apportato alla disciplina distinguendo tre cluster: un primo contenente introduzioni ad approcci nuovi, un altro rapporti su tecniche esistenti mentre l'ultimo discorsi generali sul processo di threat modeling. Ciascun cluster viene poi ulteriormente raffinato analizzando gli aspetti caratterizzanti le varie discussioni e.g. se lo studio tratti di tecniche "formali", ovvero basate su modelli di tipo matematico o "grafiche" oppure di metodi "manuali" o "automatici" ecc.
Un' altra distinzione riguarda la tipologia di sistema cui è destinata una particolare tecnica.

Dalla tabella si può constatare come parte delle ricerche abbia come target l'ambito degli smart systems e dei sistemi ciberfisici.
Tra queste, ad esempio, [14] analizza i potenziali rischi di sicurezza nel contesto delle Smart Home, in particolar modo quelli legati alla privacy dei dati raccolti dai dispositivi che le compongono. Il threat model discusso non viene introdotto formalmente e si appoggia alla classificazione STRIDE, utilizza DFD in fase di descrizione e concentra il focus maggiormente su rischi derivanti da fattori di tipo cyber come protocolli e tecnologie wireless caratterizzanti l'infrastruttura di rete. In [15], il processo di threat modeling supporta il confronto e la valutazione di diverse architetture di sistemi ciberfisici destinati al settore sanitario. In questo caso, vista la criticità dell' ambito, vengono presi in considerazione sia gli aspetti cyber legati alla trasmissione delle informazioni sia ovviamente quelli fisici relativi all'utilizzo dei dispositivi destinati alla somministrazione delle cure ai pazienti. Anche [16] propone una un nuovo approccio all' identificazione delle minacce nelle reti di sensori e nei sistemi SCADA basato su una valutazione delle proprietà di sicurezza di maggiore rilevanza per questo ambito e delle tipologie di attacco che mirano a impedirne l'attuazione. Il threat model studia le possibili interazioni interne ed esterne alle infrastrutture oggetto di analisi ed introduce una propria tassonomia degli attacchi. [17] approfondisce le relazioni tra entità di tipo fisico e cyber in un threat model destinato a sistemi ciberfisici ed introducendone in maniera dettagliata le componenti. Le minacce ed i rischi sono analizzati dopo aver reinterpretato lo scenario osservato attraverso un linguaggio formale, una particolare categoria di automi a stati finiti e delle regole che giustifichino le transizioni da uno stato all' altro.
Tutti i casi precedentemenenti enunciati trascurano il fattore umano.

Dietro il concepimento di un threat model destinato all' insider threat analysis in [18] c'è invece un approccio sistematico allo studio dell' accoppiata uomo-macchina. In questo caso viene proposto un framework che ha lo scopo di individuare minacce di tipo cyber e umano, basato su ricerche in ambito sociologico e sui metodi formali dell' IT. [19] infine, discute di un framework per l'analisi di sicurezza di sistemi socio-tecnici, ovvero composti da persone ed oggetti fisici, mettendo in relazione infrastrutture ed agenti che le manipolano e si muovono attraverso di esse Il risultato è uno strumento che valuta e quantifica i rischi associati ai vari contesti che possono venire a crearsi e le relative possibilità di attacco e che nonostante non tratti oggetti digitali si propone di farlo nell' immediato futuro.

Altri esempi di threat models destinati all' ambito smart e non, compaiono in [20] e [21]. Nel primo, viene presentato un linguaggio di modellazione, enterpriseLang, (nato da un metalinguaggio denominato Mal) che sfrutta la Mitre ATT&CK Enterprise Matrix come knowledge base per la descrizione di sistemi IT d'impresa. Nel secondo, viene descritto il framework MulVal, anch'esso dotato di un linguaggio

di modellazione ed uno di query verso una base di conoscenza per l'aggiornamento delle configurazioni di rete sui nodi in esame.
Le metodologie presentate fin qui sono domain-specific oppure si limitano all' utilizzo di pattern pre-esistenti e non analizzano contemporaneamente gli aspetti cyber, fisici e umani che caratterizzano i sistemi ibridi.
Sarebbe azzardato affermare che in quest'ambito non sia stata mai discussa un'applicazione dei processi di threat modeling nelle condizioni suddette. Non lo è tuttavia evidenziare come la varietà, l'estensione e la complessità di questi sistemi giochino a sfavore della generalizzazione delle tecniche di analisi e modellizzazione.

Anche volgendo lo sguardo a modelli più longevi, come fa il sommario in [22] nessuno di questi possiede i requisiti adatti a rispondere all'esigenze di sicurezza dettate dalla complessità delle infrastrutture che dominano l' ambito smart.
Uno degli approcci discussi qui, focalizzato sull' ambito IT, è ad esempio STRIDE, di casa Microsoft. Esso prevede una fase di modellazione basata sull'utilizzo di data flow diagram (DFD), un tipo di diagramma che consente di rappresentare flussi di informazioni all' interno e all'esterno del sistema, verso agenti al di fuori del contesto analizzato. La fase di identificazione ha invece l'obiettivo di ricondurre le potenziali minacce individuate ad un set predefinito, da cui deriva il nome del metodo stesso: **S**poofing identity, **T**ampering with data, **R**epudiation, **I**nformation disclosure, **D**enial of service, **E**levation of privilege. OCTAVE è un altro esempio, sviluppato successivamente e con un focus diverso. Il settore è quello aziendale e lo scopo è quello di fornire, introducendo una serie di attività da eseguire autonomamente, pratiche e strumenti per allenare il comparto operativo ad una corretta valutazione dei rischi legati all' information security. Tra le tecniche "grafiche" viene anche menzionata quella degli alberi d'attacco, diagrammi che consentono di visualizzare i diversi modi con cui un avversario può danneggiare un certo asset.
Un altro artificio, parente del precedente e dallo scopo simile, è il grafo d'attacco, un tipo di diagramma che si presta molto bene alla descrizione di sequenze o combinazioni di eventi che rappresentano una minaccia per la sicurezza di un sistema.

### 3.2.2 Attack Graph

Un grafo d'attacco offre una visualizzazione compatta dei percorsi che un attaccante può sfruttare per compromettere una o più risorse d'interesse.
L' impatto visivo risultante dall' applicazione di questa tecnica è tale da consentire di individuare immediatamente le problematiche relative allo scenario analizzato e avere una visione d'insieme dei punti di contatto e di prosecuzione dell' avversario all' interno del sistema.

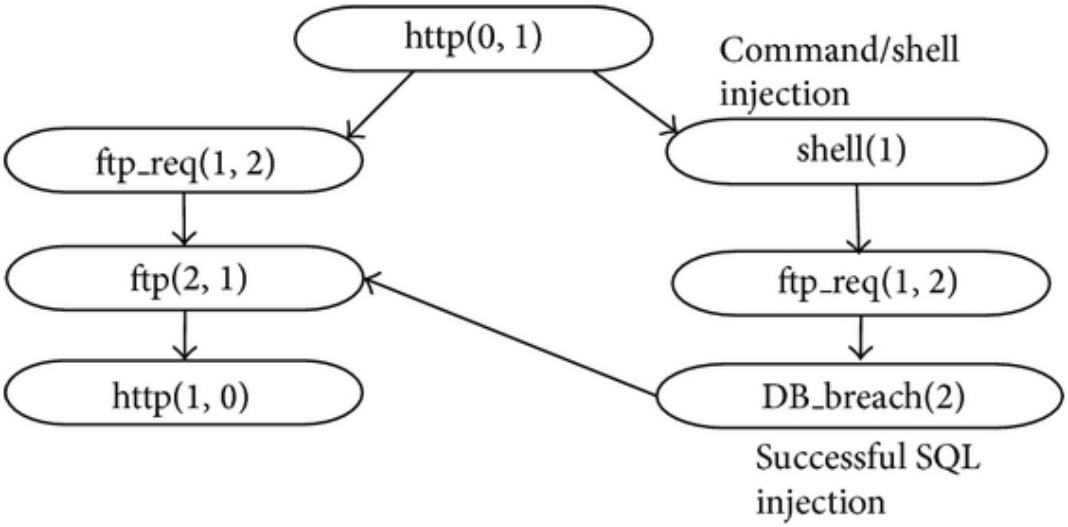



Figura 3.4: Esempio di grafo d'attacco tratto da [24].

Malgrado la confusione generale sulla semantica e sulla terminologia utilizzata in questo ambito, sostanzialmente dovute alla mancanza di uno standard di riferimento per questo tipo di rappresentazione, i benefici di un approccio di questo tipo superano gli svantaggi [23].

In linea generale un nodo del grafo identifica una condizione o la capacità di un avversario di prendere vantaggio dell'entità cui esso corrisponde. Un arco invece coincide con una transizione, un evento o un percorso che conduce da un nodo all' altro (o da più nodi verso altri nodi).
Un "attack path" consiste dunque di una sequenza di nodi, all' interno del sistema target di cui l'attaccante, a causa di vulnerabilità o debolezze di altro tipo, si è impossessato o è in grado di sfruttare a proprio favore.

Il grafo d'attacco è uno strumento che consente due tipologie di analisi:

- **analisi statica** eseguita quando il sistema è "a regime" per conoscere a quali tipologie di rischio esso è esposto;
- **analisi dinamica** eseguita quando il sistema è "a regime" ma successivamente ad un mutamento dello stato di sicurezza di una o più componenti è necessario aggiornare il resoconto globale.

La validità dell' attack graph come elemento di supporto all' analisi di sicurezza è discussa ad esempio in [25] o [26]: nel primo caso per diminuire l'overhead dovuto a tecniche di vulnerability scanning nei sistemi industriali di controllo delle smart grid. La rappresentazione in forma di grafo consente di classificare i nodi del sistema sulla base dei livelli di rischio cui sono esposti ed ottimizzare così lo scheduling di determinate procedure; nel secondo caso il grafo viene generato automaticamente insieme al threat model del sistema, grazie all' utilizzo di un linguaggio di descrizione derivato da MAL, nell' ambito dell'analisi di sicurezza delle centrali elettriche.

# Capitolo 4

# Problem statement

I capitoli precedenti hanno evidenziato le fragilità che caratterizzano i sistemi intelligenti dal punto di vista della sicurezza e l'attenzione necessaria a modellare quelle infrastrutture in cui l' uomo, in relazione con le entità cyberfisiche esistenti, può giocare un ruolo determinante in un potenziale scenario d'attacco.
Si è inoltre osservato che, a scapito dei recenti trend osservati in materia di sistemi ibridi, in letteratura sono poco discusse metodologie e pratiche di modellazione in questa direzione mentre lo sono ampiamente, in via teorica e formale, sia tecniche domain-specific che generali, ma che concentrano i loro sforzi in contemporanea solo due dei tre fattori citati fin qui. "Tameless", il threat model utilizzato come base per gli sviluppi che verranno discussi nei capitoli successivi, definisce un linguaggio generale per la descrizione di questi sistemi ed è agnostico rispetto alla natura della particolare componente nell' infrastruttura sotto esame.
Il funzionamento di questo strumento è vincolato alle convenzioni del "Prolog" - un linguaggio per la programmazione logica molto diffuso - con l' obiettivo di sfruttarne i meccanismi inferenziali e ottenere, in base alle assunzioni teoriche su cui poggia il modello e ai "fatti" caratterizzanti il sistema modellato (le risorse, le connessioni e i rischi cui è esposto), una nuova serie di informazioni in grado di riassumere lo stato di sicurezza globale e componente per componente del sistema sotto esame.

Il lavoro di tesi è consistito dapprima nel comprendere le basi teoriche dietro il modello e il contesto in cui si inserisce, nell' approfondire le fondamenta del linguaggio che lo implementa e nello sviluppare degli use cases che dimostrassero la duttilità del threat model e ne esplorassero le potenzialità.
Il passo successivo è stato quello di estendere il comportamento di base del threat model, integrando la possibilità di visualizzare i risultati della sua applicazione in formato grafico, attraverso l'impatto di un grafo d'attacco e di ottenere gli insiemi

delle possibili contromisure da attuare per contrastare le minacce evidenziate.
La generazione dell' attack graph e la valutazione delle remediations rappresentano l'argomento centrale delle pagine seguenti.

L'estensione su cui si è lavorato prende la forma di un programma Java che, attraverso una libreria di supporto denominata "JPL" si interfaccia con il nucleo scritto in Prolog e affianca alla modalità usuale di interazione tramite query una modalità più articolata, in linea con la precedente, ma che in più consente di:

1. costruire iterativamente un grafo d'attacco ed esportarlo in formato dot;
2. eseguire interrogazioni puntuali sulle varie componenti del sistema ed ottenere in risposta le possibili mitigazioni in formato JSON.

# Capitolo 5

# Design della soluzione

Per contestualizzare al meglio il lavoro fatto, in questo capitolo viene introdotta la teoria che sta dietro *Tamless*, il tool che definisce un threat model e una threat analysis destinata a sistemi ibridi.
Verranno anche presentati gli approcci che hanno guidato l'aggiunta delle funzionalità di generazione dell' attack graph e di individuazione delle contromisure, insieme ad alcuni esempi di utilizzo.

## 5.1 Tameless

*Tameless*, nella sua forma originale, è uno strumento di supporto per l'analisi di sicurezza che si ispira alla logica del primo ordine[1].
Il tool si basa su un nuovo threat model, definito "ibrido", in grado di rappresentare le relazioni tra le varie componenti di un sistema e le proprietà di sicurezza di ciascuna, tenendo in considerazione fattori che sono indipendentemente di tipo cyber, umano o fisico. Esso introduce inoltre un set di regole che consente di eseguire una threat analysis su un qualunque sistema descritto secondo i formalismi del modello suddetto e, sfruttando i meccanismi della logica del primo ordine, di individuare, quasi automaticamente, percorsi e pattern d'attacco di un possibile scenario critico. Nei paragrafi successivi verranno presentate, nel dettaglio e con alcuni esempi, queste due parti del tool.

[1]Una logica del primo ordine è "[...] una teoria formale in cui è possibile esprimere enunciati e dedurre le loro conseguenze logiche in modo del tutto formale e meccanico." *Wikipedia*

### 5.1.1 Hybrid threat model

Il threat model dietro *Tameless* fornisce una terminologia che ha l'obiettivo da un lato di astrarre le componenti del sistema analizzato rispetto alla loro varia natura e dall' altro di reinterpretare le possibili connessioni formate da insiemi di componenti secondo delle relazioni prestabilite.
Tra i concetti introdotti, quello di *entità*, ad esempio, è utilizzato per modellare le risorse, gli elementi e gli agenti che ruotano attorno il sistema oppure quello di *threat* per indicare quelle azioni che, direttamente o indirettamente, possono far mutare lo stato di sicurezza di una o più *entità*. I legami tra le entità, o tra threat ed entità vengono espresse in termini di *relazioni* e *proprietà di sicurezza* predefinite.

Esempi di relazioni tra entità sono:

- **Contain** Contain(A,B): per indicare che l' entità A contiene l'entità B. Un possibile esempio è quello in cui A corrisponde a un "server" e B corrisponde a dei generici dati;
- **Depend** Depend(A,B): per indicare che la funzionalità dell' entità A dipende dalla funzionalità dell' entità B, come nel caso di un antivirus e dell' attività del relativo database delle minacce;
- **Connect** Connect(A,B,C): per indicare che l'entità A connette B a C, come ad esempio una rete collega due server remoti.

Esempi di relazioni tra entità e threat sono:

- **Protect** Protect(B,A,T): per indicare che l'entità B protegge A dal threat T, come una serratura che protegge la porta da accessi non autorizzati;
- **Spread** Spread(A,T): per indicare che l'entità A è in grado di propagare il threat T, ad esempio un impiegato ingannato tramite ingegneria sociale può installare sul proprio pc del software malevolo (in questo caso T è il software malevolo che ha avuto modo di infettare il pc con la complicità dell' impiegato, A);
- **Monitor** Monitor(A,B,T): per indicare che l'entità A monitora B rispetto al threat T. Un possibile esempio è un sistema di videosorveglianza che sorveglia un locale rispetto alle intrusioni.

L'insieme delle *proprietà di sicurezza* è un altro elemento importante nella caratterizzazione del sistema.
Tra queste si distinguono le *proprietà di base*:

- **Compromised** Comp(A,T)[2]: in relazione a un'entità A che è stata compromessa da un threat T;
- **Malfunctioned** Malfun(A): in relazione a un'entità A che non funzioni correttamente;
- **Vulnerable** Vul(A,T): in relazione a un'entità A vulnerabile rispetto a un threat T.

L' insieme delle *proprietà ausiliarie*, che hanno l'obiettivo di descrivere lo stato di sicurezza o di funzionamento dell' entità cui si riferiscono, ad esempio:

- **Detected** Det(A,T): per indicare che è possibile rilevare che l'entità A sia stata compromessa dal threat T;
- **Restored** Rest(A): per indicare che è possibile ripristinare l'entità A ad uno stato sicuro quando questa sia stata compromessa.

Infine, l' insieme delle *proprietà di alto livello* che, semplicemente, offrono delle scritture equivalenti per proprietà che risultano dalla combinazione delle precedenti, permettendo di semplificare la definizione delle regole di derivazione su cui si basa la threat analysis.
Alcuni esempi sono:

- **Valid**[3] Val(A): in relazione a un'entità che non è né compromessa rispetto ad alcun threat T né non funzionante;
- **Defended**[4] Def(A,T): in relazione a un'entità che è protetta da almeno un entità B che risulti valida;
- **Safe**: Safe(A,T): in relazione a un'entità che è difesa o non è affatto vulnerabile rispetto a un threat T.

[2]presente anche nella forma Comp(A) senza specificare il threat T.
[3]presente anche con lo mnemonico equivalente **Usable**.
[4]presente anche con lo mnemonico equivalente **Protected**.

$$\text{Val}(A) := \neg\text{Comp}(A) \wedge \neg\text{Malfun}(A)$$
$$\text{Def}(A,T) := \exists B.\ \text{Protect}(B,A,T) \wedge \text{Val}(B)$$
$$\text{Safe}(A,T) := \neg\text{Vul}(A,T) \vee \text{Def}(A,T)$$
$$\text{Mon}(A,T) := \exists B.\ \text{Monitor}(B,A,T) \wedge \text{Val}(B)$$
$$\text{Check}(A) := \exists B.\ \text{Check}(B,A) \wedge \text{Val}(B)$$
$$\text{Rep}(A) := \exists B.\ \text{Replicate}(B,A) \wedge \text{Val}(B)$$

$$\neg\text{Val}(A) := \text{Comp}(A) \vee \text{Malfun}(A)$$
$$\neg\text{Def}(A,T) := \nexists B.\ \text{Protect}(B,A,T) \vee (\forall B.\text{Protect}(B,A,T) \wedge \neg\text{Val}(B))$$
$$\neg\text{Safe}(A,T) := \text{Vul}(A,T) \wedge \neg\text{Def}(A,T)$$
$$\neg\text{Mon}(A,T) := \nexists B.\ \text{Monitor}(B,A,T) \vee (\forall B\ \text{Monitor}(B,A,T) \wedge \neg\text{Val}(B))$$
$$\neg\text{Check}(A) := \nexists B.\ \text{Check}(B,A) \vee (\forall B\ \text{Check}(B,A) \wedge \neg\text{Val}(B))$$
$$\neg\text{Rep}(A) := \nexists B.\ \text{Replicate}(B,A) \vee (\forall B.\ \text{Replicate}(B,A) \wedge \neg\text{Val}(B))$$

Figura 5.1: Una panoramica delle *proprietà di alto livello*. Nella tabella superiore in forma naturale, in quella inferiore nelle relative forme negate.

È importante sottolineare che le categorie delle *proprietà di base* e delle *proprietà ausiliarie* possiedono la facoltà di essere interpretate in maniera differente.
Quando sono utilizzate per effettuare delle ipotesi riguardanti lo stato di sicurezza delle varie entità del sistema, si definiscono proprietà "assunte" e si usa indicarle con la sintassi $\alpha$Comp, $\alpha$Malfun ecc. Quando sono frutto delle procedure di derivazione conseguenti l'applicazione delle regole della threat analysis, si definiscono proprietà "derivate" e si indicano con la sintassi $\kappa$Comp, $\kappa$Malfun.

Ricondurre il sistema ed il contesto osservati ad una rappresentazione che segua i formalismi di tale modello e ne rispetti i vincoli è il compito più interessante e sfidante che derivi dall' utilizzo del tool.
La terminologia introdotta consente di astrarre efficientemente la natura complessa e diversificata delle infrastrutture ibride e fornisce una panoramica essenziale delle componenti e degli agenti che governano il sistema analizzato. Tuttavia, il fatto di essere questo un processo da eseguire manualmente, lo rende prono all' errore o continuamente oggetto di aggiustamenti che possono involontariamente influenzare la successiva threat analysis.

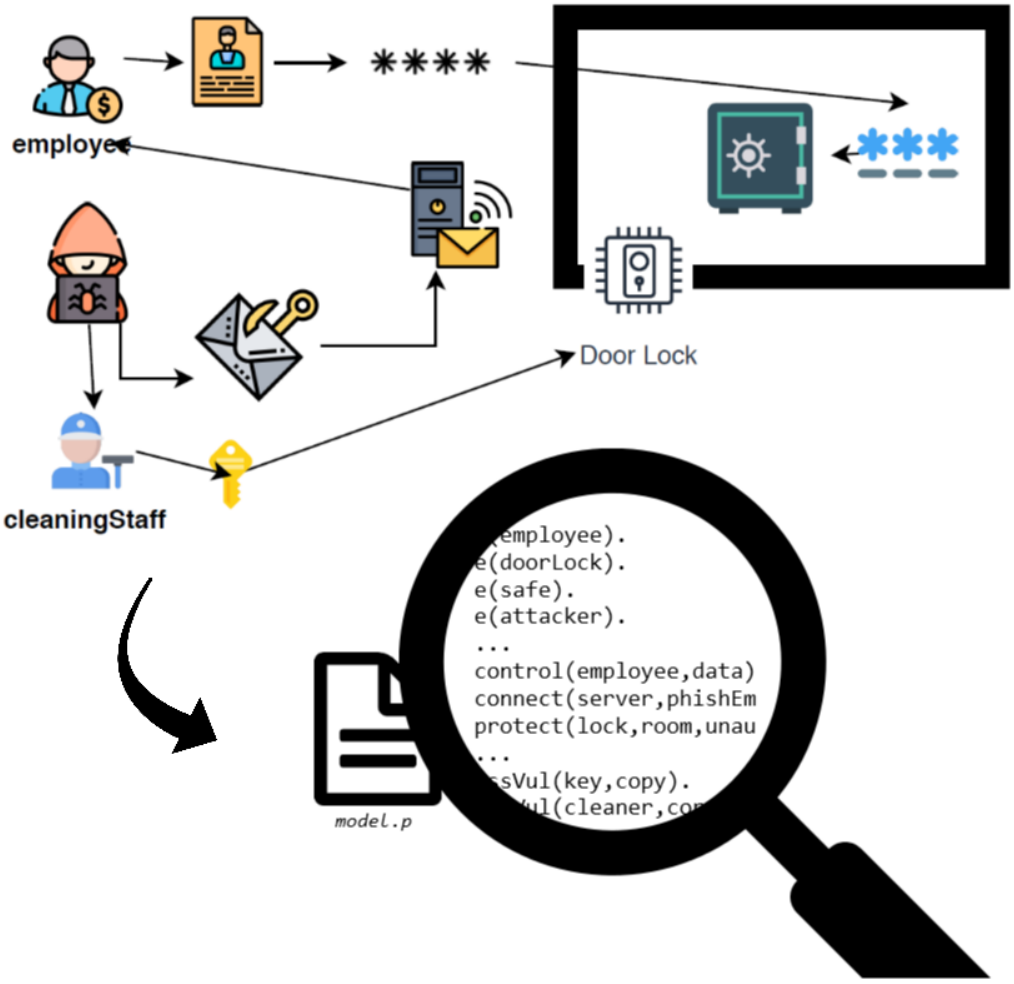



Figura 5.2: Dal sistema al threat model implementato tramite sintassi *Prolog*.

### 5.1.2 Threat analysis

In questa parte del processo entrano in gioco i meccanismi di inferenza, nella forma di regole di derivazione, che seguono i dettami teorici della logica del primo ordine. In pratica, avendo fornito una descrizione del sistema sotto esame secondo la sintassi e la semantica del modello, applicando queste regole è ora possibile derivare meccanicamente nuove conoscenze circa lo stato di sicurezza del sistema e delle sue entità.

Le regole introdotte per prime coinvolgono le proprietà di base.
L'insieme è quello delle *regole di derivazione base*, attraverso le quali si può concludere che se un proprietà si assume vera allora può essere naturalmente derivata

vera:

$$\alpha\text{Comp(A,T)} \rightarrow \kappa\text{Comp(A,T)}$$
$$\alpha\text{Vul(A,T)} \rightarrow \kappa\text{Vul(A,T)}$$
$$\alpha\text{Malfun(A)} \rightarrow \kappa\text{Malfun(A)}$$

Altre regole mettono in relazione entità e relative proprietà di sicurezza con il contesto in cui vivono, con l'obiettivo di derivare nuove conoscenze.
Dalle seguenti risulta che, con le determinate condizioni, ogni entità corra il rischio di vedere mutare il proprio stato verso uno stato non sicuro, descritto dalla rispettiva proprietà.
Ad esempio nel caso di un'entità A sotto il controllo di un'entità B compromessa:

$$\text{Control(B,A)} \land \kappa\text{Comp(B)} \land \text{Spread(B,T)} \land \neg\text{Safe(A,T)} \rightarrow \kappa\text{Comp(A,T)}$$

Oppure nel caso in cui la medesima entità compromessa sia legata da una relazione di connessione con l'entità vittima:

$$\text{Connect(C,B,A)} \land \kappa\text{Comp(B)} \land \text{Spread(B,T)} \land \neg\text{Safe(A,T)} \land (\kappa\text{Comp(C)} \lor \neg\text{Def(C,T)}) \rightarrow \kappa\text{Comp(A,T)}$$

O ancora quando questa sia contenuta (o contenga) un'entità compromessa, come segue:

$$(\text{Contain(B,A)} \lor \text{Contain(A,B)}) \land \kappa\text{Comp(B)} \land \text{Spread(B,T)} \land \neg\text{Safe(A,T)} \rightarrow \kappa\text{Comp(A,T)}$$

Questi insiemi di regole rappresentano l'innesco dei meccanismi della logica del primo ordine nonché il collante tra le ipotesi sugli elementi del sistema, e.g. "un' entità può essere vulnerabile a…", "un' entità può essere compromessa da…" e le relazioni che coinvolgono gli stessi, e.g. "l'entita è connessa con…", "l'entità protegge dal threat…".

Tutte le *proprietà di base* e tutte le *proprietà ausiliarie* sono coinvolte in regole che hanno l'obiettivo di individuare i potenziali rischi legati alla struttura del sistema, come nel caso delle prime, e valutare l'efficacia delle contromisure esistenti, come nel caso delle seconde.

## 5.2 Attack Graph

Nel processo di costruzione e di design del grafo d'attacco è stato centrale il problema di dare un significato ai concetti di nodo e arco.
L' obiettivo del diagramma è visualizzare i potenziali punti di contatto e i percorsi di cui può avvalersi un attaccante per danneggiare o impossessarsi degli asset d' interesse di un sistema.
Nella threat analysis introdotta da *Tameless*, queste associazioni dipendono dalla possibilità di dimostrare o meno una certa implicazione logica e quindi dal verificarsi delle condizioni che la caratterizzano.

Questo stesso motivo è stato replicato nel processo di generazione del grafo.
La rappresentazione utilizzata consiste nell' associare alla porzione sinistra di una regola di derivazione o a un sottoinsieme dei termini in essa contenuti, un set di nodi che coincidano con le concause di una certa implicazione logica. Nel momento in cui il rule engine fosse in grado di verificare le condizioni che consentano l'atto di derivazione, si farà corrispondere la proprietà di sicurezza appena desunta al nodo destinazione e all'arco che conduce dalle concause a questo, l'identificativo della regola applicata.
I nodi e gli archi del grafo vengono aggiunti passo dopo passo, ogni volta che avviene la derivazione di una regola.

Il diagramma mette in luce esclusivamente gli aspetti critici del sistema e non quelli ancora validi.
Ai fini della rappresentazione infatti l'interesse è diretto verso tutti quei nodi compromessi, non funzionanti e vulnerabili o che, pur essendo monitorati o replicati, a causa di falle nei meccanismi di monitoring o restoring perdono queste facoltà. Di conseguenza, alla forma naturale, soprattutto per quanto riguarda le *proprietà ausiliarie* e le *proprietà di alto livello* è stata preferita quella negata.
Per comprendere meglio tale meccanismo è possibile fare riferimento alla figura seguente, che riassume sostanzialmente l'approccio seguito. È interessante come, partendo da una serie di semplici assunzioni e dal modello del sistema si possa, con l'ausilio di formule meccaniche, giungere ad un grafo d'attacco abbastanza significativo:

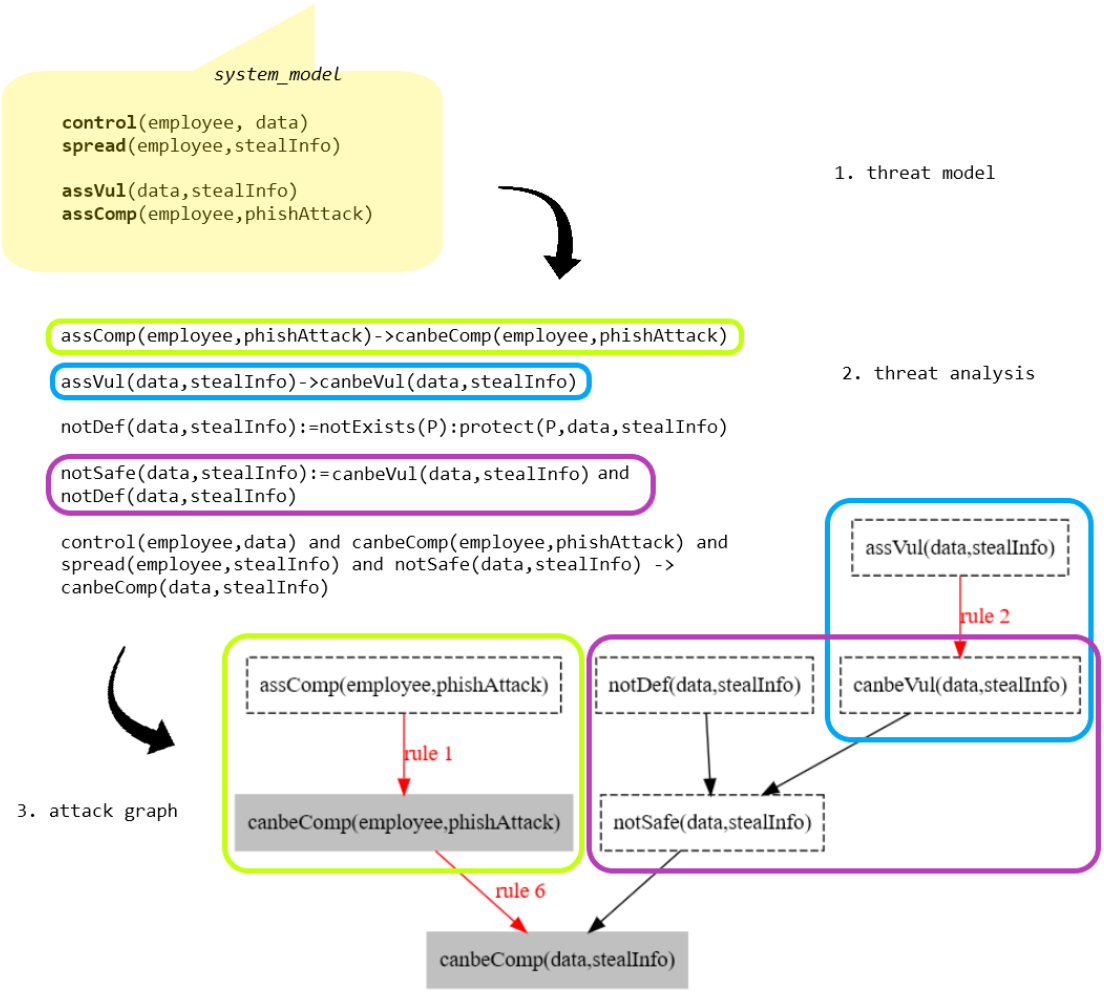



Figura 5.3: Origine del grafo d'attacco.

Quest' approccio si è basato sulla modifica di alcuni meccanismi dell'implementazione originale e al collegamento della stessa ad un insieme di classi Java che ne estendono le funzionalità.
Un nodo può contenere una coppia o una tupla di informazioni a seconda che il termine (definito nel threat model) cui si riferisce sia di tipo binario o ternario, nello specifico: il nome dell' entità e la proprietà di sicurezza che la caratterizza e, se necessario, il threat coinvolto. La classe Java `Node` astrae questo concetto mentre quella denominata `AttackGraph` contiene le funzioni per la creazione di archi e nodi, per l'export del diagramma e mantiene inoltre le strutture dati di supporto a queste e altre operazioni più articolate, come la gestione dei dati temporanei derivanti dall' esecuzione dell' algoritmo.

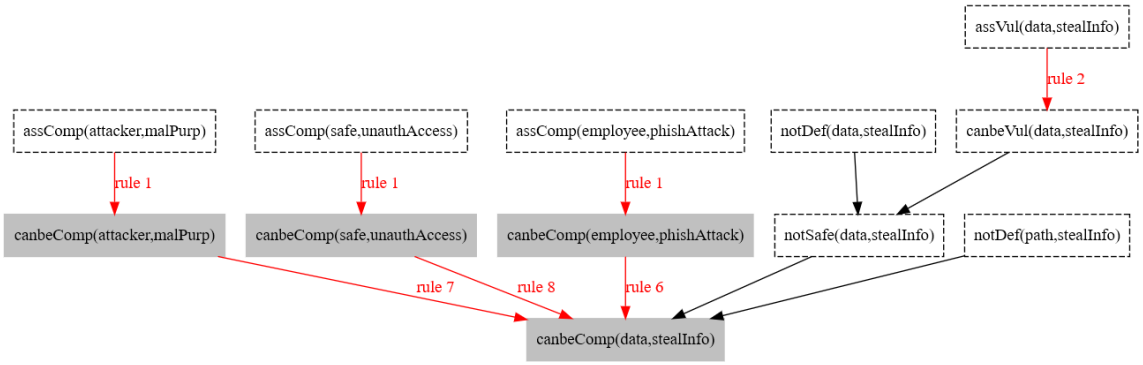



Figura 5.4: È possibile compromettere l'asset "data" sfruttando tecniche diverse.

Tra i vantaggi di questa rappresentazione c'è quello di essere una soluzione intuitiva che riflette la tecnica con cui avviene la derivazione di nuove proprietà all' interno del sistema. Tuttavia, a causa della eterogeneità di ciascuna regola di derivazione non è stato possibile individuare un pattern da applicare univocamente.
Una sfida interessante è stata tradurre nel diagramma le procedure comprendenti i quantificatori, come nel caso della dimostrazione di **Valid** oppure **Defended**: dalla teoria, un'entità è provata essere "difesa" solo qualora esista almeno un meccanismo di protezione valido nei suoi confronti. Un' entità che non goda di questa proprietà quindi, deve comparire nel grafo insieme a tutte le relative protezioni che non siano valide e a tutte le cause che hanno determinato questa condizione. Una serie di informazioni che, per come era stato impostato originariamente il threat model, risultavano difficili da accumulare. Tra gli svantaggi, la confusione nel caso di sistemi molto complessi e con tanti nodi oppure nel momento in cui un elemento può essere vittima di più attacchi perpetrati da più agenti e con diverse tecniche. Un problema a cui avrebbe potuto porre rimedio soltanto un dettagliamento più profondo degli elementi di ciascuna regola, in grado di comprendere anche gli operatori logici.

## 5.3 Remediations

L' obiettivo di questa funzionalità è quello di offrire in maniera automatica diverse soluzioni alle problematiche sorte durante la threat analysis.
Le possibili contromisure variano non solo sulla base delle criticità emerse fino al momento dell' invio della richiesta ma anche sulla particolare struttura del sistema analizzato. Per questo motivo le mitigazioni proposte possono essere puntuali, e.g. "se la componente non è protetta, inserire una protezione..." oppure far riferimento alla condizione o alle condizioni che hanno determinato la perdita della proprietà

di sicurezza d'interesse, e.g. "se la componente è protetta ma le sue protezioni non sono valide, ripristinare almeno una protezione...".
Le tabelle seguenti mostrano la terminologia elementare introdotta per indicare il tipo di contromisura (puntuale) da adottare in ogni situazione, suddivisa in base alla categoria di proprietà di sicurezza di appartenenza.

| Proprietà di base | |
|---|---|
| Compromised(A,T) | removeCompromised(A,T) |
| Malfunctioned(A) | removeMalfunctioned(A) |
| Vulnerable(A,T) | removeVulnerable(A,T) |

| Proprietà ausiliarie | |
|---|---|
| notDetectable(A,T) | makeDetectable(A,T) |
| notRestorable(A) | makeRestorable(A) |
| notFixable(A) | makeFixable(A) |

| Proprietà di alto livello | |
|---|---|
| notValid(A) | makeValid(A) |
| notDefended(A,T) | makeDefended(A,T) |
| notSafe(A,T) | makeSafe(A,T) |
| notReplicated(A) | makeReplicated(A) |
| notMonitored(A,T) | makeMonitored(A,T) |
| notChecked(A) | makeChecked(A) |

A seconda del numero e della tipologia dei percorsi d'attacco esistenti è possibile effettuare una o più scelte che comprendono:

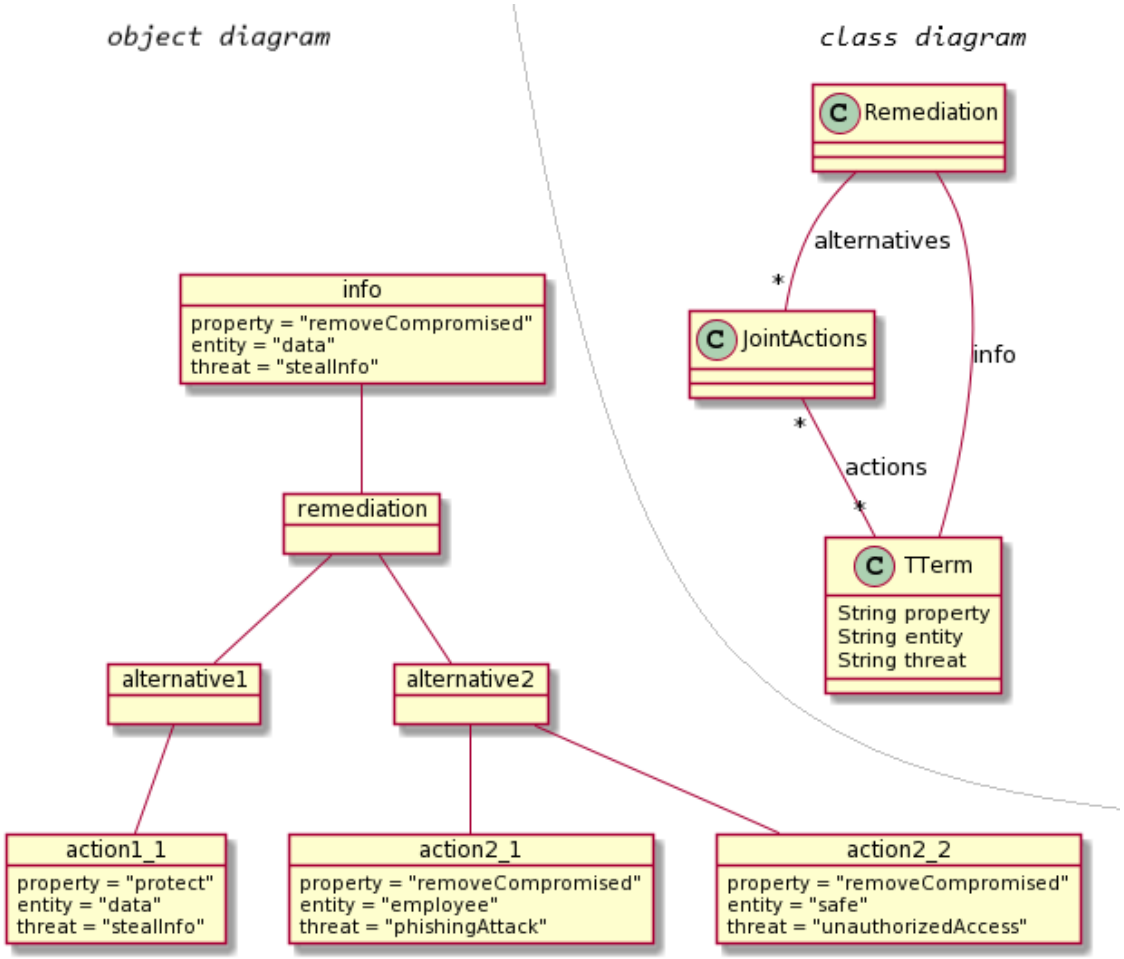
- azioni da eseguire necessariamente tutte insieme, e.g. affinché un nodo (ri)acquisisca la proprietà **Valid** devono essere mitigate tutte le cause di malfunzionamento o compromissione ad esso legate;
- azioni da eseguire alternativamente, e.g. affinché un nodo (ri)acquisisca la proprietà **Defended** è sufficiente che una tra le protezioni che possiede (ri)acquisisca la proprietà **Valid**.

Il concetto di contromisura si concretizza nelle classi `Remediation` e `JointActions` ed il processo di scoperta delle soluzioni disponibili si realizza in parte parallelamente alla costruzione del grafo, in parte al momento della richiesta della contromisura per la specifica entità. Il funzionamento ad alto livello è il seguente:

1. a seconda della regola di derivazione invocata durante la threat analysis, viene creata un'istanza della classe `JointActions`, che rappresenta una possibile sequenza di azioni da eseguire contemporaneamente al fine di mitigare una certa criticità;

2. le `JointActions` vengono accumulate nell' istanza di `Remediation` relativa all'entità di cui si vogliono scoprire le soluzioni.

Di seguito, un esempio della struttura e delle relazioni delle classi Java che realizzano l'approccio e una possibile istanziazione in riferimento ad uno scenario d'attacco.
La classe `TTerm` ha l'obiettivo di contenere gli attributi base dei vari termini dell' analisi (proprietà di sicurezza, nome dell' entità e del threat coinvolti).



# Capitolo 6

# Implementazione

In questo capitolo verranno introdotti gli strumenti che hanno supportato la fase di sviluppo delle estensioni al framework originale di *Tameless*, discutendone le caratteristiche in maniera funzionale rispetto all'applicazione che ne è stata fatta.
Successivamente verrà presentato il codice e i meccanismi dietro la sua particolare implementazione, con l'obiettivo di semplificarne la comprensione e guidare l'integrazione di eventuali sviluppi futuri.

## 6.1 Strumenti

Gli strumenti che hanno accompagnato lo sviluppo del tool sono diversi.
Il punto di partenza è *SWI-Prolog*[1], il rule engine attraverso il quale è stato implementato *Tameless*, mentre in *Java* sono definiti metodi e oggetti che hanno lo scopo di estenderne le funzionalità. La libreria *JPL* è l'interfaccia che fa da ponte tra questi due linguaggi consentendo l'invocazione dei metodi per la manipolazione del grafo d'attacco, a sua volta realizzato utilizzando le strutture dati offerte dalle librerie *JgraphT*.
Il progetto e le sue dipendenze sono infine gestite attraverso *Apache Maven*.

[1] Un'implementazione del linguaggio *Prolog*.

### 6.1.1 SWI-Prolog

*SWI-Prolog* è un' implementazione di *Prolog*, un linguaggio di programmazione basato su logica del primo ordine. Un programma *Prolog* consiste di diversi elementi: fatti, regole e query.
Un **fatto** rappresenta una certa unità di conoscenza che il programmatore ritiene significativa, e.g. "la mela è un cibo", "il riso è un cibo". Alcuni esempi di fatti descritti attraverso la sintassi del linguaggio sono:

```
% contenuti di un file 'test.pl'
cibo(mela).
cibo(riso).
cibo(pasta).
```

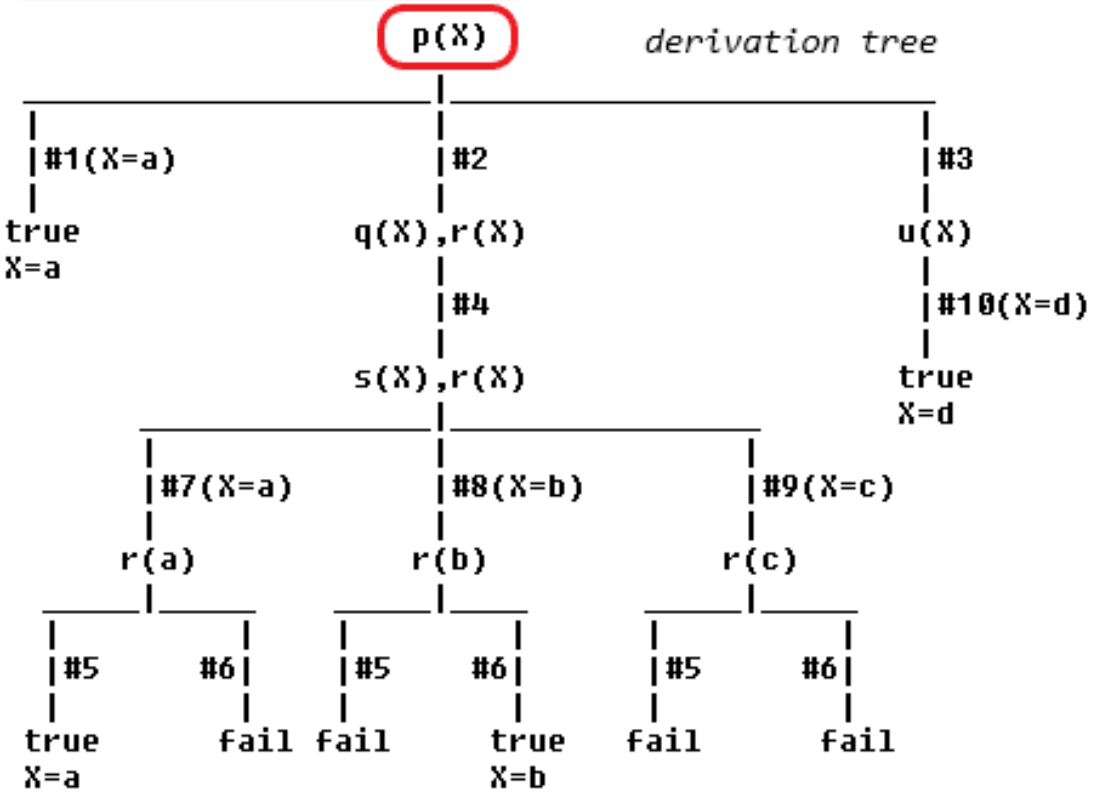
Una collezione di fatti viene caricata nel database attraverso il comando `consult` e contribuisce ad arricchire le conoscenze del rule engine.
Una **query** è il modo attraverso il quale è possibile effettuare un' interrogazione. Essa consiste di uno o più "goal" ossia richieste che *Prolog* tenta di soddisfare trovando valori conosciuti da assegnargli.
La risposta dipende dai fatti che sono stati introdotti fino a tale momento.

```
?- consult('test.pl').  % True (il file è stato caricato)
?- cibo(mela).
True
?- cibo(COSA).
COSA = mela
COSA = pasta
COSA = riso
```

Il caso precedente equivale a chiedere "per quali valori di COSA è possibile dimostrare che COSA è un cibo?".
In generale, il compito del rule engine è quello di tentare di provare un enunciato utilizzando i fatti (e le regole) che gli sono stati somministrati, restituendo un valore vero o falso nel caso di query puntuali o tutti i valori che lo soddisfino nel caso di query che hanno per oggetto delle variabili.
Per comprendere il concetto di **regola** si introduce il seguente esempio[2] corredato da una figura che mostra anche come vengono ottenute le soluzioni.

[2]Tratto da `https://www.cpp.edu/~jrfisher/www/prolog_tutorial/3_1.html`

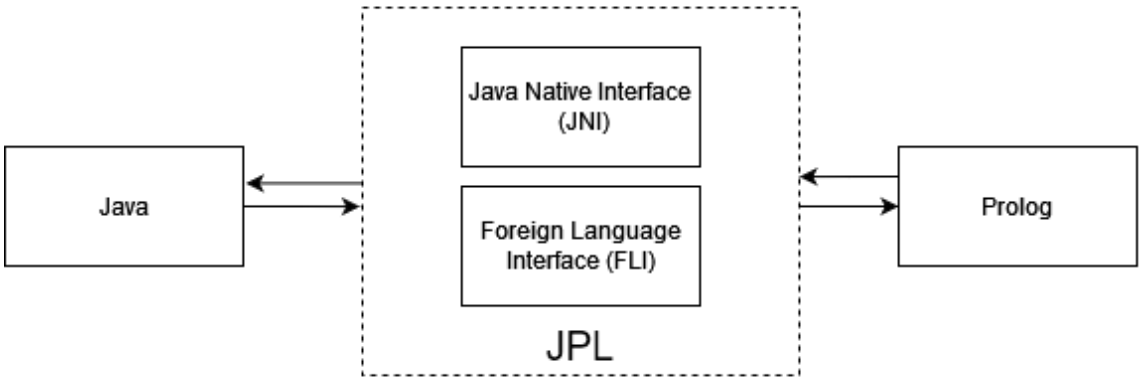
```
% regole
p(a).                  % 1
p(X) :- q(X), r(X).    % 2
p(X) :- u(X).          % 3
q(X) :- s(X).          % 4
% fatti
r(a).                  % 5
r(b).                  % 6
s(a).                  % 7
s(b).                  % 8
s(c).                  % 9
u(d).                  % 10
```



La query `?- p(X).` è equivalente alla richiesta "per quali valori di X è possibile dimostrare p(X)?". Come prima cosa il rule engine cerca se tra i fatti noti ce ne sia uno in grado di soddisfare la query. In questo caso un fatto esiste ed è `p(a)`. Una volta terminati i fatti, il passo successivo è individuare la "testa" delle regole che nuovamente fanno match con la query, rimpiazzare il goal corrente con il corpo della regola selezionata e ricominciare il processo a partire dal goal più a sinistra della nuova regola. `p(X)` può essere sostituito con `q(X), r(X)` (la `,` indica che per provare la testa della regola, entrambi i goal devono essere dimostrati).
Mano a mano che sono effettuati i match il derivation tree cresce e le relative soluzioni vengono restituite. In alternativa, il backtracking consente di risalire l'albero e tentare scelte differenti.

### 6.1.2 JPL

JPL è un set di classi Java e funzioni del C che forniscono un canale bidirezionale tra Java e Prolog[3].
Questa integrazione è resa possibile dalla cooperazione di due moduli: un' interfaccia verso il linguaggio C denominata Foreign Language Interface (FLI) e la Java Native Interface (JNI). La FLI è incaricata di tradurre predicati e tipi primitivi del Prolog rispettivamente in funzioni e tipi del C e viceversa. La JNI consente invece al codice in esecuzione nella Java Virtual Machine di interoperare con il linguaggio C e, di conseguenza, anche con il rule engine di Prolog.
Le funzionalità realizzate nel tool poggiano sullo scambio di informazioni e sull' invocazione innestata dei predicati e dei metodi appartenenti a questi due domini.




[3] `https://jpl7.org/`

L' invocazione dei metodi Java da Prolog avviene attraverso il predicato `jpl_call/4` della rispettiva API, con la seguente firma:

```
jpl_call(+JRef,+Method, +Params, -Result)
```

- **JRef** un riferimento a un oggetto Java per invocarne metodi istanza e statici oppure il nome di una classe Java per invocarne i soli metodi statici;
- **Method** il nome del metodo da invocare;
- **Params** una lista di parametri da passare al momento dell' invocazione;
- **Result** una variabile in cui verrà istanziato (o si tenterà di istanziare) il valore di ritorno del metodo invocato.

Questo predicato è stato usato estensivamente nell'implementazione per accedere all' istanza singleton della classe `AttackGraph` e manipolare la struttura a grafo, parallelamente alla generazione del derivation tree.
L'accesso all'istanza e alle funzioni avviene per nome poiché i metodi coinvolti sono dichiarati statici, i parametri necessari (nome dell' entità, proprietà di sicurezza ed eventualmente threat) sono collezionati all' atto di derivazione e i valori di ritorno gestiti coerentemente.
Di seguito, un esempio di utilizzo nell' implementazione. Qui, le chiamate all' interfaccia JPL sono state nascoste in predicati Prolog per compattarne la scrittura e semplificare la lettura del codice.

```
%%%%%%%%%%%%%%%%%%%%Helper rules%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

addNode(P,E,T,Ref) :-
        jpl_call('it.polito.tameless.AttackGraph', addNode, [P,E,T],Ref).

addEdge(Src,Dst,Label) :-
        jpl_call('it.polito.tameless.AttackGraph', addEdge, [Src,Dst,Label],_).
```

Figura 6.1: Predicati che fanno da wrapper alle chiamate verso JPL e ai metodi nella classe `it.polito.tameless.AttackGraph`.

L' API opposta (da Java verso Prolog) è stata invece utilizzata per le funzioni di apertura e chiusura dei file in input e per l'esecuzione delle query immesse dall' utente dalla linea di comando.
Le classi fornite, supportano la costruzione di oggetti che fanno le veci dei termini e delle query Prolog e fanno riferimento al package `org.jpl7`.

### 6.1.3 JGraphT

JGraphT è una libreria Java che fornisce classi e algoritmi per la manipolazione di vari modelli di grafo.
La libreria descrive l'interfaccia `Graph<V,E>` da cui derivano altre interfacce che affinano i vari vincoli caratterizzanti le diverse categorie di grafo e introduce delle classi concrete che implementano metodi per la creazione, distruzione e aggiunta di vertici e archi.

Il grafo d'attacco dell' implementazione fa riferimento al tipo `DefaultDirectedGraph<Node,RelationshipEdge>` dove la classe `Node` mantiene la tupla di informazioni relativa alla componente generica del sistema analizzato (entità, proprietà di sicurezza e threat) e la classe `RelationshipEdge` ridefinisce `DefaultEdge` per avere un maggiore controllo sulle etichette da associare agli archi del diagramma.

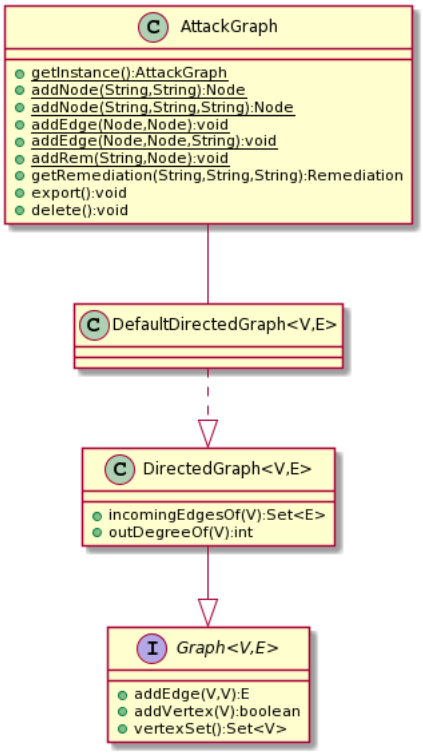



Figura 6.2: Gerarchia di classi del package `org.jgrapht` in relazione con la classe `AttackGraph`.

Tra le funzionalità messe a disposizione da questa libreria c'è inoltre quella di serializzare ed esportare qualsiasi grafo in un formato standard.
Tra questi è stato scelto il linguaggio *DOT*, un linguaggio di descrizione dei grafi

particolarmente comune, con un ampia possibilità di personalizzazione e per il quale sono disponibili diversi tool di processing.

## 6.2 Codice

Il tool si presenta come un' applicazione Java eseguibile da linea di comando.
Una volta avviata, consente di selezionare tra due modalità di lavoro: semplice o aumentata. A seconda della scelta effettuata verranno inseriti file con regole strutturate in maniera diversa: nel caso semplice, il file utilizzato come nucleo del reasoning engine (*'rule.p'*) coincide con quello originale, pertanto l'interazione è identica a quella offerta dalla comune interfaccia *SWI-PROLOG*. Nel caso complesso, il file utilizzato (*'rule2.p'*) è quello aumentato delle chiamate all' interfaccia *JPL* e ai metodi in `AttackGraph`. Qui è possibile eseguire la threat analysis a seguito dell' inserimento del file di descrizione del sistema, esportare in formato *Dot* il grafo generato e ottenere le contromisure per le criticità che sono state evidenziate. La gestione del menù e la navigazione tra le varie selezioni che è possibile effettuare è delegata alla classe `App`, contenente il metodo `main`, da cui si diparte il flusso d'esecuzione.
Nei paragrafi successivi verrà presentato il codice dietro questi comandi.

```
$ Type 1 to start simple mode.  Type 2 to start augmented mode.  Type
<exit> to exit.
> 2
$ Waiting for input:  <open> or <query> or <+query> or <export> or
<restart> or <back>.
> open test.p
$ Waiting for input:  <open> or <query> or <+query> or <export> or
<restart> or <back>.
> canbeComp(pc, malware)
$ Query succeded.
$ Waiting for input:  <open> or <query> or <+query> or <export> or
<restart> or <back>.
> export
$ Correctly exported.
```

Figura 6.3: Esempio di navigazione da linea di comando.

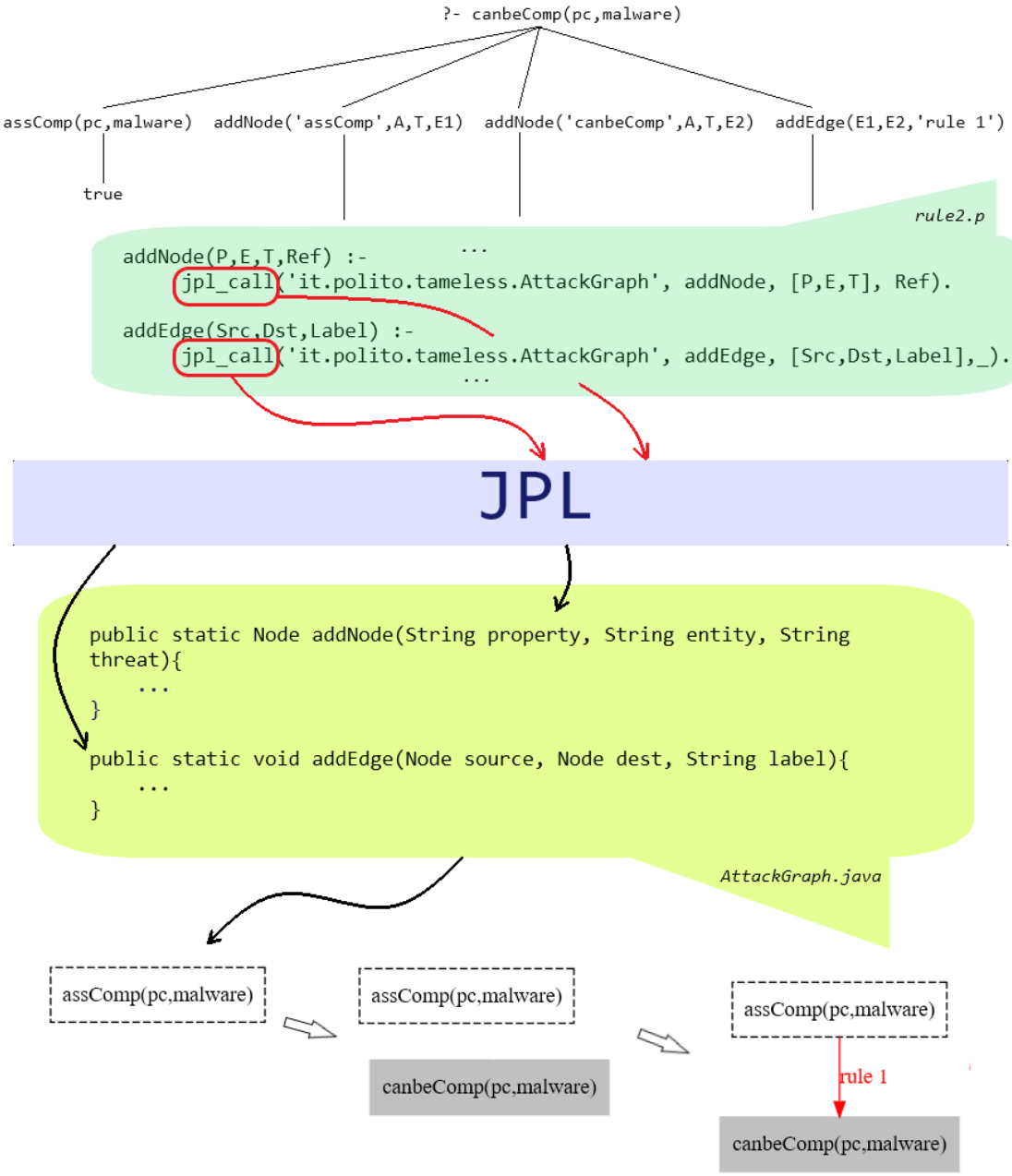
### 6.2.1 Attack Graph

Per costruire il grafo d'attacco, ai goals originali vengono affiancate delle "azioni" - goals non rilevanti ai fini dell' analisi - che, con l'ausilio dell' interfaccia JPL, realizzano dei task particolari.
Una possibile alternativa alla generazione "on demand" del grafo consisteva nel tracciare l'esecuzione di *Prolog* attraverso i comandi `trace` e `notrace` invocati rispettivamente prima e dopo l'immissione di una query e salvarne l'output su un file. In questo modo sarebbe stato uno stacktrace con il dettaglio dei goal eseguiti e dei risultati ottenuti nell' ambito della query . Il file sarebbe stato poi utilizzato per indagare lo storico delle regole utilizzate e dedurre conseguentemente il grafo, sapendo che questo è correlato ad una una ricerca in profondità sull' albero di derivazione.
Questo avrebbe consentito, in conclusione, di avere meno modifiche sul nucleo orginale ma avrebbe appesantito l'interfaccia utente e i tempi d'esecuzione, cui si sarebbero aggiunti quelli per l'I/O del file di trace. In più il parsing di un file del genere avrebbe richiesto metodi di processamento differenti rispetto a quelli previsti.
Il comando `trace` ad ogni modo non è stato del tutto abbandonato e si è rivelato utile in fase di validazione per verificare che non ci fossero incongruenze tra il diagramma ottenuto e lo stack delle computazioni effettuate.

Per comprendere il meccanismo alla base dell' approccio scelto è utile fare un semplice esempio.
Di seguito viene presentata una delle regole fondamentali del modello. Nel corpo della regola, in blu, è evidenziato il goal originale mentre in giallo sono evidenziati quelli aggiunti. Questi ultimi sono dei wrapper per le chiamate alla libreria JPL, visti precedentemente.

```
canbeComp(A,T):-
                assComp(A,T),
                addNode('assComp',A,T,E1), % crea oppure recupera il nodo sorgente
                addNode('canbeComp',A,T,E2), % crea il nodo destinazione
                addEdge(E1,E2,'rule 1'). % crea l'arco tra i due nodi e gli assegna un' etichetta
```

Tenendo a mente la struttura di questa regola si consideri uno scenario d'analisi che valuta la possibilità per un pc di essere compromesso da malware o nel gergo del modello si assuma $\alpha$`Comp(pc,malware)`.
Al momento dell' immissione della query `?- canbeComp(pc,malware)` il Prolog engine si adopera per dimostrare tale enunciato, che risulterà vero per l' esecuzione della regola di cui sopra. Il flusso è il seguente:


?- canbeComp(pc,malware)
assComp(pc,malware)
addNode('assComp',A,T,E1)
addNode('canbeComp',A,T,E2)
addEdge(E1,E2,'rule 1')
true
rule2.p
addNode(P,E,T,Ref) :-
jpl_call('it.polito.tameless.AttackGraph', addNode, [P,E,T], Ref).
addEdge(Src,Dst,Label) :-
jpl_call('it.polito.tameless.AttackGraph', addEdge, [Src,Dst,Label],_).
JPL
public static Node addNode(String property, String entity, String threat){
public static void addEdge(Node source, Node dest, String label){
AttackGraph.java
assComp(pc,malware)
assComp(pc,malware)
assComp(pc,malware)
canbeComp(pc,malware)
rule 1
canbeComp(pc,malware)

Il dominio iniziale è quello del Prolog, successivamente l'esecuzione del predicato `jpl_call` dà il via alla transizione verso il linguaggio Java ed in particolare verso la classe `AttackGraph` che definisce i metodi per la manipolazione del grafo d'attacco. I nodi del grafo sono modellati attraverso la classe `Node` mentre gli archi dalla classe `RelationshipEdge`.

```
public static Node addNode(String property, String entity, String threat){
    Graph<Node, RelationshipEdge> _graph = _instance.graph;
    Map<String, Node> _nodes = _instance.nodes;
    String key = property+"_"+entity+"_"+threat;
    if(_nodes.containsKey(key))
        return _nodes.get(key);
    Node newNode = new Node(property, entity, threat);
    _nodes.put(key, newNode);
    _graph.addVertex(newNode);
    return newNode;
}

public static void addEdge(Node source, Node dest, String label){
    Graph<Node, RelationshipEdge> _graph = _instance.graph;
    if (!_graph.containsEdge(source,dest))
        _graph.addEdge(source,dest,new RelationshipEdge(label));
}
```

Figura 6.4: Dettaglio dei metodi Java per la creazione di nodi e archi.

Le difficoltà incontrate nell'utilizzare quest' approccio hanno riguardato principalmente la diversa interpretazione delle regole originali.
Per generare il grafo è stato infatti necessario ottenere la maggior parte delle informazioni non dalla dimostrazione dei singoli goal contenuti nei predicati, ma dal loro fallimento. Un capovolgimento rispetto alle intenzioni iniziali.
Per mantenere l'uniformità con il threat model di *Tameless* in alcuni casi si è reso necessario cambiare la forma di alcune regole e ottenere informazioni cui in prima istanza non era stato dato peso ma indispensabili per una corretta rappresentazione dei percorsi d'attacco.
Sono state oggetto di riscrittura in particolare le regole che coinvolgevano i quantificatori, presenti nella definizione di tutte le *proprietà di alto livello* del threat model:

```
%originali
usable(A):-
  e(A),(\+canbeComp(A,_T), \+canbeMalfun(A)).
protected(A,T):-
  protect(B,A,T), usable(B).
monitored(A,T):-
  monitor(B,A,T), usable(B).
replicated(A):-
  replica(B,A), usable(B).
checked(A):-
  check(B,A), usable(B).

%modificate (le chiamate a JPL sono state omesse)
usable(A):-
  e(A),(findall(_, (canbeComp(A,_T);canbeMalfun(A)), []).
protected(A,T):-
  protect(B,A,T), (findall(_, usable(B), [])) -> fail; true).
monitor(A,T):-
  monitor(B,A,T), (findall(_, usable(B), [])) -> fail; true).
replicated(A):-
  replica(B,A), (findall(_, usable(B), [])) -> fail; true).
checked(A):-
  check(B,A), (findall(_, usable(B), [])) -> fail; true).
```

La presenza dell' operatore di negazione "\+" si è rivelata ad esempio un ostacolo all'applicazione dell' algoritmo.
L' implementazione in *Prolog* di tale operazione è nota come "negation as failure", un meccanismo in grado di fermare il backtracking e concludere prematuramente la ricerca delle soluzioni. Questo comportamento non consentiva di accumulare ad esempio tutte le cause che rendevano `notValid` un' entità ma solo ed esclusivamente la prima. Per ovviare a questo problema si è scelto di usare un espediente sintattico che evitasse di intaccare il flusso di ricerca, restituendo invece un valore da poter usare.
Il predicato `findall/3` ha ovviato al problema. L'obiettivo ottenuto attraverso quest'ultimo è stato quello di accumulare certe informazioni riguardo l'entità specificata, e.g. i threat in grado di comprometterla o i malfunzionamenti ad essa legati, e agire conseguentemente alla valutazione di tali informazioni, e.g. assegnare all' entità la proprietà `notValid` e aggiungerla al grafo.

Un'altra problematica è risultata essere quella della rappresentazione delle porzioni di grafo derivanti da regole che esprimessero l'esistenza di almeno una o esattamente N determinate condizioni.
Alcuni esempi sono quelli delle regole `Protected` (`Defended`) oppure `Monitored`, dimostrabili soltanto se almeno una delle entità ad un estremo delle rispettive relazioni ("protect" e "monitor") sia `Valid`. Per questo scopo sono state inserite in Java ulteriori funzioni con l'obiettivo di memorizzare ed utilizzare dati temporanei derivanti da queste valutazioni.

```
public static void mem(String property, String entity, String threat, String B){
    Map<TTerm,List<String>> _map = _instance.map;

    TTerm term = threat==null? new TTerm(property, entity) : new TTerm(property, entity, threat);
    List<String> res = _map.get(term);

    if (res == null) {
        res = new ArrayList<>();
        _map.put(term, res);
    }
    res.add(B); // add B to entities that protect term (A)
}
public static void ret(String property, String entity, String threat, Node notDef) {
    Map<String, Node> _nodes = _instance.nodes;
    List<String> entities = _instance.entitiesWithRelation(property, entity, threat);

    if (entities!=null){
        for (String target : entities){
            Node notValid = _nodes.get("notValid_" + target);
            if (notValid!=null)
                addEdge(notValid, notDef,
                            property + "(" + (threat == null ? entity : threat) + ")");
        }
    }
}
```

Queste funzioni vengono invocate ad esempio nel seguente caso.

```
protected(A,T):-
 e(A), t(T), ( findall(B,
 (protect(B,A,T),(usable(B)-> true; mem('protect',A,T,B), fail))
 , [])
 -> addNode('notDef',A,T,E1), addRem('makeDefended',E1),
ret('protect',A,T,E1), fail; true ).
```

È possibile dividere i contenuti di questa regola in due parti.
Nella prima, attraverso `findall` vengono valutate tutte le entità in rapporto di "protect" con l'entità target (`A`). Qui si è interessati soltanto a quelle che non siano valide (dovranno comparire nel grafo in quanto fonte di percorsi d'attacco) e che, grazie anche all'operatore `->` (in sostanza un if-else), vengono salvate in una mappa all'interno della classe `AttackGraph` attraverso il metodo `mem`. Una volta che si sarà verificato che tra le protezioni enumerate, nemmeno una soddisfi la condizione di essere `Valid` sarà possibile attingere, attraverso la funzione `ret` dalla stessa mappa, per collegare i nodi all'entità che avrebbero dovuto proteggere.

### 6.2.2 Remediation

Una volta conclusa la threat analysis è possibile interrogare il tool attraverso una serie di query realizzate ad-hoc e ottenere informazioni su come comportarsi per mitigare le criticità emerse.

Una contromisura consiste in un insieme di azioni da intraprendere per ripristinare un'entità che ha visto mutare lo stato di una o più proprietà d'interesse. Queste possono essere azioni puntuali oppure insiemi di misure da attuare per far fronte a diverse potenziali minacce su una stessa entità del sistema. Nel contesto di Tameless, la classe `Remediation` modella le possibili sequenze di azioni da mettere in campo per eliminare una certa criticità. Per semplificare la manipolazione della classe suddetta e l'aggregazione delle alternative è stata introdotta anche la classe `JointActions` che concretizza la "sequenza di azioni" di cui si è parlato precedentemente. Queste alternative vengono accumulate in parte parallelamente alla generazione del derivation tree e parte "on demand", come risultato della particolare conformazione del diagramma.

Il tool offre le possibili alternative ma è l'utilizzatore finale a dover scegliere quella d'interesse. Questa potrà essere una contromisura diretta all'entità che presenta la criticità oppure a un nodo predecessore che direttamente o indirettamente è responsabile della propagazione di una probabile minaccia.

Il funzionamento del meccanismo di aggiunta di una contromisura è il seguente:

1. durante la fase di threat analysis, a seconda della regola di derivazione applicata, una nuova alternativa viene aggiunta al set di contromisure relative a una certa entità utilizzando i predicati `addRem/2`, e.g. se l'entità è dimostrata essere `notSafe` le possibili azioni da intraprendere comprendono il patch della

vulnerabilità di cui soffre oppure l'introduzione di una qualche protezione sul nodo;

2. al momento dell' immissione della query che ritorna le mitigazioni plausibili, nel caso delle proprietà di base, viene effettuata una scansione delle entità collegate a quella in oggetto. Questo è necessario per proporre, come contromisura alternativa, l'eliminazione di ogni nodo compromesso nel vicinato.

# Capitolo 7

# Validazione

In questo capitolo verranno affrontati degli use case per validare i diversi aspetti del threat model introdotto.

## 7.1 Use case 1

Il primo use case descrive uno scenario base in cui può essere utilizzato *Tameless*. Si parte con il modellare gli elementi che compongono il sistema e definire le proprietà di ciascuno, quindi viene eseguita la threat analysis per generare il grafo ed infine viene interrogato il database del tool per ottenere le contromisure.

Il contesto descritto è quello di una centrale elettrica in cui i vari organi fisici sono connessi e controllati da un sistema SCADA. L'accesso al sistema è consentito sul posto agli utenti autorizzati e da remoto a coloro che dispongono di credenziali di livello amministratore, così da evitare intrusioni da parte di malintenzionati.

```
e(scadaSystem).
e(adminCredentials).
e(employee).
e(attacker).

t(unauthAccess).
t(malPurp).
t(stealCredentials)
```

```
control(employee,adminCredentials).
assVul(adminCredentials,stealCredentials).
spread(employee,stealCredentials).

protect(adminCredentials,scadaSystem,unauthAccess).
connect(remoteAccess,attacker,scadaSystem).
spread(attacker,unauthAccess).
assVul(scadaSystem,unauthAccess).
assComp(attacker,malPurp).
% ...
```

Attraverso lo SCADA è possible gestire attività critiche per gli scopi della centrale. Queste attività vengono coordinate grazie a una rete locale che mette in connessione il sistema di monitoraggio con l'infrastruttura fisica. È ovviamente possibile, intenzionalmente o a causa di un guasto, provocare un' interruzione dell'energia elettrica.

```
% ...
e(electricalSubstation).

t(powerOutage).

connect(localNetwork,scadaSystem,electricalSubstation).
spread(scadaSystem,powerOutage).
assVul(electricalSubstation,powerOutage).
% ...
```

Dalle precedenti definizioni è possibile già sfruttare alcune delle regole di derivazione per ottenere ulteriori informazioni circa lo stato di sicurezza iniziale del sistema.
Esistono delle vulnerabilità intrinseche in alcune entità del sistema ma non le condizioni, per un attaccante, di trarne vantaggio. Anche le misure di protezione messe in piedi sono sufficienti a ritenere sicure le risorse più rilevanti, pertanto è possibile concludere che il sistema è globalmente protetto.
Dalle formule si ottiene ad esempio:

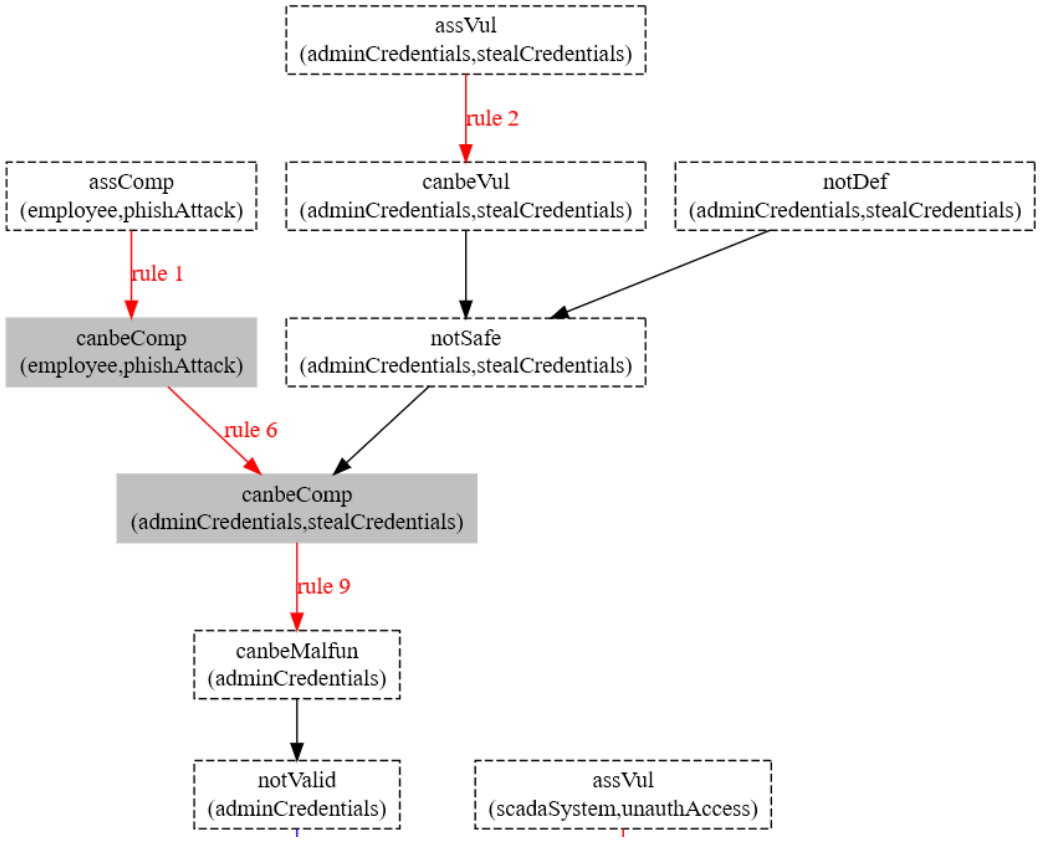
```
¬Safe(adminCredentials) = assVul(adminCredentials,stealCredentials)∧
      ¬Def(adminCredentials,stealCredentials)
```

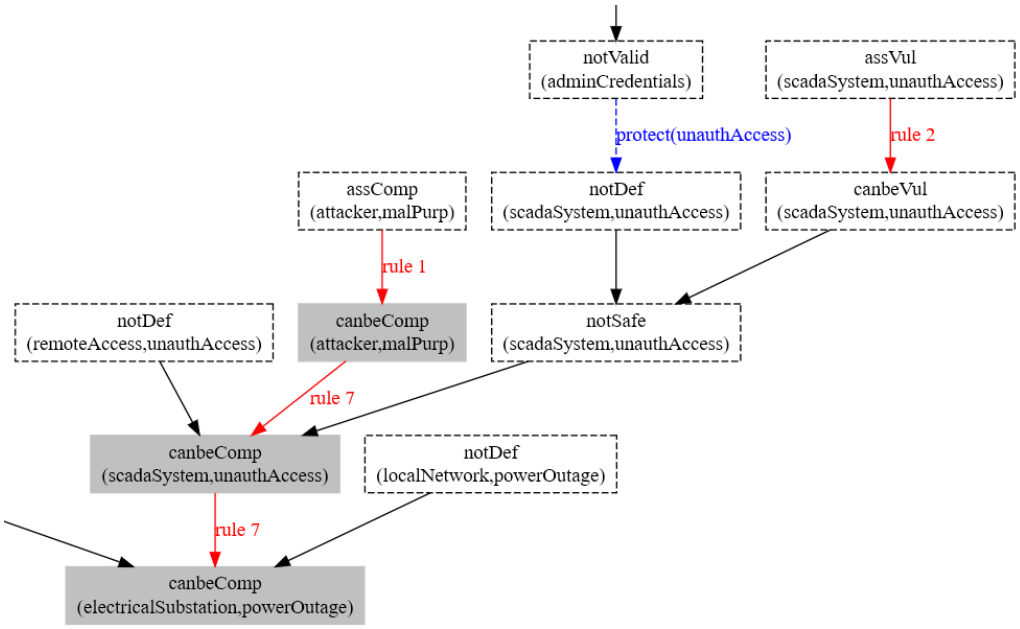
```
Def(scadaSystem,unauthAccess) =
      protect(adminCredentials,scadaSystem,unauthAccess)∧
      Valid(adminCredentials)
```

Introducendo nel sistema una singola variazione ed effettuando le relative modifiche alla descrizione iniziale, la situazione cambia. Le relazioni tra le varie componenti determinano una cascata di eventi che si trasforma in un potenziale percorso d'attacco per l'attaccante.

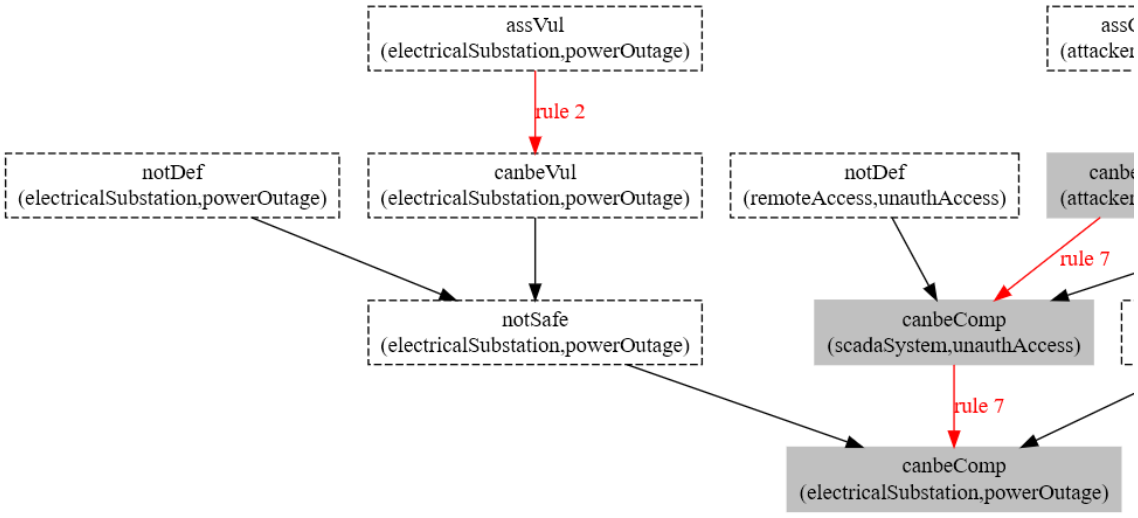
```
% ...
t(phishAttack).
assComp(employee,phishAttack).
```

Assumere che l'impiegato sia stato vittima di un attacco di phishing implica che le credenziali (oramai non più) sotto il suo controllo non possano più essere considerate un presidio di sicurezza valido nei confronti degli accessi non autorizzati






l'attaccante può quindi introdursi all' interno del sistema e, sfruttando i collegamenti dello SCADA con il resto dell' infrastruttura fisica, attraverso la rete locale, è ora in grado di compromettere il funzionamento dell' intero sottosistema elettrico determinando eventualmente un' interruzione di corrente



Osservando il grafo d'attacco è possibile individuare quali proprietà si desidera ripristinare e su quali nodi si intende farlo. Ad esempio con la query seguente si può interrogare il tool per ottenere informazioni sulle contromisure da attuare per mitigare la criticità `powerOutage` sull'entità `electricalSubstation`:

```
$ +makeNotComp(electricalSubstation,powerOutage)
```

Il risultato è un'istanza della classe *Remediation* serializzata in formato JSON. Tra le info compare lo mnemonico del comando insieme all' entità e al threat d'interesse; seguono un insieme di alternative dove ogni alternativa è invece un'istanza dell' oggetto *JointActions*:

```
{
  "info" : {
    "property" : "removeComp",
    "entity" : "electricalSubstation",
    "threat" : "powerOutage"
  },
  "alternatives" : [ {
    "actions" : [ {
      "property" : "makeSafe",
      "entity" : "electricalSubstation",
      "threat" : "powerOutage"
    } ]
  }, {
    "actions" : [ {
      "property" : "makeDetectable",
      "entity" : "electricalSubstation",
      "threat" : "powerOutage"
    } ]
  }, {
    "actions" : [ {
      "property" : "removeComp",
      "entity" : "scadaSystem",
      "threat" : "unauthAccess"
    } ]
  } ]
}
```

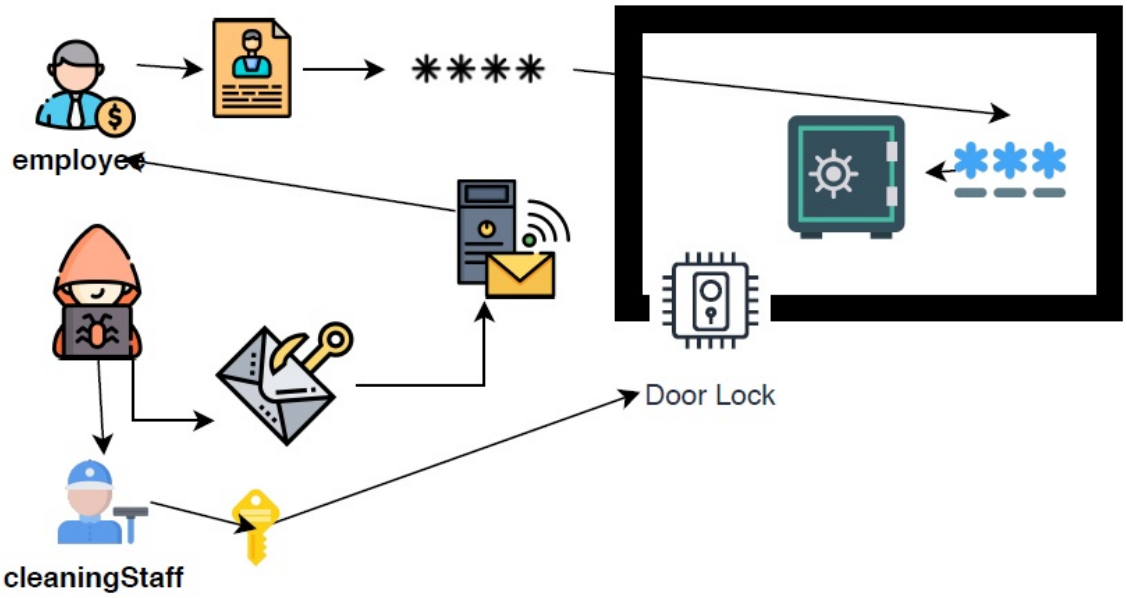
Il meccanismo si ripropone sulle alternative ottenute. In questo modo è possibile scegliere la combinazione di soluzioni preferita per risolvere le criticità più rilevanti all' interno del sistema.

```
$ +makeSafe(electricalSubstation,powerOutage)
```

```
{
  "info" : {
    "property" : "makeSafe",
    "entity" : "electricalSubstation",
    "threat" : "powerOutage"
  },
  "alternatives" : [ {
    "actions" : [ {
      "property" : "removeVul",
      "entity" : "electricalSubstation",
      "threat" : "powerOutage"
    } ]
  }, {
    "actions" : [ {
      "property" : "makeDefended",
      "entity" : "electricalSubstation",
      "threat" : "powerOutage"
    } ]
  } ]
}
```

## 7.2 Use case 2

Lo scenario che si vuole analizzare in questo secondo caso è rappresentato dalla seguente figura.




Una cassaforte protetta da un PIN si trova all' interno di una stanza chiusa da una serratura che non è possibile forzare. Un attaccante interessato ai contenuti di tale cassaforte deve conoscere il PIN, noto solo al personale autorizzato e possedere la chiave per accedere al locale in cui è custodita, maneggiata, oltre che dallo stesso personale, anche agli addetti alle pulizie.

```
e(employee).
e(attacker).
e(cleaningStaff).
e(pinSafeLock).
e(safeLock).
e(keyDoorLock).
e(doorLock).
e(safe).

t(unauthorizedAccess).

connect(path,attacker,safe).
```

```
control(keyDoorLock,doorLock).
control(pinSafeLock,safeLock).
protect(doorLock,path,unauthorizedAccess).
protect(pinSafeLock,safe,unauthorizedAccess).

assVul(safe,unauthorizedAccess).
% ...
```

Si assume che il PIN che blocca la cassaforte sia debole, in quanto fortemente legato ai dati personali dell' impiegato che lo ha settato e che questi dati siano ottenibili attraverso un mirato attacco di phishing. Si assume inoltre che sia possibile recuperare, corrompendo lo staff delle pulizie, una copia della chiave del locale contenente la cassaforte. In pratica, l' attaccante può sfruttare rispettivamente una vulnerabilità cyber ed una umana per violare questa risorsa.

```
% ...
e(phishingMail).
e(employeeData).
e(safeLock).
e(keyDoorLock).
e(doorLock).
e(safe).

t(corruption).
t(stealingInformation).
t(derivationFromAssociation).
t(copy).

connect(logicalAssociation, employeeData, pinSafeLock).
connect(serverMail, phishingMail, clientMailPc).
control(attacker,phishingMail).
spread(attacker,corruption).
spread(cleaningStaff,copy).
spread(employee, stealingInformation).

assVul(pinSafeLock, derivationFromAssociation).
assVul(employee, stealingInformation).
```

```
assComp(phishingMail,attackerPurpose).
% ...
```

L'attack graph evidenzia come la possibilità di violare i sistemi di protezione sfruttando le vulnerabilità presentate consenta all'attaccante di arrivare all' obiettivo.
Le figure seguenti mostrano un attacco letteralmente ai fianchi del sistema e un accesso semplice alla risorsa target.

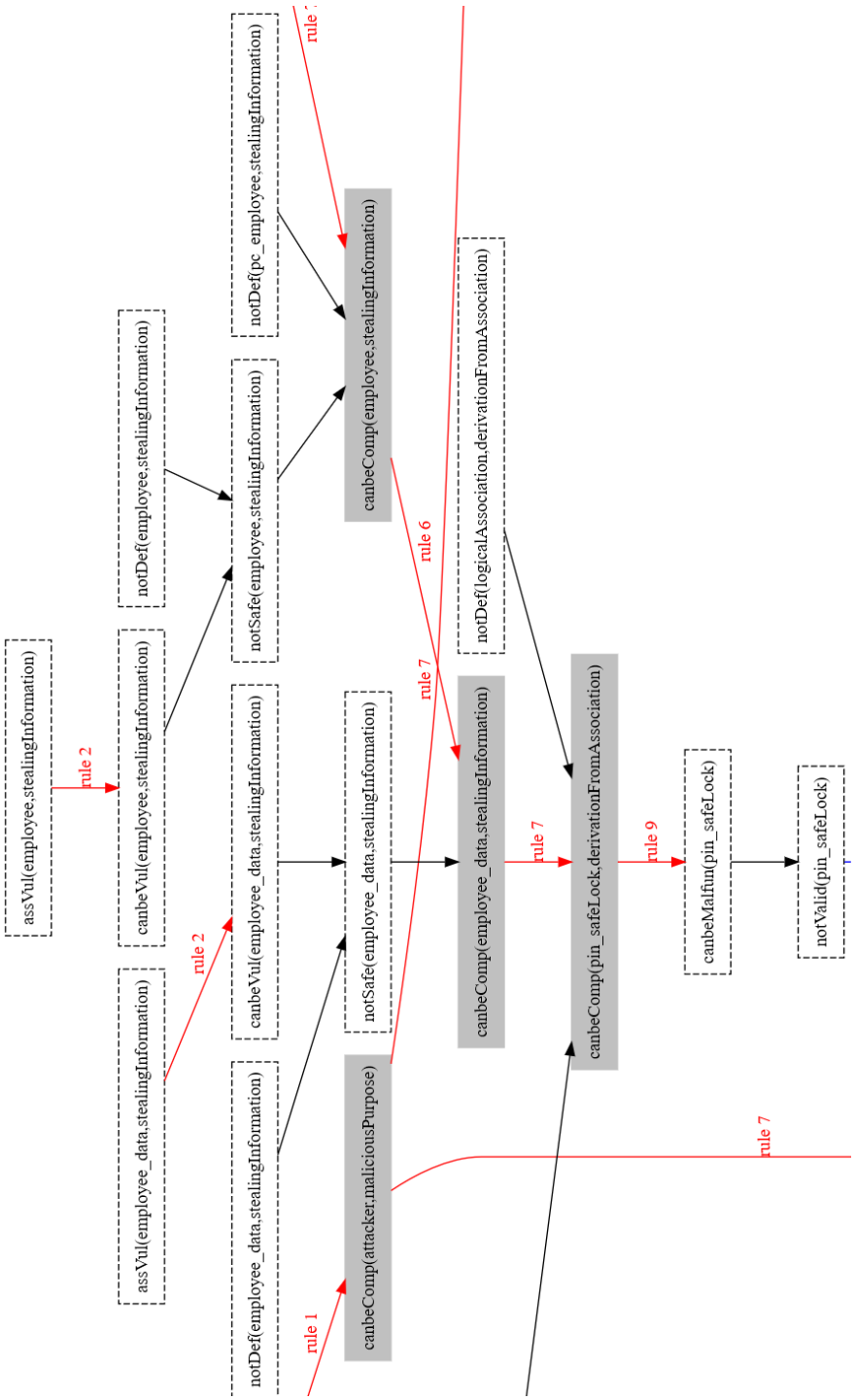



Figura 7.1: Dettaglio dell' attack path che rende invalida la protezione con PIN.

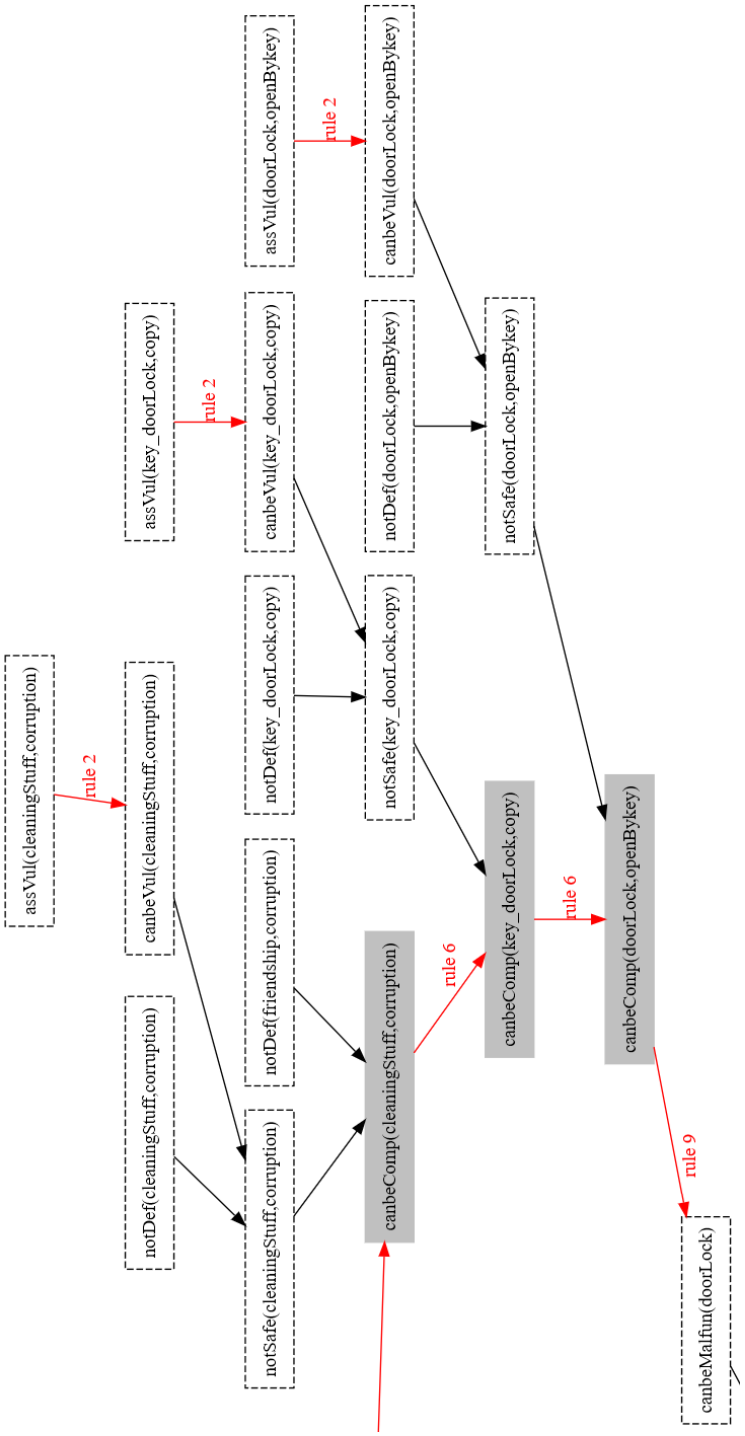



Figura 7.2: Dettaglio dell' attack path che rende invalida la protezione con chiave.

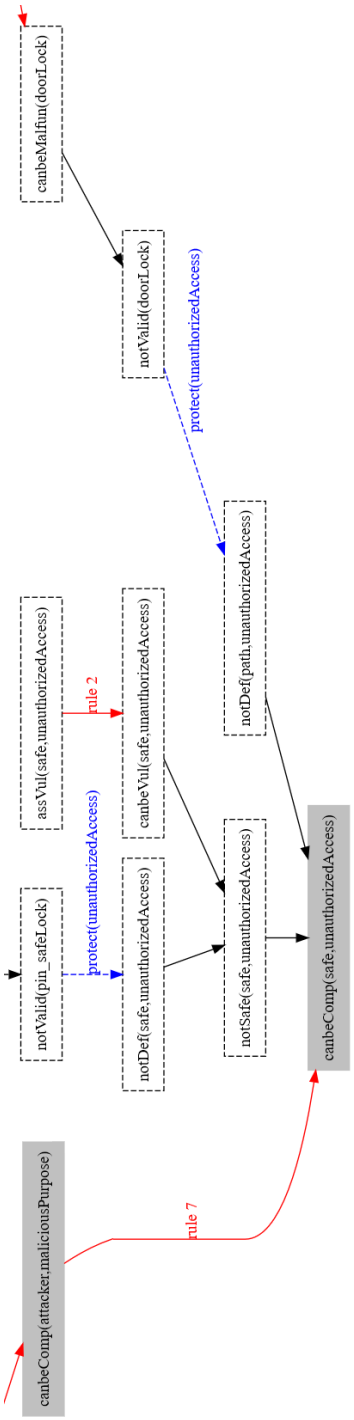



Figura 7.3: Dettaglio sull'azione dell'attaccante che ha portato il fallimento delle protezioni e alla compromissione della cassaforte.

```
$ +removeComp(safe,unauthorizedAccess)
```

```
{
  "info" : {
    "property" : "removeComp",
    "entity" : "safe",
    "threat" : "unauthorizedAccess"
  },
  "alternatives" : [ {
    "actions" : [ {
      "property" : "makeSafe",
      "entity" : "safe",
      "threat" : "unauthorizedAccess"
    } ]
  }, {
    "actions" : [ {
      "property" : "removeComp",
      "entity" : "attacker",
      "threat" : "maliciousPurpose"
    } ]
  }, {
    "actions" : [ {
      "property" : "makeDetectable",
      "entity" : "safe",
      "threat" : "unauthorizedAccess"
    } ]
  } ]
}
```

```
$ +makeDefended(safe,unauthorizedAccess)
```

```
{
  "info" : {
    "property" : "makeDefended",
    "entity" : "safe",
    "threat" : "unauthorizedAccess"
  },
  "alternatives" : [ {
    "actions" : [ {
      "property" : "protect",
      "entity" : "safe",
      "threat" : "unauthorizedAccess"
    } ]
  }, {
    "actions" : [ {
      "property" : "makeValid",
      "entity" : "pin_safeLock"
    } ]
  } ]
}
```

# Capitolo 8

# Conclusioni

Il lavoro di questa tesi si è concentrato sull' estensione delle funzionalità di un threat modeling tool destinato a sistemi cosiddetti “ibridi”. Con questo termine si fa riferimento a quella categoria di sistemi, principalmente Smart, che poggiano su un' infrastruttura eterogenea basata sull'interconnessione e l'interazione di elementi di tipo cyber, fisico e umano.

La novità in letteratura è presentata dal focus di questo threat model su una caratterizzazione degli elementi indipendente dal dominio di appartenenza e fortemente influenzata dalle relazioni che ognuno instaura con le altre parti del sistema e con le potenziali minacce.

Le funzionalità aggiunte al framework originale sono due. La prima consiste nella possibilità di generare un grafo d'attacco parallelamente alla threat analysis condotta, interrogando il reasoning engine basato su Prolog: i meccanismi offerti dal linguaggio vengono aumentati dall'espressività del Java per affiancare al processo di generazione del derivation tree l'istanziazione delle strutture dati necessarie a presentare il diagramma. La seconda funzionalità permette di individuare per ogni entità e per ogni proprietà di sicurezza,l' insieme di contromisure da attuare per ripristinarne le condizioni verso uno stato considerato ”sicuro”, sfruttando i risultati della threat analysis condotta e la conformazione del grafo ottenuto.

Per gli sviluppi futuri ci sono diverse possibilità che spaziano dal semplificare i passi iniziali del threat model, che prevedono di fornire una descrizione manuale del sistema sotto esame, all' estenderne ulteriormente le capacità ponendo il focus sull' introduzione di variabili probabilistiche legate al successo o al fallimento di determinate regole o di aspetti legati alla resilienza delle componenti in gioco.

# Bibliografia


[1] Wikipedia contributors. *Smart system — Wikipedia, The Free Encyclopedia*. 2021. URL: https://en.wikipedia.org/w/index.php?title=Smart_system%5C&oldid=1009818233.

[2] Philip Ross e Kasia Maynard. "Towards a 4th industrial revolution". In: *Intelligent Buildings International* 0.0 (2021), pp. 1–3. DOI: 10.1080/17508975.2021.1873625. URL: https://doi.org/10.1080/17508975.2021.1873625.

[3] Business Innovation Observatory. *Service Innovation for Smart Industry*. 2015. URL: https://ec.europa.eu/docsroom/documents/13392/attachments/2/translations/en/renditions/native.

[4] Fadi Al-Turjman e Muhammad Imran. *8.1 Introduction*. 2020. URL: https://app.knovel.com/hotlink/khtml/id:kt012941L1/iot-technologies-in-smart/cybersecur-introduction.

[5] The EU Agency for Cybersecurity. *Enisa Good Practices for Security of Smart Cars*. 2019. URL: https://www.enisa.europa.eu/publications/smart-cars/view/++widget++form.widgets.fullReport/@@download/Good+practices+for+security+of+Smart+Cars.pdf.

[6] Charlie Miller and Chris Valasek. *Remote Exploitation of and Unaltered Passenger Vehicle*. 2015. URL: http://illmatics.com/Remote%5C%20Car%5C%20Hacking.pdf.

[7] Dennis Giese. *Reverse Engineering and Hacking of Xiaomi IoT Devices*. 2018. URL: https://dontvacuum.me/talks/DEFCON26/DEFCON26-Having%5C_fun%5C_with%5C_IoT-Xiaomi.pdf.

[8] The EU Agency for Cybersecurity. *Industry 4.0 Cybersecurity: challenges and recommendations*. 2019. URL: https://www.enisa.europa.eu/publications/industry-4-0-cybersecurity-challenges-and-recommendations.

[9] Grady Caitlin, Rajtmajer Sarah e Dennis Lauren. "When Smart Systems Fail: The Ethics of Cyber–Physical Critical Infrastructure Risk". In: *IEEE Transactions on Technology and Society* 2.1 (2021), pp. 6–14. DOI: 10.1109/TTS.2021.3058605.

[10] Md. Mahmud Hossain, Maziar Fotouhi e Ragib Hasan. "Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things". In: *2015 IEEE World Congress on Services*. 2015, pp. 21–28. DOI: 10.1109/SERVICES.2015.12.

[11] The EU Agency for Cybersecurity. *Towards secure convergence of Cloud and IoT*. 2018. URL: https://www.enisa.europa.eu/publications/towards-secure-convergence-of-cloud-and-iot.

[12] Verizon. *2021 Data Breach Investigations Report*. 2021. URL: https://enterprise.verizon.com/resources/reports/2021-data-breach-investigations-report.pdf.

[13] Wenjun Xiong e Robert Lagerström. "Threat modeling – A systematic literature review". In: *Computers & Security* 84 (2019), pp. 53–69. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.03.010. URL: https://www.sciencedirect.com/science/article/pii/S0167404818307478.

[14] Olayemi Olawumi et al. "Security Issues in Smart Homes and Mobile Health System: Threat Analysis, Possible Countermeasures and Lessons Learned". In: *International Journal on Information Technologies and Security* 9 (mar. 2017), pp. 31–52.

[15] Darren Seifert e Hassan Reza. "A Security Analysis of Cyber-Physical Systems Architecture for Healthcare". In: *Computers* 5.4 (2016). ISSN: 2073-431X. DOI: 10.3390/computers5040027. URL: https://www.mdpi.com/2073-431X/5/4/27.

[16] Alvaro A. Cardenas, Tanya Roosta e Shankar Sastry. "Rethinking security properties, threat models, and the design space in sensor networks: A case study in SCADA systems". In: *Ad Hoc Networks* 7.8 (2009). Privacy and Security in Wireless Sensor and Ad Hoc Networks, pp. 1434–1447. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2009.04.012. URL: https://www.sciencedirect.com/science/article/pii/S1570870509000468.

[17] Mike Burmester, Emmanouil Magkos e Vassilis Chrissikopoulos. "Modeling security in cyber–physical systems". In: *International Journal of Critical Infrastructure Protection* 5.3 (2012), pp. 118–126. ISSN: 1874-5482. DOI: https://doi.org/10.1016/j.ijcip.2012.08.002. URL: https://www.sciencedirect.com/science/article/pii/S1874548212000443.

[18] Florian Kammüller e Christian W. Probst. "Modeling and Verification of Insider Threats Using Logical Analysis". In: *IEEE Systems Journal* 11.2 (2017), pp. 534–545. DOI: 10.1109/JSYST.2015.2453215.

[19] Gabriele Lenzini, Sjouke Mauw e Samir Ouchani. "Security analysis of socio-technical physical systems". In: *Computers & Electrical Engineering* 47 (2015), pp. 258–274. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2015.02.019. URL: https://www.sciencedirect.com/science/article/pii/S0045790615000671.

[20] Wenjun Xiong et al. "Cyber security threat modeling based on the MITRE Enterprise ATT&CK Matrix". In: *Software and Systems Modeling* (giu. 2021). DOI: 10.1007/s10270-021-00898-7.

[21] Xinming Ou, Sudhakar Govindavajhala e A. Appel. "MulVAL: A Logic-based Network Security Analyzer". In: *USENIX Security Symposium*. 2005.

[22] Nataliya Shevchenko et al. *THREAT MODELING: A SUMMARY OF AVAILABLE METHODS*. Rapp. tecn. Carnegie Mellon University, Software Engineering Institure, lug. 2018.

[23] Harjinder Singh Lallie, Kurt Debattista e Jay Bal. "A review of attack graph and attack tree visual syntax in cyber security". In: *Computer Science Review* 35 (2020), p. 100219. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2019.100219. URL: https://www.sciencedirect.com/science/article/pii/S1574013719300772.

[24] Tzonelih Hwang et al. "Use of Attack Graphs in Security Systems". In: *Journal of Computer Networks and Communications* (ott. 2014). DOI: 10.1155/2014/818957. URL: https://doi.org/10.1155/2014/818957.

[25] Kai Xu et al. "A vulnerability scanning scheme based on attack graph for smart grid industrial control system". In: *IOP Conference Series: Earth and Environmental Science* 645 (gen. 2021), p. 012060. DOI: 10.1088/1755-1315/645/1/012060. URL: https://doi.org/10.1088/1755-1315/645/1/012060.

[26] Engla Rencelj Ling e Mathias Ekstedt. "Generating Threat Models and Attack Graphs based on the IEC 61850 System Configuration description Language". eng. In: *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*. 2021.