Alma Mater Studiorum · Università di Bologna

Scuola di Scienza e Ingegneria · Sede di Bologna
Dipartimento di Informatica - Scienza e Ingegneria
Laurea in Informatica

Sviluppo di un Web-client per Ranflood: Accessibilità, Controllo e Automazione

Relatore:
Chiar.mo Prof.
Saverio Giallorenzo

Presentata da:
Filippo Terzi

Correlatore:
Dott.
Simone Melloni

II Sessione
Anno Accademico 2023/2024

Abstract

L'aumento degli attacchi ransomware ha spinto lo sviluppo di soluzioni innovative per mitigare i danni causati da questi malware. In questo contesto, Ranflood è un software progettato per contrastare gli attacchi ransomware attraverso la tecnica del Data Flooding Against Ransomware (DFaR), che utilizza file esca per rallentare e ostacolare il processo di cifratura del malware. Tuttavia, l'interfaccia a linea di comando di Ranflood (CLI) presenta limitazioni in termini di accessibilità e usabilità, soprattutto per utenti non tecnici.

Questa tesi descrive il processo di progettazione e sviluppo di un'interfaccia web per Ranflood, mirata a rendere il software più accessibile e facile da usare. La web app è stata sviluppata per garantire una gestione centralizzata e intuitiva delle operazioni di difesa contro i ransomware, semplificando l'interazione rispetto alla CLI e consentendo la gestione di più istanze di Ranflood su diversi server.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi anni, il crescente aumento di attacchi ransomware ha rappresentato una delle principali minacce alla sicurezza informatica a livello globale. Questi attacchi, in grado di cifrare i dati delle vittime e richiedere un riscatto per il loro ripristino, causano perdite economiche significative e danneggiano la reputazione delle organizzazioni. In questo contesto, lo sviluppo di strumenti efficaci per contrastare tali minacce è diventato una priorità per la comunità di sicurezza informatica.

Il progetto presentato in questa tesi si concentra sulla realizzazione di un'interfaccia web per Ranflood, un software avanzato di mitigazione contro gli attacchi ransomware. Ranflood utilizza una tecnica innovativa, chiamata Data Flooding Against Ransomware (DFaR), che consiste nell'inondare aree del sistema con file esca per rallentare e ostacolare l'azione del malware. Tuttavia, l'interfaccia di Ranflood, originariamente sviluppata come linea di comando (CLI), presentava alcune limitazioni in termini di accessibilità e usabilità, rendendo complesso il suo utilizzo per utenti non esperti.

L'obiettivo di questo progetto è quello di rendere Ranflood accessibile a un pubblico più ampio, migliorandone l'interazione tramite la progettazione di una web app. La scelta di sviluppare una web app si basa su diversi fattori. In primo luogo, un'interfaccia web rende il software disponibile su diverse piattaforme senza la necessità di

installare software aggiuntivo, facilitando così l'accesso da vari dispositivi. In secondo luogo, l'interfaccia grafica di una web app semplifica notevolmente l'interazione rispetto alla CLI, migliorando l'esperienza utente e riducendo la possibilità di errori operativi. Infine, una web app consente una gestione centralizzata e intuitiva di più istanze di Ranflood distribuite su diversi sistemi, semplificando il monitoraggio e il controllo dei daemon in esecuzione.

Questa tesi esamina le scelte progettuali e implementative che hanno portato alla realizzazione della web app per Ranflood, illustrando le tecnologie adottate e le soluzioni implementate per migliorare la comunicazione tra client e server, garantire efficienza operativa e ottimizzare l'esperienza utente.

Il progetto è open-source e pubblicato su GitHub[1].

Capitolo 2

Preliminari

In questo capitolo vengono introdotti i concetti fondamentali necessari per comprendere le scelte che hanno dettato lo sviluppo dell'interfaccia web per Ranflood, un software progettato per contrastare i ransomware. Si analizza innanzitutto il ransomware, un tipo di malware che cripta i dati delle vittime per estorcere un riscatto, e si descrivono le tecniche di mitigazione di Ranflood, come il Data Flooding Against Ransomware (DFaR), che utilizza file esca per rallentare gli attacchi. Vengono inoltre illustrati i protocolli di comunicazione HTTP e WebSocket, essenziali per garantire un'interazione efficace tra l'utente e il sistema Ranflood tramite l'interfaccia web.

2.1 Ransomware

Il ransomware[2] è una categoria di malware[3] progettata per estorcere denaro o altre risorse alle vittime, impedendo loro di accedere ai propri dati o esfiltrandoli minacciando di renderli pubblici, fino al pagamento di un riscatto. Questo tipo di attacco ha visto un aumento significativo negli ultimi anni[4]. Le tecniche di attacco ransomware si sono evolute, con i criminali informatici che spesso combinano tecniche di social engineering per ingannare gli utenti e vulnerabilità software per diffondersi in modo più efficace. Le conseguenze di un attacco possono essere devastanti, non solo

per la perdita di dati, ma anche per i danni finanziari e reputazionali. Le strategie di difesa contro i ransomware si concentrano principalmente su tre aree: prevenzione, rilevamento e mitigazione. La prevenzione mira a impedire che l'attacco avvenga, adottando misure come l'aggiornamento costante dei software, l'implementazione di politiche di sicurezza informatica solide e l'educazione degli utenti a riconoscere potenziali minacce, come e-mail di phishing. Il rilevamento si basa sull'uso di strumenti che monitorano in tempo reale il comportamento anomalo all'interno dei sistemi, come l'accesso sospetto ai file o l'attività di cifratura rapida, permettendo di identificare l'attacco il prima possibile. La mitigazione, infine, entra in gioco quando l'attacco è già in corso, cercando di ridurre al minimo i danni. Questo può avvenire attraverso backup regolari dei dati, l'uso di software che limitano l'impatto della cifratura (come Ranflood) o il blocco immediato delle reti compromesse, per limitare la diffusione del malware e facilitare il recupero dei dati.

2.2 Ranflood

Figura 2.1: Ranflood Logo

Ranflood[5] è uno strumento di mitigazione progettato per contrastare gli attacchi ransomware, in particolare il ransomware di tipo "crypto", che cifra i file degli utenti e richiede un riscatto per il loro ripristino. Questo software implementa il concetto di **Data Flooding Against Ransomware (DFaR)**[6], un'evoluzione delle tecniche basate su honeypot. Ranflood inonda determinate aree del disco, come le cartelle

dell'utente o i luoghi di attacco, con file esca appositamente generati per distrarre il ransomware, che finisce per cifrare questi file fittizi invece di quelli reali. In questo modo, Ranflood rallenta il processo di cifratura, contendendo l'accesso alle risorse come CPU, memoria e disco, dando così al team di difesa più tempo per intervenire. Ranflood sviluppa tre caratteristiche principali per mitigare gli attacchi ransomware.

Contesa delle Risorse: Ranflood contrasta il ransomware competendo per l'accesso alle risorse del sistema, come I/O, utilizzo della memoria e CPU. Il software esegue un'inondazione massiccia di dati tramite file esca, creando un carico competitivo per l'accesso a queste risorse. Ciò rallenta significativamente il processo di cifratura del ransomware, fornendo più tempo ai team di difesa per intervenire.

Difesa da bersagli mobili: Generando e distribuendo file esca, Ranflood implementa una difesa dinamica con obiettivi mobili. Questo approccio confonde il ransomware, rendendo difficile per il malware distinguere i file reali da quelli fittizi. Di conseguenza, l'efficacia dell'attacco viene compromessa, riducendo le probabilità che i dati dell'utente vengano cifrati.

Honeypot Dinamico: La superficie di difesa di Ranflood non è statica. I luoghi in cui vengono collocati i file esca vengono determinati in tempo reale durante l'attacco. Questo concetto è basato sull'idea dell'honeypot, una trappola digitale progettata per attirare gli attaccanti e far loro credere di aver trovato risorse preziose, quando in realtà stanno interagendo con file fittizi. L'adattamento dinamico di Ranflood consente al software di rispondere in modo flessibile ai diversi profili di attacco, migliorando l'efficacia della protezione.

Ranflood offre inoltre tre strategie di flooding per ottimizzare la difesa:

- Random: Questa strategia genera file esca casuali in posizioni specifiche del disco senza richiedere configurazioni preliminari. I file generati sono unici e utilizzano estensioni mirate dai ransomware (come .pdf e .jpg). Questa strategia è semplice da implementare e confonde il ransomware, ma non garantisce il recupero dei file reali.
- On-The-Fly: Invece di generare file casuali, questa strategia crea copie dei file

reali dell'utente in posizioni specifiche, aumentando le possibilità di preservare i file originali. Prima di replicare i file, Ranflood evita quelli già cifrati, risparmiando risorse.

- Shadow: Questa strategia conserva copie archiviate dei file reali dell'utente, riducendo lo spazio richiesto per le copie grazie alla compressione (ad esempio, usando formati come tar.gz). Le copie archiviate possono essere conservate su dischi secondari, NAS o in cloud, rendendo più difficile per il ransomware cancellare tutti i backup locali.

Con queste strategie combinate, Ranflood massimizza l'efficacia nel rallentare e mitigare gli attacchi ransomware, offrendo flessibilità e una maggiore probabilità di recupero dei file cifrati.

Architettura di Ranflood: Ranflood segue un'architettura di tipo client-daemon. Il daemon è un componente sempre attivo che esegue operazioni come la creazione di snapshot dei file e il flooding. È progettato per gestire più operazioni in parallelo grazie alla sua architettura modulare, che divide le operazioni in task, rendendo il sistema resiliente a eventuali errori durante l'esecuzione. Il client, invece, interagisce con il daemon tramite un'interfaccia leggera e asincrona, permettendo agli utenti o ad altri software di configurare e avviare operazioni di flooding e snapshot. Il task manager implementa il pattern Proactor, che permette di gestire efficacemente l'elevato parallelismo e minimizzare i tempi di latenza.

Comandi principali: Ranflood offre una serie di comandi per eseguire operazioni di flooding e snapshot. Tra i comandi principali ci sono:

- `ranflood snapshot/flood list`: Elenca tutti gli snapshot o le operazioni di flooding in corso.
- `ranflood snapshot take <method> <targetFolder>`: Avvia un'operazione di snapshot (On-the-Fly, Shadow) su una specifica cartella.
- `ranflood snapshot remove <method> <targetFolder>`: Rimuove, se presente, lo snapshot specificato.

- `ranflood flood start <method> <targetFolder>`: Avvia un'operazione di flooding (Random, On-the-Fly, Shadow) su una specifica cartella.
- `ranflood flood stop <method> <ids>`: Interrompe una lista di operazioni di flooding identificate dai loro ID.

2.3 Protocollo HTTP

L'HyperText Transfer Protocol (HTTP) è il protocollo di base per lo scambio di dati su Internet. Funziona secondo un modello client-server, in cui il client effettua richieste e il server risponde fornendo le informazioni richieste. HTTP è di tipo stateless, il che significa che ogni interazione è indipendente dalle precedenti, senza mantenere memoria dello stato tra le varie richieste. Tra i metodi più utilizzati all'interno del protocollo ci sono GET, per ottenere risorse, e POST, per inviare dati al server. HTTP opera su TCP (Transmission Control Protocol), garantendo una trasmissione affidabile, e le versioni più recenti, come HTTP/2, hanno migliorato aspetti come l'efficienza, il parallelismo e la compressione dei dati.

2.4 WebSocket

I WebSocket[7] sono una tecnologia che consente comunicazioni bidirezionali e persistenti tra client e server attraverso una singola connessione TCP. A differenza dei tradizionali protocolli come HTTP, che seguono un modello di richiesta-risposta, i WebSocket permettono uno scambio continuo di dati in tempo reale senza dover riaprire nuove connessioni per ogni messaggio. Dopo una fase di handshake iniziale, che avviene tramite HTTP, la connessione viene "upgradata" a WebSocket, rimanendo aperta fino a quando una delle parti decide di chiuderla. Questo riduce la latenza e il sovraccarico di rete, rendendo i WebSocket ideali per applicazioni come chat, giochi online o aggiornamenti di dati in tempo reale.

Capitolo 3

Requisiti

L'interfaccia web di Ranflood è stata progettata con due obiettivi principali: migliorare l'esperienza utente delle funzionalità già presenti nella Command Line Interface (CLI) e introdurre nuove feature per estendere le capacità del sistema. Questi requisiti sono stati definiti con l'intento di semplificare l'interazione con Ranflood, rendendolo accessibile anche a chi non possiede competenze tecniche avanzate e, al contempo, migliorare l'efficacia delle operazioni di difesa contro i ransomware. Inoltre, è stato deciso di costruire un'architettura daemon-webclient per garantire consistenza rispetto all'architettura preesistente, mantenendo l'approccio modulare già presente nel sistema. Di seguito, verranno esaminati in dettaglio i principali requisiti che hanno guidato lo sviluppo dell'interfaccia.

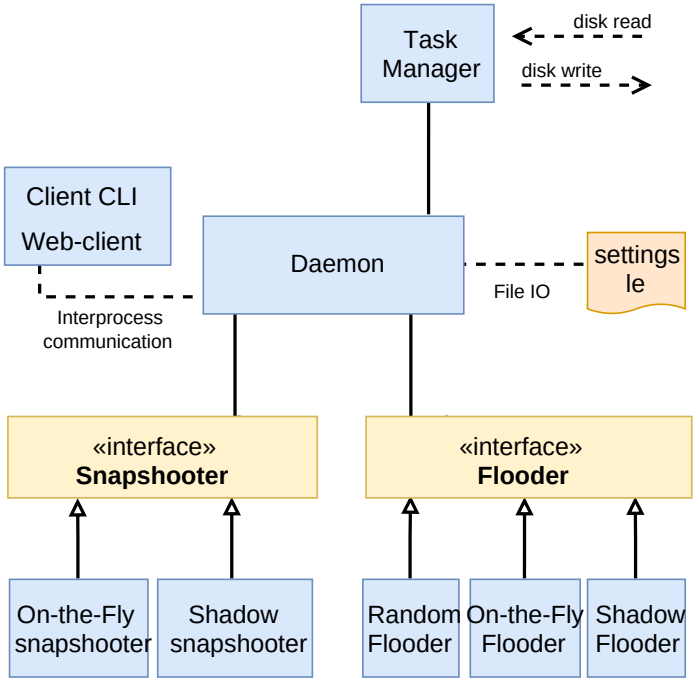

Figura 3.1: Ranflood architecture

3.1 User Experience

Il miglioramento della User Experience (UX) è stato uno degli aspetti principali nello sviluppo dell'interfaccia web di Ranflood. A differenza della CLI, che richiede competenze tecniche avanzate e familiarità con i comandi testuali, l'interfaccia web è stata progettata per essere semplice e accessibile anche per chi non ha molta esperienza. L'obiettivo è rendere l'uso di Ranflood molto più intuitivo, offrendo un'esperienza visiva e facile da usare.

Grazie all'interfaccia grafica, gli utenti possono eseguire operazioni complesse con pochi clic, senza dover memorizzare o digitare comandi. Inoltre, il design visivo fornisce un feedback immediato e chiaro, come notifiche e messaggi che aggiornano l'utente sui progressi o eventuali errori. Questo approccio riduce la possibilità di errori e rende il sistema più accessibile anche ai non tecnici, migliorando l'efficacia e l'efficienza nelle operazioni di difesa contro i ransomware.

3.2 Gestione di più daemon Ranflood

Un altro requisito fondamentale introdotto con l'interfaccia web riguarda la gestione di più daemon Ranflood in modo centralizzato e semplificato. Nella CLI, l'utente deve collegarsi manualmente a ciascun daemon su ogni macchina, inserendo manualmente l'indirizzo e lanciando comandi per eseguire operazioni, il che rende complesso e inefficiente gestire più istanze di Ranflood su diversi server.

L'interfaccia web, invece, fornisce una visione d'insieme chiara e immediata dello stato di tutti i daemon attivi, rendendo possibile il monitoraggio e la gestione centralizzata senza dover inserire manualmente i dettagli di connessione per ogni singola macchina. Questo migliora significativamente l'efficienza e l'esperienza utente, poiché l'utente può controllare e gestire diversi daemon da un'unica interfaccia, eseguendo operazioni di flooding e snapshot su macchine diverse con facilità. In questo modo, l'interazione con Ranflood risulta notevolmente più rapida e organizzata, semplificando la gestione di ambienti distribuiti.

3.3 Strategie

Una delle feature richieste dell'app web di Ranflood é la possibilità di selezionare ed eseguire strategie predefinite, progettate per automatizzare e semplificare le operazioni di contrasto ai ransomware. Queste strategie consistono in un insieme di azioni configurate in precedenza, per esempio lo snapshot o il flooding di cartelle frequentemente prese di mira dai ransomware, come Desktop, Documenti, Immagini, o Download, e che l'utente può eseguire in modo semplice e intuitivo tramite l'interfaccia web. L'obiettivo di questa funzionalità è ridurre la complessità operativa e rendere Ranflood accessibile anche agli utenti meno esperti.

A differenza della CLI, che richiede una configurazione manuale delle operazioni, l'interfaccia web permette di selezionare una strategia dalla lista di opzioni preimpostate e avviarla con un solo clic. Ad esempio, una strategia può includere una fase di snapshot per proteggere i dati sensibili seguita da un'operazione di flooding per rallentare l'attacco del ransomware. Una volta selezionata la strategia, l'interfaccia

si occuperà di eseguire l'intero flusso, riducendo al minimo l'interazione necessaria da parte dell'utente.

Questa funzionalità è cruciale per automatizzare processi complessi e migliorare ulteriormente l'accessibilità dello strumento, in particolare per chi non ha una conoscenza tecnica approfondita. Attraverso le strategie, l'utente può proteggere il sistema senza doversi occupare dei dettagli operativi, aumentando l'efficienza complessiva del processo di difesa.

3.4 Errori

Nella CLI attuale, uno dei limiti più significativi è la gestione degli errori. L'utente non riceve un feedback diretto sull'esito dei comandi inviati al daemon Ranflood, e i log degli errori vengono mostrati solo nella console del daemon stesso, complicando la diagnosi e la risoluzione dei problemi.

L'interfaccia web è stata progettata per affrontare questo problema fornendo un sistema di notifica immediato per l'utente. Al termine di ogni operazione, viene visualizzato un messaggio che indica l'esito del comando: verde in caso di successo, rosso in caso di errore. In presenza di problemi, viene mostrato il messaggio di errore ricevuto dal daemon. Questo approccio migliora significativamente la trasparenza e l'usabilità del sistema, consentendo all'utente di capire rapidamente lo stato delle operazioni e di intervenire prontamente in caso di necessità.

Capitolo 4

Implementazione del Web-client

Vediamo ora come sono stati implementati i vari requisiti illustrati nel capitolo precedente, l'intero codice è disponibile su Github [1].

4.1 Tecnologie

La scelta delle tecnologie per lo sviluppo dell'interfaccia web di Ranflood è stata dettata dall'esigenza di creare un'applicazione moderna, scalabile e facile da mantenere. In particolare, sono stati utilizzati React, TypeScript e Material UI per rispondere a queste necessità in maniera efficiente.

React è stato scelto come framework principale per la sua capacità di gestire in modo efficace interfacce dinamiche e reattive e per la sua diffusione nella community degli sviluppatori [8].

TypeScript è stato scelto per aggiungere un ulteriore livello di sicurezza e robustezza al codice. Grazie al suo sistema di tipizzazione statica, TypeScript permette di rilevare errori già in fase di sviluppo, riducendo i bug e migliorando la qualità del codice.

Infine, Material UI è stato selezionato per il design dell'interfaccia utente per diverse ragioni. Oltre a offrire un set di componenti già pronti e altamente personalizzabili,

Material UI si distingue anche per la sua larga diffusione e adozione all'interno della community di sviluppo frontend[9]. Questa popolarità garantisce un supporto continuo, una vasta documentazione e una costante evoluzione del framework, fattori che hanno contribuito alla scelta di questa libreria. Material UI accelera lo sviluppo dell'interfaccia grafica, permettendo di creare rapidamente layout professionali e reattivi che migliorano l'usabilità e l'accessibilità dell'applicazione, senza dover sviluppare da zero componenti complessi.

4.2 Protocollo di comunicazione, HTTP vs WebSocket

Sono state valute diverse scelte implementative per il meccanismo di comunicazione tra client e server, confrontando in particolare l'uso delle richieste HTTP e dei WebSocket. Dopo aver implementato entrambe le soluzioni, ci siamo resi conto che l'approccio basato su richieste HTTP risulta significativamente meno efficiente, soprattutto per la gestione di comandi asincroni come "flood start". Con il protocollo HTTP, per monitorare l'esito di questi comandi, si sarebbe costretti a utilizzare il polling, ovvero inviare ripetute richieste al server finché il comando non è completato. Questo metodo non solo genera un numero elevato di richieste, ma diventa inefficiente quando i comandi richiedono molto tempo per essere eseguiti, creando traffico inutile.

Questo problema diventa ancora più pronunciato quando eseguiamo strategie che comportano l'invio di azioni multiple in parallelo. Se utilizzassimo il polling per monitorare tutti questi comandi, il numero di richieste al server aumenterebbe esponenzialmente, sovraccaricando il sistema. Inoltre, nel caso in cui ci siano dipendenze tra i comandi, ossia quando alcuni comandi devono aspettare il completamento di altri, il tempo complessivo di esecuzione della strategia aumenterebbe.

L'utilizzo dei WebSocket ha permesso di risolvere questi problemi, stabilendo un canale di comunicazione bidirezionale continuo. Questo permette al server di inviarci notifiche sullo stato dei comandi appena sono disponibili, ed elimina la necessità

di inviare richieste periodiche, migliorando l'efficienza complessiva del sistema. Di conseguenza, abbiamo anche incrementato la velocità nella gestione delle strategie, poiché il server può comunicare in tempo reale il completamento dei comandi, consentendo l'esecuzione rapida di quelli successivi.

4.3 Gestione di più daemon Ranflood

Per semplificare e centralizzare la gestione di più daemon Ranflood, è stato implementato un sistema che permette all'utente di monitorare e controllare diverse macchine in maniera efficiente e intuitiva. Tutte le macchine vengono mostrate nel menù laterale e sono organizzate in gruppi, Figura 4.1.

La lista dei gruppi e delle macchine è salvata nel localStorage del browser, garantendo così persistenza delle informazioni anche in caso di aggiornamento o chiusura della pagina web. Il localStorage è una funzionalità del browser che consente di memorizzare dati localmente sul dispositivo dell'utente sotto forma di coppie chiave-valore, con una capacità di conservazione permanente finché non vengono esplicitamente rimossi. Questo significa che, all'apertura successiva del client, l'utente ritroverà esattamente la stessa configurazione di gruppi e macchine. Inoltre, per facilitare il trasferimento di configurazioni su diversi browser o client, è stata implementata la possibilità di importazione ed esportazione della lista in formato JSON, offrendo una soluzione semplice per la migrazione o la condivisione delle impostazioni tra dispositivi.

All'avvio del client, viene automaticamente eseguito un tentativo di connessione a tutte le macchine presenti nella lista. Questo processo verifica lo stato di ciascun daemon Ranflood e lo visualizza nell'interfaccia attraverso un sistema di "semafori" posto accanto al nome di ciascuna macchina. Il semaforo offre un'indicazione visiva immediata dello stato di ogni macchina:

- Rosso indica che la macchina è offline o irraggiungibile.
- Verde indica che la macchina è online e pronta per l'esecuzione delle operazioni.

- Giallo segnala che è attualmente in corso un'operazione di flooding sulla macchina.

Questo sistema visivo consente agli utenti di capire rapidamente lo stato delle macchine e di agire di conseguenza. La combinazione di un menu laterale organizzato, la persistenza dei dati, e il feedback visivo immediato rende la gestione di più daemon Ranflood semplice e intuitiva, riducendo notevolmente la complessità di monitorare e gestire diversi server in contemporanea.

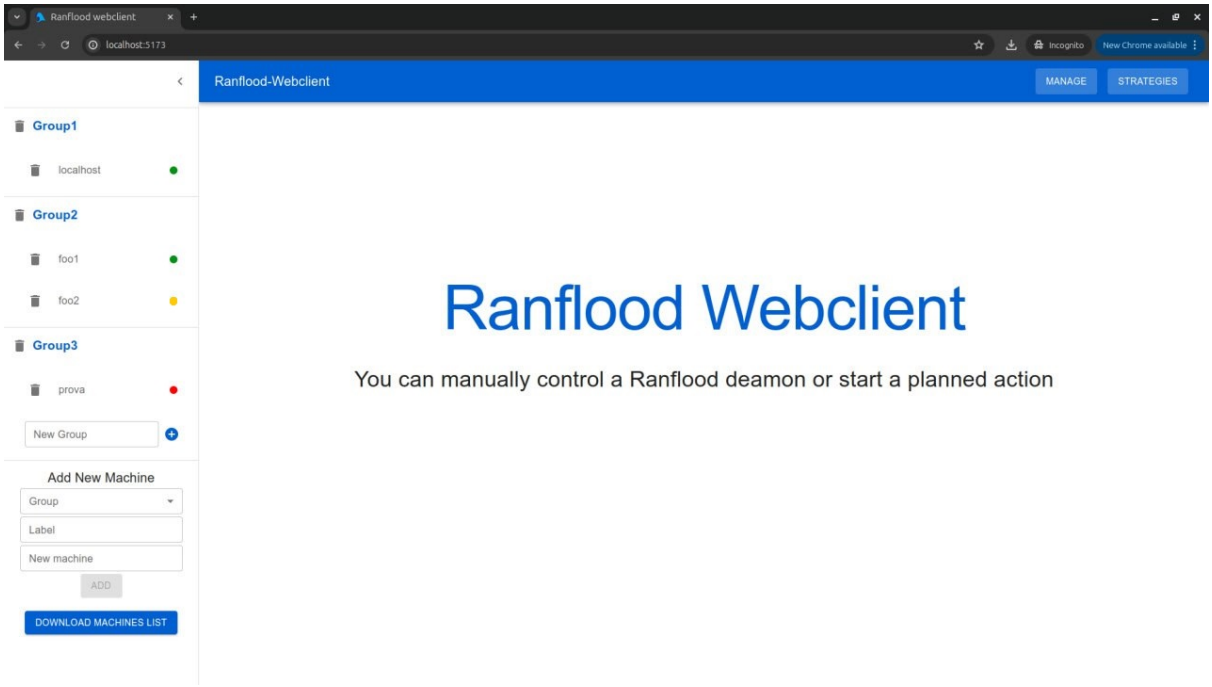

Figura 4.1: Homepage

4.4 Gestione manuale dei singoli deamon Ranflood

Per la gestione manuale dei singoli daemon, è stata creata una pagina dedicata chiamata Manage, Figura 4.2, che permette di eseguire tutti i comandi disponibili nella CLI in modo più rapido ed efficiente. In aggiunta, la pagina Manage tiene traccia di tutti i comandi eseguiti attraverso un log visibile, che registra ogni operazione lanciata dall'interfaccia. Il log è costantemente aggiornato, includendo anche l'esito dei

comandi asincroni, e fornisce un feedback in tempo reale all'utente. Questo sistema di log migliora la trasparenza e il controllo sulle operazioni eseguite, garantendo una gestione più efficiente e affidabile dei singoli daemon.

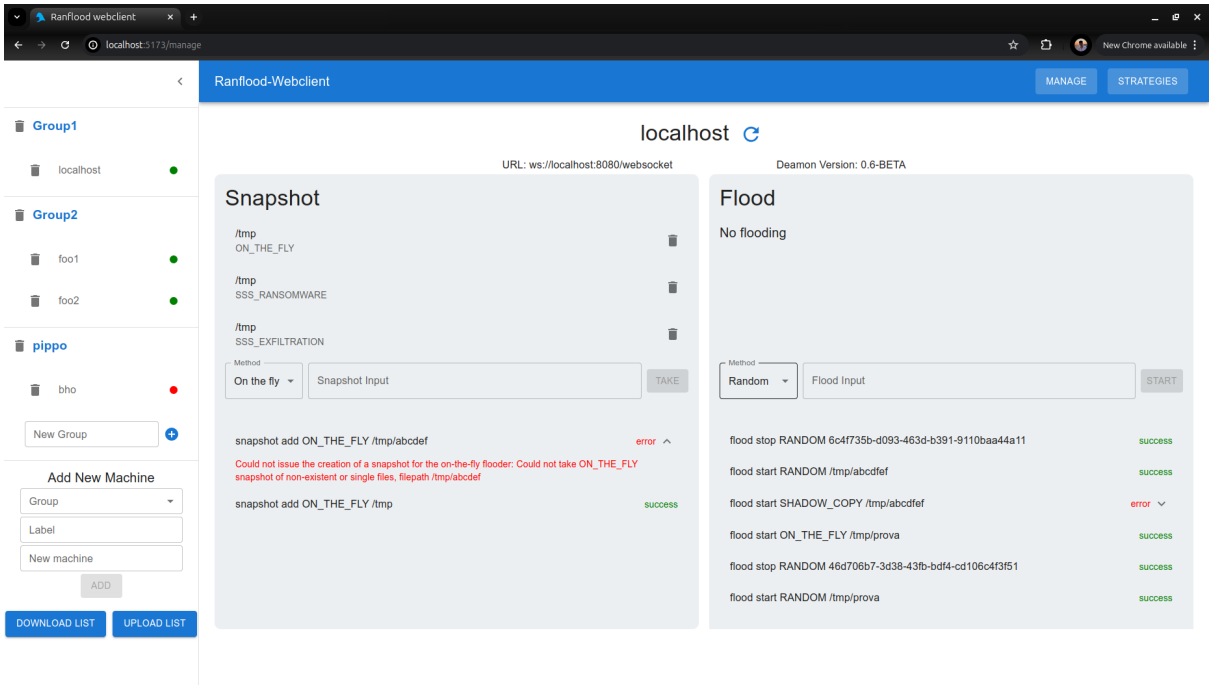

Figura 4.2: Manage page

4.5 Strategie

La pagina Strategies è stata progettata per offrire una gestione intuitiva e centralizzata delle strategie predefinite di Ranflood. All'interno della pagina, l'utente può visualizzare una lista di tutte le strategie disponibili, con i dettagli delle azioni che ciascuna strategia contiene. Ogni strategia può essere scaricata o caricata in formato JSON, offrendo la possibilità di esportare configurazioni personalizzate o importare nuove strategie da altri client. Le strategie sono salvate nel localStorage del browser, garantendo la persistenza delle configurazioni anche tra sessioni differenti.

Durante l'esecuzione di una strategia, l'utente ha la possibilità di fermarla in qualsiasi momento tramite l'interfaccia, consentendo un controllo più flessibile sulle

operazioni in corso. Questo migliora l'efficienza operativa, permettendo all'utente di interrompere rapidamente un'azione se necessario, senza dover attendere il completamento automatico.

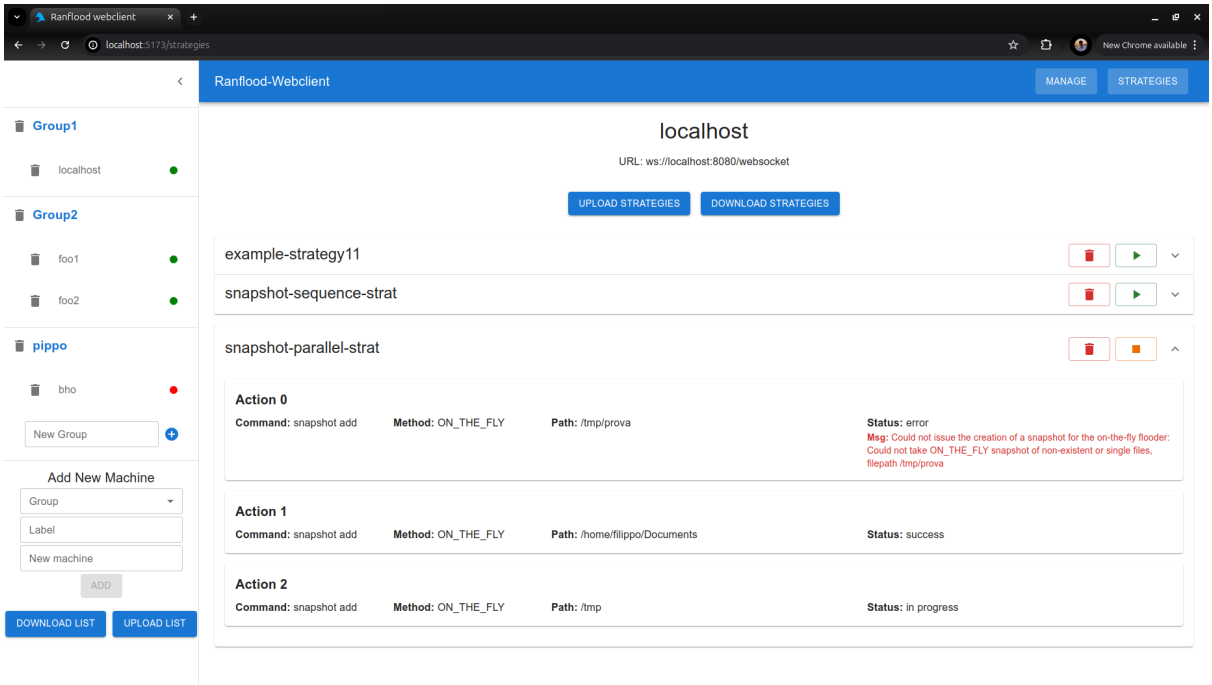

Figura 4.3: Strategies page

4.5.1 Struttura del file JSON

Le strategie vengono definite utilizzando un file in formato JSON, vedi Appendice A, che descrive l'insieme di comandi da eseguire. La struttura del file è progettata per essere semplice e flessibile, permettendo di specificare i comandi e le loro proprietà. Un esempio di file JSON di strategia è il seguente: Descrizione della struttura:

- name: Definisce il nome della strategia.
- commands: Un array che contiene la lista dei comandi da eseguire.

Ogni comando è rappresentato come un oggetto con i seguenti attributi:

- id: Identificatore univoco del comando all'interno della strategia.

- command: Il tipo di operazione, ad esempio “snapshot” o “flood”.
- subcommand: Definisce l’azione specifica del comando, come “add” per aggiungere uno snapshot o “start” per avviare un flooding.
- method: Specifica la strategia di flooding, “RANDOM”, “ON_THE_FLY” o “SHADOW”.
- path: Indica il percorso della directory su cui viene eseguito il comando.
- duration: (Solo per i comandi di flooding) Definisce la durata dell’operazione di flooding in secondi, sostituisce il comando flood stop.
- dependencies: (Opzionale) Un array di ID che rappresenta i comandi da completare prima dell’esecuzione di questo comando.

4.5.2 Strategies engine

L’engine è il componente che si occupa di gestire l’esecuzione delle strategie definite attraverso file JSON. Ogni strategia è costituita da una sequenza di comandi, che vengono eseguiti in parallelo per ottimizzare l’efficienza, a meno che non siano presenti vincoli specifici definiti nell’attributo dependencies. Quando un comando ha delle dipendenze, l’engine attende il completamento dei comandi elencati come dipendenze prima di procedere con la sua esecuzione. Questo meccanismo permette di definire flussi di esecuzione sequenziali, garantendo che comandi che dipendono da altri vengano eseguiti nell’ordine corretto, migliorando l’affidabilità dei processi.

Algoritmo di esecuzione delle strategie: L’algoritmo di esecuzione si basa su una combinazione di gestione delle dipendenze e controllo dello stato di esecuzione dei comandi. L’engine mantiene due grafi di dipendenze: un grafo diretto che mappa le dipendenze di ogni comando, e un grafo inverso che tiene traccia dei comandi che dipendono da ciascun comando. Questo consente all’engine di controllare quando un comando può essere eseguito, ovvero solo quando tutte le sue dipendenze sono state completate con successo.

Ogni comando ha tre possibili stati:

- Pending: in attesa di essere eseguito, poiché le sue dipendenze non sono ancora state soddisfatte.
- In progress: attualmente in esecuzione.
- Completed: eseguito con successo o fallito.

L'algoritmo inizia eseguendo tutti i comandi che non hanno dipendenze, e successivamente attiva i comandi dipendenti man mano che le loro dipendenze vengono soddisfatte. Questo avviene in modo dinamico, attraverso l'aggiornamento del grafo inverso: ogni volta che un comando viene completato, i comandi che dipendono da esso vengono controllati per verificare se possono essere eseguiti.

Algoritmo di validazione delle strategie: Quando le strategie vengono importate, l'engine effettua una fase di validazione per garantire che non ci siano errori nella definizione dei comandi. L'algoritmo di validazione verifica due aspetti principali:

- Unicità degli ID: Ogni comando all'interno di una strategia deve avere un ID univoco. L'algoritmo utilizza un set per raccogliere tutti gli ID e rilevare eventuali duplicati, poiché ID duplicati potrebbero compromettere il corretto funzionamento dell'engine e causare conflitti durante l'esecuzione.
- Assenza di cicli nelle dipendenze: L'algoritmo costruisce un grafo aciclico diretto (DAG)[10] basato sulle dipendenze di ciascun comando. Utilizza una ricerca in profondità (DFS, Depth-First Search) per rilevare la presenza di cicli. Se vengono rilevati cicli, l'engine blocca l'importazione della strategia, poiché i cicli nelle dipendenze causerebbero un deadlock, impedendo a uno o più comandi di essere eseguiti. Vedi Appendice C per un esempio di strategia errata.

Questo meccanismo di validazione è cruciale per evitare errori prima dell'esecuzione delle strategie. Nel caso in cui la validazione fallisca (ad esempio, per ID duplicati o dipendenze cicliche), l'utente viene informato con un messaggio di errore, Figura 4.4, e la strategia non viene importata nel sistema. Questo assicura che solo strategie corrette vengano eseguite, evitando problemi durante il runtime.

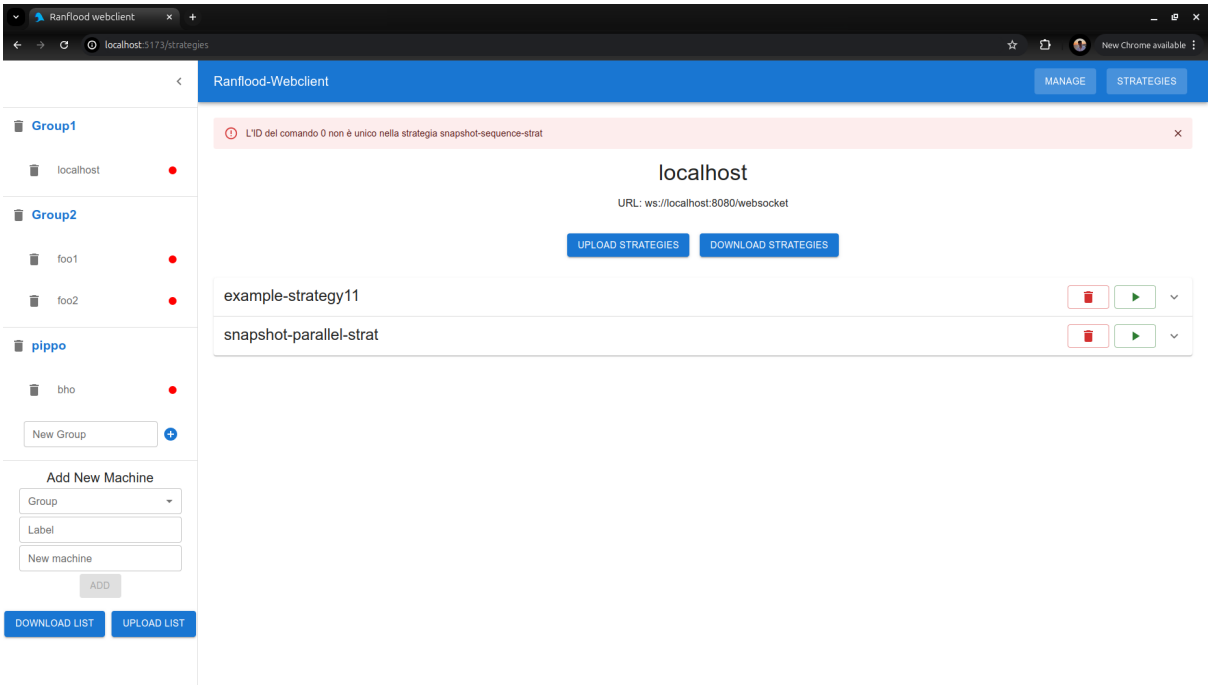

Figura 4.4: Strategies validation example error

4.6 Notifiche e log

Nell'interfaccia web di Ranflood, la gestione del log dei comandi e l'esito delle operazioni, sia positivo che negativo, è stata progettata per offrire un feedback chiaro e immediato all'utente. Esistono due categorie principali di errori che possono verificarsi durante l'esecuzione dei comandi: errori di connessione o legati al webclient e errori restituiti dal daemon. Entrambi i tipi di errore vengono gestiti in modo distinto per facilitare la comprensione e la risoluzione dei problemi.

Errori di connessione o legati al webclient: Questi errori riguardano problemi che si verificano quando il client non riesce a comunicare con il daemon, ad esempio a causa di una macchina offline o di una connessione instabile. Quando si verifica un errore di questo tipo, un messaggio di errore viene visualizzato chiaramente in cima alla pagina, sopra la label della macchina corrispondente. Questo posizionamento permette all'utente di capire immediatamente che c'è un problema di connessione

prima di eseguire altri comandi, riducendo confusione e possibili errori ripetuti.

Errori restituiti dal daemon: Questi errori si verificano quando un comando inviato al daemon non viene eseguito correttamente. Per questa tipologia, l'errore viene visualizzato nel log dei comandi, insieme all'esito delle operazioni. Ogni comando inviato al daemon viene accompagnato da un ID univoco generato dal client, che consente di tracciare l'esito del comando in modo preciso. Quando il daemon termina l'esecuzione del comando, invia una risposta, vedi Appendice B, che include l'ID del comando, permettendo al client di associare il risultato al comando specifico e di mostrare il relativo esito all'utente. Se l'operazione fallisce, l'esito negativo viene evidenziato nel log con un messaggio di errore.

Capitolo 5

Implementazione del server

Inizialmente, il daemon accettava esclusivamente connessioni TCP per la gestione dei comandi, limitando le modalità di interazione con i client. Per ampliare le capacità del sistema, sono stati aggiunti un server HTTP e un server WebSocket. Sebbene il server HTTP sia stato implementato per garantire compatibilità con le moderne applicazioni web, durante lo sviluppo del client web è stato scelto di privilegiare l'uso dei WebSocket per i vantaggi descritti nel capitolo 4.2. Questa decisione ha consentito una gestione più efficiente dei comandi asincroni, migliorando l'interazione in tempo reale con il daemon. L'intero codice è disponibile su Github[11].

5.1 Server HTTP e WebSocket

Nel corso dell'implementazione di questi due protocolli di comunicazione, abbiamo deciso di mantenere lo stesso formato JSON per rappresentare i comandi che è utilizzato nelle comunicazioni TCP. Questa scelta è stata presa per garantire la consistenza tra i vari protocolli di comunicazione e semplificare l'integrazione con nuovi client, riducendo il rischio di incongruenze e duplicazioni nel codice.

Il formato JSON, essendo semplice e ampiamente supportato, permette di serializzare e deserializzare facilmente i comandi e i risultati, indipendentemente dal protocollo utilizzato (TCP, HTTP, WebSocket). Questo approccio unificato facilita

la manutenzione e l'espansione del sistema, garantendo che l'implementazione del server sia flessibile e compatibile con un'ampia gamma di client.

Per configurare il server HTTP e WebSocket, è possibile specificare la porta su cui questi servizi ascoltano le richieste nel file di configurazione "setting.ini", che viene passato come parametro all'avvio del daemon. Questa configurazione permette agli amministratori di sistema di adattare facilmente il daemon alle specifiche necessità di rete.

5.2 Gestione degli ID dei comandi

Un'importante modifica apportata durante lo sviluppo è stata l'estensione della gestione degli ID dei comandi. In origine, gli ID venivano utilizzati esclusivamente per i comandi di flooding, per monitorare e gestire l'esecuzione asincrona di tali comandi. Tuttavia, con l'introduzione di funzionalità più avanzate e comandi più complessi, si è reso necessario estendere l'uso degli ID a tutti i comandi.

L'utilizzo degli ID per tutti i comandi consente una gestione più robusta e precisa delle operazioni asincrone. Quando un client invia un comando al daemon, può includere un ID nel payload del messaggio. Se il client non specifica un ID, il daemon ne genera automaticamente uno utilizzando un identificatore univoco globale (UUID). L'ID viene poi associato al comando e inviato al client insieme alla risposta. Questo meccanismo permette al client di associare correttamente ogni risposta al comando che l'ha generata, risolvendo il problema del tracciamento delle risposte in contesti asincroni.

Questo sistema di gestione degli ID è particolarmente vantaggioso nel contesto dei WebSocket, dove i comandi possono essere inviati in rapida successione e le risposte possono arrivare in ordine non sequenziale. Gli ID univoci garantiscono che ogni risposta possa essere correlata al comando originario, indipendentemente dall'ordine di arrivo. Questo migliora notevolmente la robustezza del sistema, soprattutto quando si gestiscono operazioni complesse e asincrone su larga scala.

L'estensione dell'utilizzo degli ID ha reso il sistema più scalabile e versatile, consentendo di monitorare con precisione l'avanzamento e l'esito di ciascun comando in modo affidabile, sia per i comandi sincroni che asincroni.

Capitolo 6

Conclusioni

In questa tesi è stato presentato lo sviluppo di un'interfaccia web per Ranflood, un software progettato per mitigare gli attacchi ransomware utilizzando la tecnica del Data Flooding Against Ransomware (DFaR). L'obiettivo principale era migliorare l'usabilità del sistema, rendendolo accessibile anche a utenti non tecnici e potenziando l'efficienza operativa attraverso una gestione centralizzata e intuitiva.

6.1 Sviluppi futuri

Questo lavoro potrebbe essere ulteriormente approfondito e migliorato su diversi aspetti. Di seguito alcune possibili evoluzioni di questo progetto.

6.1.1 Gestione flotte di deamon

Uno degli sviluppi futuri più promettenti per Ranflood riguarda l'implementazione di una gestione centralizzata delle flotte di daemon. Attualmente, l'interfaccia web permette di inviare comandi a un singolo daemon alla volta, ma in ambienti complessi con più macchine distribuite, sarebbe vantaggioso poter gestire gruppi di daemon in modo collettivo. L'idea alla base di questa evoluzione è quella di consentire all'utente di inviare comandi simultaneamente a interi gruppi di daemon, migliorando così la scalabilità e l'efficienza operativa.

Questa funzionalità potrebbe essere implementata creando gruppi di daemon configurabili attraverso l'interfaccia, in cui ogni gruppo rappresenta un insieme di macchine con specifiche caratteristiche o ruoli operativi. Una volta creati i gruppi, sarebbe possibile eseguire operazioni di flooding o snapshot simultaneamente su tutte le macchine appartenenti a quel gruppo, riducendo il tempo necessario per distribuire le operazioni di difesa e migliorando la coordinazione tra diversi sistemi.

6.1.2 Sicurezza

Attualmente, Ranflood non include alcun sistema di sicurezza come l'autenticazione o la protezione delle comunicazioni, lasciando il software vulnerabile ad accessi non autorizzati. Uno sviluppo futuro importante sarebbe l'introduzione di meccanismi di autenticazione che richiedano agli utenti di fornire credenziali per accedere e gestire il sistema. Questo garantirebbe che solo utenti autorizzati possano interagire con i daemon e inviare comandi.

Inoltre, implementare la sicurezza nelle comunicazioni tra il client e i daemon, assicurando che i dati siano protetti da potenziali attacchi, migliorerebbe la sicurezza complessiva del sistema. Questi miglioramenti contribuirebbero a rendere Ranflood più sicuro e adatto a operare in ambienti critici, prevenendo accessi non autorizzati e garantendo la protezione delle operazioni.

Riferimenti bibliografici

[1] Available from https://github.com/Flooding-against-Ransomware/ranflood-webclient.

[2] Available from https://www.theguardian.com/technology/askjack/2016/jul/28/how-can-i-remove-ransomware-infection.

[3] Rabia Tahir. A study on malware and malware detection techniques. *International Journal of Education and Management Engineering*, 8:20–30, 03 2018. doi: 10.5815/ijeme.2018.02.03.

[4] Fatim Ghulam, Mustafa Irfan, and Farooq Hassan. A study of ransomware attacks on windows platform. *i-manager's Journal on Computer Science*, 9:21, 01 2022. doi: 10.26634/jcom.9.4.18530.

[5] Davide Berardi, Saverio Giallorenzo, Andrea Melis, Simone Melloni, and Marco Prandini. Ranflood: A mitigation tool based on the principles of data flooding against ransomware. *SoftwareX*, 25:101605, 2024. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2023.101605. URL https://www.sciencedirect.com/science/article/pii/S2352711023003011.

[6] Davide Berardi, Saverio Giallorenzo, Andrea Melis, Simone Melloni, Loris Onori, and Marco Prandini. Data flooding against ransomware: Concepts and implementations. *Computers Security*, 131:103295, 2023. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2023.103295. URL https://www.sciencedirect.com/science/article/pii/S0167404823002055.

[7] Available from https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

[8] . Available from https://2023.stateofjs.com/en-US/libraries/.

[9] . Available from https://2023.stateofcss.com/en-US/css-frameworks/.

[10] Jrgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2008. ISBN 1848009976.

[11] Available from https://github.com/Flooding-against-Ransomware/ranflood.

Appendici

Appendice A

Esempio struttura strategie.json

```
[
    {
        "name": "example-strategy",
        "commands": [
          {
            "id": "0",
            "command'': "snapshot",
            "subcommand": "add"',
            "method": "ON_THE_FLY"',
            "path": "/tmp/prova"
          },
          {
            "id": "1",
            "command": "flood",
            "subcommand": "start",
            "method": "ON_THE_FLY"',
            "path": "/tmp/prova",
            "duration": 10,
            "dependencies": ["2"]
```

```
            },
            {
              "id": "2",
              "command": "snapshot",
              "subcommand": "add",
              "method": "ON_THE_FLY",
              "path": "/tmp/prova"
            }
          ]
      },
]
```

Appendice B

Esempio response deamon


```
{
    command: "snapshot"
    data: "Could not issue the creation of a snapshot for the
        on-the-fly flooder: Could not take ON_THE_FLY snapshot of
        non-existent or single files, filepath /nonExistingFolder"
    id:"13c5cf69-3dd5-464c-a837-791884ab7ba8"
    parameters: {method: "ON_THE_FLY", path: "/nonExistingFolder"}
    method: "ON_THE_FLY"
    path: "/nonExistingFolder"
    status: "error"
    subcommand: "add"
    timestamp: "2024-10-18T07:51:17.208884716Z"
}
```

Appendice C

Strategia con dipendenze cicliche

```
{
    "name": "strategiaErrata",
    "commands": [
      {
        "id": "0"',
        "command": "snapshot",
        "subcommand": "add",
        "method": "ON_THE_FLY",
        "path": "/home",
        "dependencies": ["1"]
      },
      {
        "id": "1",
        "command'': "snapshot",
        "subcommand'': "add",
        "method'': "ON_THE_FLY",
        "path": "/tmp",
        "dependencies": ["0"]
      }
    ]
}
```

Ringraziamenti

Per prima cosa, un grande ringraziamento al mio relatore Saverio Giallorenzo e al mio correlatore Simone Melloni, sempre pronti a guidarmi in ogni fase della realizzazione della tesi.

Infine vorrei ringraziare i miei compagni di corso, albe, max e lore, per i diversi progetti di gruppo fatti insieme, i miei amici, casa, cifu, fre, gabri, giozz, jaco, goberts, canna, moro, ale, pirro, zanno, e la mia famiglia per il loro costante supporto emotivo e morale.