ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea in Ingegneria e Scienze Informatiche

ClientShield: Implementazione di un Servizio Windows per la Sicurezza DNS

Tesi di laurea in:
Programmazione ad Oggetti

Relatore
Prof. Viroli Mirko

Candidato
Federico Diotallevi

IV Sessione di Laurea
Anno Accademico 2023 - 2024

Sommario

Lo sviluppo della tecnologia cresce sempre più velocemente e, con essa, anche i rischi che gli utenti corrono semplicemente navigando in rete. Questa tesi si inserisce nell'ambito dell'internet filtering, cioè la possibilità di filtrare il traffico in rete negando l'accesso a siti potenzialmente pericolosi.

L'obiettivo di questo progetto è lo sviluppo di un software che sia facilmente installabile su un dispositivo Windows e filtri tutte le richieste web effettuate. Si vuole offrire la possibilità ad un un amministratore di rete di attivare e disattivare la protezione del dispositivo da remoto, decidendo eventualmente anche quali categorie di siti bloccare.

Per lo sviluppo è stata fondamentale la collaborazione con FlashStart, azienda di riferimento nel mercato italiano per quanto riguarda il filtraggio Domain Name System (DNS).

Il software è stato sviluppato in contesto aziendale come prototipo e sfrutta ampiamente le soluzioni di filtraggio aziendali. Ciò ha facilitato la gestione delle categorie e dell'effettivo filtraggio DNS, permettendo al progetto di focalizzarsi sulla cattura e redirezione del traffico internet del dispositivo verso i server gestiti da FlashStart.

Optional dedication.

Indice

Introduzione

Navigare in rete è senz'altro una delle attività più diffuse al giorno d'oggi in tutto il mondo. La quantità di informazioni a cui si può accedere è pressoché infinita e, soprattutto, a disposizione di chiunque.

La digitalizzazione delle risorse è ormai diventata essenziale per qualsiasi realtà, dal settore pubblico a quello privato. Tuttavia, tale pratica espone inevitabilmente al rischio che documenti, dati e informazioni sensibili diventino obiettivo di attacchi informatici. A conferma di ciò, il recente rapporto [plSI24] evidenzia come gli attacchi informatici gravi a livello globale siano passati, in media, da 4.5 a 9 al giorno in soli cinque anni. Secondo il rapporto, gli incidenti critici sono aumentati dal 47% all'81% del totale e, in Italia, il 58% degli attacchi è costituito da malware, phishing e social engineering.

Con il recente e rapidissimo sviluppo dell' Intelligenza Artificiale (AI) il campo dell'Internet Filtering ha subito un vera e propria rivoluzione. I nuovi modelli di Machine Learning sono capaci di analizzare e identificare con precisione siti malevoli, grazie anche ai giganteschi volumi di dati di cui dispongono. Questi sistemi riescono a riconoscere tecniche di phishing e malware avanzate, anche quando progettate per eludere i metodi di rilevamento tradizionali.

Questa tesi si colloca in questo contesto e si propone l'obiettivo di sviluppare un software prototipale per il filtraggio internet sul computer di installazione. Il lavoro illustrerà dettagliatamente tutte le fasi di analisi, progettazione e sviluppo del progetto.

Data la necessità di interazione con componenti specifici del sistema operativo, il software è stato sviluppato esclusivamente per Windows, con l'obiettivo di creare un servizio di sistema.

Struttura della Tesi

La tesi si articola in 4 capitoli principali, che descrivono le diverse fasi della realizzazione del progetto, in particolare:

- **Capitolo 1 (Contesto)**: in questo capitolo si tratta in maniera approfondita il problema affrontato dalla tesi. Viene analizzato il software attualmente presente in FlashStart e le motivazioni che inducono alla necessità di un nuovo sviluppo.
- **Capitolo 2 (Requisiti)**: in questo capitolo vengono affrontati casi d'uso e requisiti, analizzando il funzionamento e la percezione del progetto dal punto di vista dell'utente.
- **Capitolo 3 (Tecnologie)**: in questo capitolo vengono riportate le diverse tecnologie utilizzate e le scelte che hanno portato al loro impiego nel progetto.
- **Capitolo 4 (Analisi e progettazione)**: in questo capitolo viene descritta la fase di analisi e progettazione del sistema, illustrandone in particolare l'architettura e i pattern utilizzati, evidenziando in che modo il software ne tragga guadagno.

Capitolo 1

Contesto

1.1 Internet Filtering

Il caso di studio oggetto di questa tesi è un software prototipale, che si inserisce nell'ambito dell'Internet Filtering. Per Internet Filtering si intende un sistema di monitorazione, controllo e limitazione dell'accesso alle risorse online, basato su criteri predefiniti.

Una tecnologia di questo tipo risulta particolarmente utile ad aziende, scuole ed enti pubblici. Tramite questa pratica è infatti possibile effettuare controlli sul contenuto delle pagine web, limitando o impedendo l'accesso a siti contenenti malware o, eventualmente, contenuti indesiderati (ad esempio siti di scommesse, pornografia, droghe...). Risulta evidente fin da subito che, per garantire un filtraggio di contenuti preciso ed efficace, sia necessaria un'attenta analisi delle pagine web, evitando di correre il rischio di impedirne immotivatamente l'accesso.

Questa tesi è svolta come progetto in collaborazione con FlashStart, leader italiana per quanto riguarda il filtraggio DNS. Un server DNS è un sistema che permette di tradurre nomi di domini leggibili da un uomo, in indirizzi interpretabili da dei computer per connettersi tra loro. Il filtraggio si basa proprio su questo principio: quando si accede ad un sito web attraverso un browser qualsiasi, il computer interrogherà un server DNS per ottenere l'indirizzo attraverso cui sia possibile accedere al sito. Con un'apposita configurazione, si può indurre il DNS a bloccare la risoluzione di pagine web malevoli o riportanti contenuti non graditi.

FlashStart offre un servizio DNS configurabile, attraverso cui è possibile definire delle categorie di contenuti da bloccare. I server dell'azienda aggiornano continuamente la lista delle categorie bloccate, facendo largo uso di AI per garantire rapido aggiornamento (in risposta alla nascita di nuovi siti) ed alta precisione. Quando un utente, utilizzando un computer protetto dal servizio di FlashStart, tenta di accedere ad un sito appartenente ad una delle categorie bloccate, il DNS risponderà alla richiesta di risoluzione del nome con l'indirizzo di una pagina indicante i motivi del blocco, al posto di quella richiesta.

1.2 Obiettivi del Progetto

Il filtraggio DNS può essere implementato in diverse modalità, di seguito le principali:

- **A livello di rete:** applicato direttamente dall'Internet Service Provider (ISP)[1], blocca l'accesso a domini malevoli prima che le richieste raggiungano la rete locale.
  Esempio: un'azienda utilizza un servizio DNS filtrato fornito dal proprio ISP per impedire l'accesso a siti malevoli su tutta la rete aziendale.

- **A livello di router:** il router funge da punto di controllo, utilizzando un server DNS con filtraggio per impedire l'accesso a contenuti non autorizzati. Tutti i dispositivi connessi alla rete collegata al router beneficiano automaticamente della protezione.
  Esempio: un istituto scolastico configura il proprio router con un DNS appositamente configurato per impedire l'accesso a siti per adulti e social network bloccandoli per gli studenti.

- **A livello di endpoint:** il filtraggio DNS è applicato direttamente sui dispositivi tramite configurazione manuale o software specifici, garantendo protezione anche fuori dalla rete aziendale o domestica.

[1] Un ISP è un fornitore che offre connettività alla rete, consentendo ai suoi utenti di accedere a Internet. Gli ISP possono anche fornire servizi aggiuntivi, come, appunto, sicurezza informatica e filtraggio.

Esempio: un dipendente che lavora da remoto utilizza un client DNS sicuro sul proprio laptop per proteggersi dalle minacce.

FlashStart offre prodotti per tutte le suddette modalità, tuttavia questo progetto prende in esame soltanto la terza tipologia di protezione DNS, ossia il filtraggio a livello di endpoint. Nel periodo tra il 2019 e il 2021 le aziende italiane, così come quelle di tutto il mondo, hanno dovuto affrontare la necessità dell'impiego del lavoro remoto. Se da una parte tale pratica rappresenta un'opportunità per i lavoratori, che ne guadagnano in flessibilità, dall'altra espone l'azienda a nuovi rischi. A dimostrazione di ciò, proprio nel 2020 è stato emanato dall'Agenzia per l'Italia Digitale (AGID) il vademecum [plDA20], cioè una serie di undici raccomandazioni che offrono linee guida per garantire la sicurezza del lavoro da remoto.

In effetti, un dispositivo che esce dall'azienda, ambiente potenzialmente controllato e protetto, diventa un rischio. Un dipendente potrebbe involontariamente navigare in siti di phishing e malware o connettersi a reti Wi-Fi infette, esponendo l'azienda al rischio di furti o compromissione di dati. Obiettivo fondante di questa tesi è lo sviluppo di un software facilmente installabile su un computer Windows, che consenta una protezione efficace e completa ai dispositivi desktop che escono dalla rete aziendale. Tale progetto dovrà filtrare la connessione dell'endpoint remoto, bloccando l'accesso a siti pericolosi, come virus e frodi, ma anche a contenuti indesiderati e distrazioni sul lavoro, come social network o videogiochi.

1.3 Contesto Aziendale

FlashStart offre già un software di protezione da malware e contenuti indesiderati dedicato ad endpoint remoti: ClientShield. Tale prodotto rappresenta un'estensione del filtro aziendale, fornendo lo stesso tipo di protezione, configurabile attraverso la medesima piattaforma cloud da parte di un amministratore di rete.

Allo stato dell'arte il prodotto utilizza una Virtual Private Network (VPN), per stabilire una comunicazione sicura con i server di FlashStart. Ciò garantisce che il dispositivo possa inviare e ricevere non soltanto le informazioni sul prodotto, ma anche richieste DNS, generalmente trasmesse in chiaro, in modo totalmente criptato.

L'esigenza di aggiornamento del software deriva dalla volontà di FlashStart di integrare il protocollo DNS over HTTPS (DoH), descritto in [PH18], nei suoi prodotti. Rispetto ad una classica VPN, DoH sfrutta la crittografia asimmetrica, implementata dal protocollo HTTP Secure (HTTPS), per garantire autenticazione, confidenzialità e integrità delle richieste DNS.

Il vantaggio che FlashStart trae da entrambi i protocolli è essenzialmente il medesimo: proteggere le comunicazioni DNS tramite la crittografia, evitando la possibilità di attacchi dovuti alla trasparenza delle informazioni. La differenza principale tra i due approcci è che una VPN cripta il traffico di rete nella sua interezza, mascherando anche l'indirizzo IP del dispositivo, mentre DoH si concentra esclusivamente sulla protezione delle richieste DNS.

Per un'azienda come FlashStart, specializzata in filtraggio DNS, mascherare l'indirizzo IP di un endpoint non è una funzionalità totalmente rilevante per l'erogazione del suo servizio. L'adozione del protocollo DoH consente quindi di ottenere la riservatezza necessaria per la comunicazione con il DNS, eliminando la necessità di un'infrastruttura VPN dedicata e riducendo l'overhead complessivo del sistema.

La novità principale del software prototipale illustrato in questa tesi, è dunque quella di fornire una soluzione per la transizione dalla tecnologia VPN, all'adozione del protocollo DoH. Di fondamentale importanza è inoltre l'analisi e la valutazione di soluzioni architetturali atte a garantire la naturale evoluzione del software, facilitando l'introduzione di nuove funzionalità.

Capitolo 2

Analisi dei Requisiti

2.1 Scopo del sistema e analisi del dominio

L'analisi del dominio è un punto chiave per quanto riguarda lo sviluppo software. Tale fase permette di comprendere il contesto in cui il software opera, gli attori principali ed i loro scopi. In questo progetto l'analisi del dominio è risultata molto rapida e diretta, in quanto il prodotto è stato sviluppato con un'azienda del settore informatico, permettendo fin da subito l'utilizzo di un linguaggio tecnico.

Come già introdotto nella sezione 1.2, il contesto in cui opera l'applicazione è quello della sicurezza informatica. Il fine ultimo di ClientShield è di fornire una soluzione di filtraggio DNS avanzata, che consenta un'efficace protezione ad un dispositivo che naviga al di fuori di una rete protetta. L'interazione col software coinvolgerà in particolare:

- **Utente finale:** cioè l'utilizzatore del dispositivo da proteggere, il quale navigherà normalmente in internet, utilizzando un browser ed inviando naturalmente richieste DNS.
- **Amministratore di rete:** il cliente FlashStart, responsabile dell'installazione e configurazione del software sul dispositivo attraverso la chiave fornita sulla piattaforma cloud. Tramite l'applicativo online potrà gestire le policy di filtraggio, aggiungendo o rimuovendo siti specifici o intere categorie di contenuti da bloccare.

- **Server FlashStart:** riceveranno le richieste DNS effettuate dal dispositivo protetto e le risolveranno applicando i filtri configurati dall'amministratore di rete.

Il software dovrà operare in modo tale da rendersi completamente trasparente all'utente finale, il quale non dovrebbe percepire alcuna differenza durante la normale navigazione.

2.2 Requisiti funzionali

Il progetto prevede lo sviluppo di un'implementazione prototipale, ciò implica che le funzionalità del software trattate in questa tesi rappresentino un sottogruppo delle capacità che dovrà avere il sistema. Tuttavia, poiché il risultato dello studio mira a gettare delle solide basi per una futura implementazione business di ClientShield, di seguito verranno elencati i requisiti del software nella sua interezza. Una visione complessiva, infatti, permette di valutare fin da subito la possibilità di una futura integrazione delle funzioni aggiuntive, consentendo un'analisi più efficace.

Nella sua implementazione prototipale, ClientShield dovrà:

- **Redirezionare le richieste DNS:** l'applicazione catturerà tutto il traffico DNS del dispositivo su cui è installata. Tutte le richieste devono essere redirezionate ai server FlashStart, sfruttando il protocollo DoH. La risoluzione dei domini deve essere effettuata dai server aziendali, ClientShield agirà da client DoH, facendo in modo di impedire la possibilità di DNS leak[1].

- **Permettere la registrazione di un dispositivo localmente:** dopo l'installazione del software, chiunque disponga di una valida chiave di registrazione, può registrare il dispositivo. Per effettuare tale operazione saranno richiesti:

[1] **DNS leak** si verificano quando le richieste DNS non sono cifrate e vengono trasmesse in chiaro. In questa condizione, un attaccante o qualsiasi entità che intercetti il traffico può monitorare i siti visitati dall'utente, compromettendo la privacy della navigazione.

  - *chiave di registrazione:* ottenibile tramite un profilo cloud con la corretta licenza.
  - *nome del dispositivo:* una qualsiasi stringa che faciliti l'identificazione dell'endpoint ad un amministratore di rete.

  Al termine della fase di registrazione, se avvenuta con successo, la protezione dovrà essere attivata e il dispositivo sarà visibile sulla piattaforma cloud.

- **Abilitare/disabilitare la protezione localmente:** accedendo all'applicazione, qualora la protezione sia disattivata, si dovrà offrire la possibilità di attivarla. Viceversa, se la protezione è attiva, l'utente avrà la possibilità di disattivarla inserendo un PIN. Tale codice è un numero intero presente sulla piattaforma cloud e dunque visualizzabile soltanto da un amministratore di rete.

- **Controllare lo stato:** l'applicazione dovrà periodicamente connettersi ai server FlashStart per verificare lo stato della protezione. Questa operazione nell'implementazione prototipale ha la funzione di verificare che il dispositivo sia ancora registrato sul cloud. La cancellazione dell'endpoint attraverso la piattaforma cloud, determinerà la disattivazione della protezione e la richiesta di una nuova registrazione.

- **Attivarsi automaticamente al riavvio:** se l'endpoint è già stato registrato, l'applicazione dovrà garantire il corretto funzionamento della protezione anche al riavvio. Ciò vuol dire che, all'accensione del dispositivo non dovrà essere necessaria, una nuova registrazione. ClientShield ricorderà le credenziali attive prima dello spegnimento, riattivando automaticamente la protezione all'accensione con le stesse.

Queste funzionalità costituiscono il complessivo comportamento del prototipo sviluppato in questo progetto. L'obiettivo essenziale è dunque garantire la protezione del dispositivo, fungendo da intermediario tra l'endpoint e i server di FlashStart. Con lo scopo di descrivere in modo semplice l'interazione tra utente, ClientShield e server DNS FlashStart è stato realizzato il diagramma in figura 2.1.

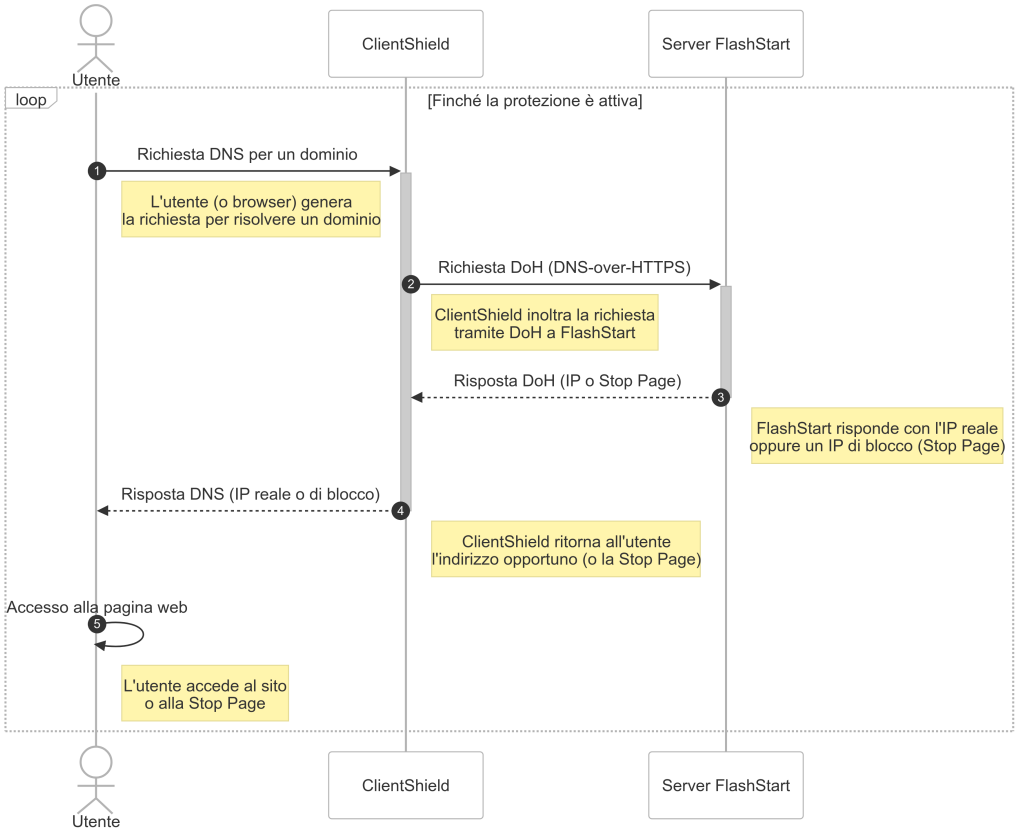

Figura 2.1: interazione tra utente, software e server FlashStart

Come precedentemente anticipato, la versione business dell'applicazione garantirà una serie di funzioni minori in più, di seguito riportate:

- **Blocco remoto della protezione:** deve essere possibile disattivare la protezione da remoto, questa funzione può essere utilizzata sia dall'amministratore di rete, attraverso la piattaforma cloud, sia direttamente dai server FlashStart. Durante i controlli di stato periodici, infatti, verrà verificata la validità della licenza e, se non risultasse valida, la protezione verrà disattivata automaticamente, impedendo qualsiasi tentativo di riattivazione locale.

- **Invio di notifiche:** durante la registrazione sarà possibile decidere di ricevere alcune notifiche dall'applicazione. Se attivate, il dispositivo riceverà

soltanto le notifiche riguardanti lo stato della protezione del dispositivo, ad esempio licenza scaduta o disabilitazione remota.

- **Applicazione unbranded:** l'applicazione prima della registrazione dovrebbe avere un nome generico e non presentare alcun logo. Dopo la registrazione dovrà essere possibile ottenere informazioni sull'acquirente, come il nome scelto per il software e il logo da applicare. L'interfaccia grafica dell'applicazione dovrà adattarsi alle informazioni ottenute durante la registrazione.
- **Controllo di stato periodico:** questa operazione è già presente in parte nell'implementazione prototipale, tuttavia nello sviluppo business dovrà reperire più informazioni. I dati supplementari, come ad esempio:
  - informazioni sul blocco remoto
  - nome e logo dell'acquirente
  - cambio di impostazioni sulle notifiche

  permetteranno l'implementazione delle funzionalità aggiuntive sopra riportate.

Come si può notare, le funzionalità business rappresentano più che altro un'estensione di funzioni già presenti nel modello prototipale. Con una corretta analisi del software, dunque, dovrebbe essere possibile aggiungere tutte le funzionalità supplementari al sistema descritto in questa tesi senza particolari sforzi.

2.3 Requisiti non funzionali

Dopo aver definito in dettaglio *cosa* deve fare ClientShield, è altrettanto importante stabilire *come* debba farlo. Oltre ai requisiti funzionali del sistema, infatti, il progetto prevede anche dei requisiti non funzionali, che mirano a garantire un livello qualitativo adeguato al sistema. Questa categoria di requisiti ha l'obiettivo di soddisfare un potenziale cliente. L'aderenza ad essi dovrebbe trasmettere la professionalità di FlashStart, garantendo un'esperienza di utilizzo piacevole per gli utilizzatori finali.

Semplicità

Di fondamentale importanza è rendere il sistema semplice: rapido da installare e facile da configurare. FlashStart ha esplicitamente sottolineato quanto questo requisito sia cruciale per garantire la qualità del software. La richiesta è naturalmente intuitiva: un'installazione e una configurazione rapide non solo migliorano l'esperienza dell'utente, ma rendono il software più facilmente integrabile nei diversi contesti aziendali. L'automatizzazione completa del processo di installazione consente agli amministratori di rete di implementare ClientShield senza difficoltà, ottimizzando i tempi e riducendo il rischio di errori nella configurazione.

Efficienza

Anche nello sviluppo prototipale, ClientShield dovrebbe garantire un'efficienza tale da risultare totalmente trasparente all'utente. L'obiettivo è limitare qualsiasi impatto sulla navigazione, un utente qualsiasi del dispositivo dovrebbe potersi scordare della presenza di un filtro. Ciò implica una rapida gestione delle richieste DNS, consentendo una veloce risoluzione dei domini, limitando di conseguenza i tempi di caricamento delle pagine web.

Questo requisito è favorito dall'abbandono dell'utilizzo di una VPN in favore del protocollo DoH e dalla velocità generale dei server di FlashStart, che, a febbraio 2025, risulta essere il DNS più veloce al mondo secondo [DNS25].

Resilienza

Per garantire la corretta protezione del dispositivo, il servizio di filtraggio dovrebbe poter essere disattivato esclusivamente da un amministratore del dispositivo. Ciò vuol dire che ClientShield dovrà assicurare il funzionamento continuo della protezione in qualsiasi condizione, anche nel caso in cui l'utente tenti di disattivare il filtro. È requisito essenziale per il software fare in modo che nessuna operazione effettuata da utenti senza privilegi amministrativi, come il banale cambio di rete, possa interrompere la protezione.

2.4 Casi d'uso

Allo scopo di descrivere in modo dettagliato il comportamento atteso del software, vengono di seguito riportati anche i casi d'uso del sistema. Ciascun caso d'uso rappresenta una specifica interazione tra gli attori e ClientShield, chiarificando le azioni che possono essere eseguite e le risposte da parte dell'applicazione. Questa visione è particolarmente utile, perché pone l'attenzione sull'utente, evidenziando come interagirà con ClientShield e cosa percepirà in seguito ad ogni possibile azione eseguita. Di seguito sono riportati i principali casi d'uso del prototipo:

- **Utente avvia l'applicazione:** all'avvio dell'applicazione se il dispositivo non risulta registrato, l'utente verrà indirizzato ad una schermata di registrazione. Se invece l'utente è già registrato e la protezione è disattivata, allora verrà offerta la possibilità di attivarla. Viceversa, se è già registrato e la protezione è attiva, deve essere disponibile un'opzione per disattivarla.

- **Registrazione:** per registrarsi l'utente dovrà inserire i campi richiesti (chiave di profilo e nome del dispositivo). Se la registrazione ha avuto successo la protezione deve essere attivata, iniziando subito a filtrare le richieste DNS. Per la comunicazione con i server FlashStart sarà necessario un codice utente univoco, ottenibile come risposta alla fase di registrazione. In caso di errore, la causa deve essere visibile dall'utente, che potrà ritentare.

- **Disattivazione protezione:** per disattivare la protezione tramite l'applicazione sarà necessario l'inserimento di un codice PIN. Il PIN è un numero intero visualizzabile dalla piattaforma cloud e deve essere validato. Se il codice è corretto, la protezione viene disattivata; in caso contrario, sarà mostrato un messaggio di errore e l'utente potrà ritentare.

- **Utente riavvia il dispositivo:** al riavvio del computer, se l'endpoint era stato correttamente registrato prima dello spegnimento, la protezione deve essere automaticamente riattivata. ClientShield dovrà recuperare il codice univoco fornito in fase di registrazione, che sarà utilizzato, come sempre, per la comunicazione con i server FlashStart.

- **Utente naviga sul web:** la navigazione su un browser deve avvenire come di consueto. In caso di accesso ad un contenuto bloccato deve essere visibile una pagina di stop, che riporti le motivazioni del blocco.

Per riassumere le azioni che un utente può compiere in ciascun momento è stato realizzato la schema in figura 2.2.

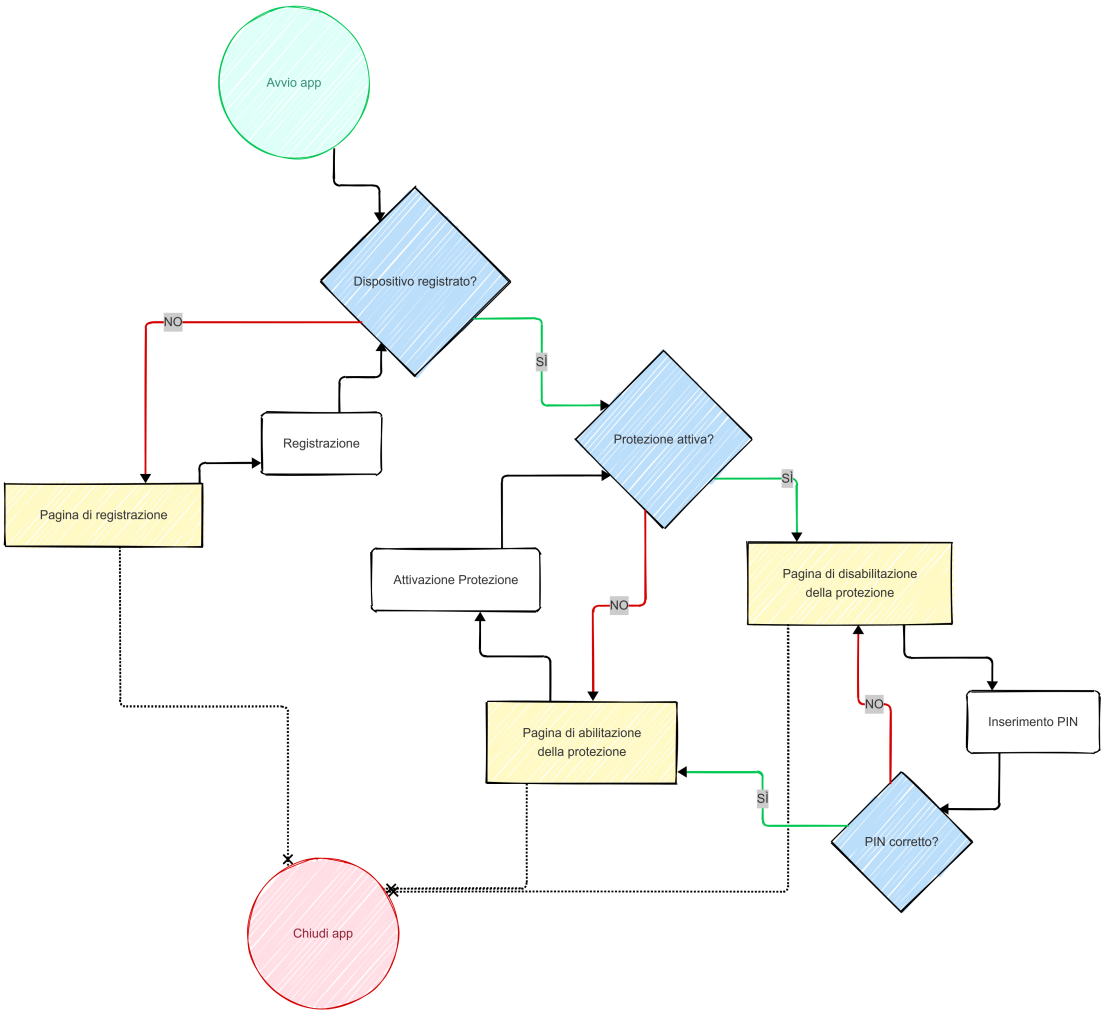

Figura 2.2: navigazione dell'utente nell'app

Capitolo 3

Analisi e progettazione

3.1 Struttura del progetto

Lo sviluppo del progetto è stato naturalmente preceduto da una necessaria fase di analisi. Identificati i requisiti del sistema, espressi nel capitolo 2, risulta necessario definire come debba essere strutturato il software sulla base di essi. Durante l'intera fase di progettazione si è tenuto conto del fatto che ClientShield ha come target di piattaforma Windows, in particolare a partire dalla versione 10.

Dall'analisi dei requisiti emerge chiaramente la presenza di due componenti ben distinti. ClientShield, infatti, dovrà costantemente filtrare richieste DNS, interagendo in maniera continuativa con il sistema operativo. L'interazione diretta con l'utente, invece, è sporadica e avviene soltanto in caso di necessità, attraverso un'interfaccia grafica. Per tale motivo, si è deciso fin da subito di suddividere lo sviluppo del software in due sotto-progetti, collegati tra loro, ma ben distinti. Un'organizzazione di questo tipo permette di ottenere vantaggi importanti.

Anzitutto, la divisione in due sotto-progetti induce ad una minore complessità generale del sistema e ad una chiara suddivisione delle responsabilità. Tale approccio, inoltre, da un punto di vista aziendale, facilita la gestione del codice e un'eventuale suddivisione del lavoro tra più sviluppatori.

In secondo luogo, sebbene il servizio di filtraggio debba necessariamente interagire con il sistema operativo, l'interfaccia utente e la logica di configurazione del filtro non hanno vincoli di piattaforma. Ciò apre la possibilità ad una potenziale

estensione multi-platform[1] per la parte grafica. Di conseguenza, un'eventuale espansione a sistemi operativi diversi (ad esempio Linux) richiederebbe unicamente la modifica del modulo di filtraggio.

3.1.1 Organizzazione dei sotto-progetti

Sulla base di questa suddivisione logica, ClientShield è stato organizzato in quattro diversi sotto-progetti:

- **Servizio di filtraggio:** questo modulo rappresenta la parte centrale del software, sarà il responsabile dell'intercettazione e del filtraggio delle richieste DNS. Dovrà essere implementato come servizio Windows, garantendo un'esecuzione continua e automatica all'avvio del sistema, definendo inoltre strategie di recupero in caso di errori.

- **Interfaccia utente:** questo modulo rappresenta la parte del software con cui l'utente interagirà. Il ruolo del sotto-progetto è di consentire la comunicazione con il servizio di filtraggio attraverso un'interfaccia grafica intuitiva.

- **Modulo comune:** questo modulo conterrà variabili, classi e configurazioni condivise tra interfaccia e servizio di filtraggio. Qui saranno specificati anche i tipi di messaggio che possono essere scambiati tra servizio e interfaccia, permettendo di mantenere sempre integrità di comunicazione tra i due sotto-progetti.

- **Installer:** questo modulo è dedicato alla creazione dell'installer del software. Si occuperà di automatizzare l'installazione del servizio e dell'interfaccia, garantendone il funzionamento corretto.

L'organizzazione del progetto finale seguirà il più possibile la struttura illustrata in figura 3.1.

[1] La programmazione **multi-platform** si pone l'obiettivo di consentire allo stesso codice di poter essere eseguito su macchine aventi sistemi operativi differenti.

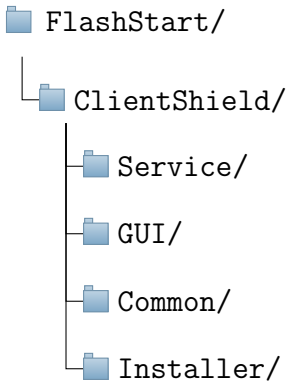

Figura 3.1: Struttura dei sotto-progetti di ClientShield

3.2 Comunicazione interprocesso

Vista la divisione in progetti separati, la successiva fase di analisi ha preso in considerazione la modalità di comunicazione tra i due progetti. L'architettura generale di ClientShield assomiglia molto ad un sistema client-server. Infatti, l'interfaccia grafica deve stabilire una connessione con il servizio Windows per inviare dei messaggi contenenti gli input dell'utente e la richiesta di svolgere determinate operazioni, per poi attendere la risposta del servizio.

Per permettere la corretta comunicazione tra il servizio Windows e l'interfaccia grafica, è necessario stabilire un protocollo di comunicazione inter-processo. Le caratteristiche fondamentali di tale protocollo sono garantire una comunicazione affidabile e di tipo half-duplex[2] tra due processi separati, ma in esecuzione sullo stesso dispositivo.

Per la scelta del protocollo, dovendo sviluppare in ambiente Windows, ci si è riferiti alla documentazione Microsoft, che presenta l'articolo [Mic24] sulle comunicazioni inter-processo. Si è optato per l'utilizzo della comunicazione tramite named pipes, consigliata soprattutto per la sua affidabilità e semplicità di implementazione, grazie al supporto nativo offerto da Windows.

[2]La comunicazione **half-duplex** tra due nodi permette la trasmissione di messaggi da parte di entrambi, ma soltanto uno per volta.

Le named pipes permettono di stabilire un canale di comunicazione identificato da un nome univoco, accessibile sia dal servizio che dall'interfaccia senza necessità di configurazioni complesse. Un ulteriore vantaggio è la loro efficienza: essendo ottimizzate per la comunicazione all'interno dello stesso sistema, garantiscono una latenza ridotta rispetto ad altre tecniche di comunicazione inter-processo. Infine le named pipes offrono la possibilità di definire controlli di accesso a livello di sistema operativo, impedendo la manipolazione dei dati da parte di processi non autorizzati, garantendo che solo l'interfaccia ufficiale possa inviare comandi al servizio, evitando interferenze da applicazioni di terze parti.

Scelto il protocollo, si è scelta la struttura dei messaggi. Vista la vasta disponibilità di librerie capaci di serializzare e deserializzare classi in formato JSON, si è deciso di utilizzare quest'ultimo per la comunicazione. La scelta è dovuta al fatto che i dati presenti nel messaggio dipendono dall'operazione e possono essere anche di tipo diverso. Infatti la disabilitazione, come sottolineato in sezione 2.4, necessita l'inserimento di un numero intero. Ciò significa che l'invio di semplici stringhe di testo richiederebbe controlli manuali sulla struttura del messaggio e sui tipi di ogni singolo campo. JSON permette l'automatizzazione dei controlli: sarà sufficiente definire delle classi rappresentanti i diversi tipi di messaggio nella comunicazione e delegare la conversione da classe a JSON, o viceversa, ad una libreria esterna, gestendo eventuali errori.

In figura 3.2 viene illustrato il possibile scambio di messaggi tra interfaccia grafica e servizio Windows.

3.3 Servizio Windows

Il servizio di filtraggio si compone essenzialmente di due parti: una gestisce la comunicazione inter-processo, l'altra si occupa del filtraggio DNS. Dei due componenti, la comunicazione inter-processo deve essere sempre disponibile, ma il servizio di filtraggio si potrà attivare e disattivare a comando. Per questo motivo, il servizio Windows deve essere costituito principalmente come un server named pipe, mentre l'analisi si concentra sull'organizzazione del servizio di protezione.

Nel vecchio software, per assicurare che tutto il traffico passasse dal filtro, veniva creata una scheda di rete virtuale. Modificando le metriche di priorità delle

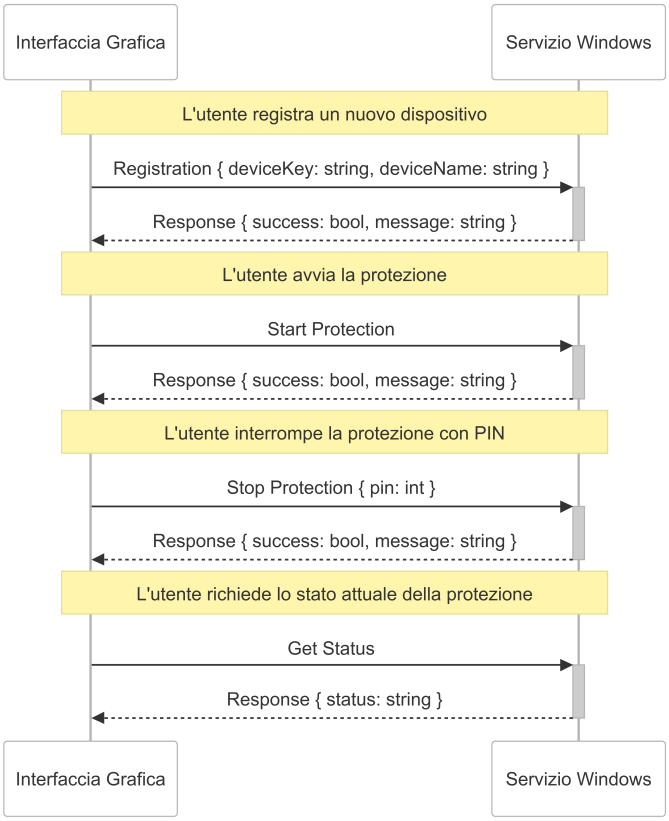

Figura 3.2: Scambio di messaggi tra GUI e servizio Windows

interfacce si rendeva quest'ultima la più "alta in graduatoria", così che il sistema instradasse il traffico su di essa prima di qualsiasi altra. Questo permetteva di intercettare le richieste e bloccare o modificare il traffico indesiderato. Tuttavia, si trattava di una soluzione più complessa da gestire e potenzialmente soggetta a conflitti con altri driver o componenti di rete.

Nella nuova versione di ClientShield, invece, si è deciso di implementare un server DNS locale, soluzione che offre dei vantaggi considerevoli:

- **Semplicità di implementazione:** configurare un processo che ascolti sulla porta 53 e risolva le richieste DNS è decisamente più semplice rispetto alla creazione di una scheda di rete virtuale. Tale scelta, inoltre, limita anche il rischio di incompatibilità con software terzi che gestiscono altre schede di rete virtuali.

- **Manutenibilità:** un server locale consente di isolare la responsabilità del software semplicemente al filtraggio. Questa soluzione permette una maggiore chiarezza e una più corretta aderenza al Single Responsibility Principle (SRP). In questo modo, infatti, il modulo di protezione gestisce solo richieste DNS, migliorando il debugging, che sarà limitato ad un singolo componente.

- **Portabilità:** l'implementazione di un server DNS non ha dipendenze specifiche dal sistema operativo. Le soluzioni esplorate in seguito consentono di relegare l'interazione diretta con il sistema operativo alla sola impostazione del server locale come DNS predefinito.

In sintesi, il nuovo design del servizio Windows prevede l'implementazione di un server named pipe che, su comando dell'utente, interagisca con il servizio di protezione per avviare l'esecuzione di un server DNS locale, impostandolo come predefinito di sistema, oppure interromperla.

3.3.1 Analisi della protezione

L'avvio della protezione è un problema più complesso di quanto possa sembrare. Per attivarla è necessaria l'interazione con molti componenti del sistema. Il server named pipe dovrebbe gestire le API FlashStart, occupandosi della registrazione, dell'autorizzazione e dei controlli periodici sullo stato del servizio. Inoltre, deve monitorare lo stato del server DNS e interagire con esso, permettendone il riavvio e l'arresto automatici o su richiesta. In aggiunta, il servizio deve occuparsi anche della gestione delle credenziali, memorizzando il codice utente fornito durante la registrazione e garantendo il ripristino della sessione al riavvio del dispositivo.

Per una corretta delegazione delle responsabilità e per ottenere semplicità nella comunicazione con diversi componenti, è stato sfruttato il pattern *Facade*, definito in [GHJV94]. Questo pattern fornisce una semplice interfaccia che consente la gestione di funzionalità distribuite su più classi.

Dalla struttura delle classi, illustrata in figura 3.3, è possibile constatare che la classe `ServerNamedPipe` si occupa esclusivamente di ricevere ed elaborare le richieste da parte del client, delegando l'esecuzione di qualsiasi operazione alla classe `ProtectionManager`. Quest'ultima funge da punto di raccordo tra

le diverse funzionalità offerte dal software, gestendole in maniera organizzata, permettendo una granularità maggiore che facilita la manutenzione del codice. `ProtectionManager` coordina l'esecuzione delle funzionalità offerte dalle classi

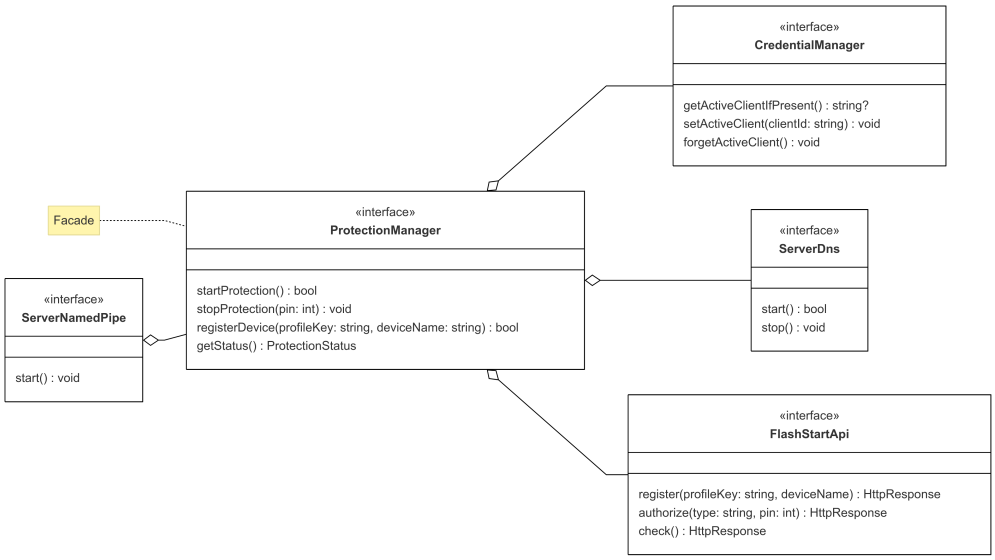

Figura 3.3: Facade pattern per la gestione della protezione

gestite, consentendo una gestione implicita delle credenziali affidata completamente a `CredentialManager`. Inoltre, `ProtectionManager` verifica costantemente la coerenza tra lo stato dei componenti e, se presente una sessione attiva, avvia automaticamente la protezione.

3.3.2 Analisi server DNS

Il server DNS locale rappresenta il nucleo centrale di ClientShield, poiché gestisce sia la risoluzione dei domini che l'applicazione dei filtri. Per garantire flessibilità ed estensibilità, il server è stato suddiviso in più componenti, ciascuno con una funzionalità specifica. La scelta è motivata da due fattori principali. In primo luogo, suddividere il server in componenti più piccoli ne facilita l'estensibilità, consentendo di aggiungere nuove funzioni senza alterare la struttura di base. Inoltre, la suddivisione delle funzionalità consente il più facile impiego di thread-pools, consentendo la creazione di worker sulla base dei componenti.

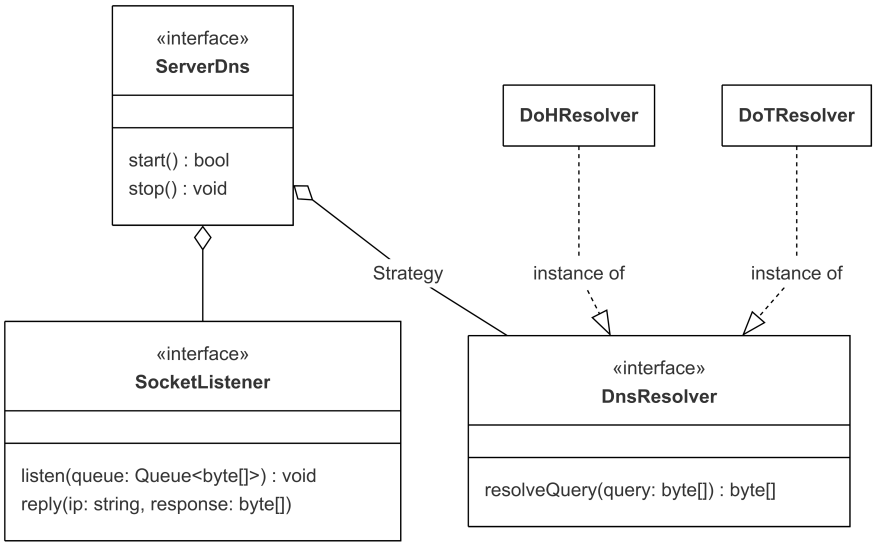

Figura 3.4: Strategy pattern per Server DNS

Come illustrato in figura 3.4, `DnsServer` utilizza il pattern *Strategy* per la risoluzione dei domini. Delegando la risoluzione dei domini a una classe secondaria, `DnsResolver`, è possibile implementare tecnologie differenti. Sebbene i requisiti del prototipo richiedano la sola implementazione DoH, FlashStart ha già implementato delle API che utilizzano il protocollo DNS over TLS (DoT). Se in futuro l'azienda decidesse di adottare DoT o un qualsiasi altro tipo di tecnologia per la risoluzione dei domini, sarà sufficiente sostituire il `DnsResolver` con un'implementazione specifica, senza modificare il resto del sistema.

Un altro aspetto fondamentale riguarda la gestione delle richieste in ingresso. Poiché la porta 53, dedicata alla comunicazione DNS, può essere gestita da un solo componente alla volta, la ricezione dei pacchetti è affidata alla classe `SocketListener`. Tale entità ha il compito di minimizzare i tempi di gestione delle query in arrivo, prevenendo la perdita di richieste. Per massimizzare l'efficienza, `SocketListener` utilizzerà una coda di messaggi, permettendo alla classe `DnsServer` di elaborare le richieste in parallelo. Se necessario, potranno essere utilizzati worker dedicati per ottimizzare le prestazioni e garantire l'elevata reattività del sistema.

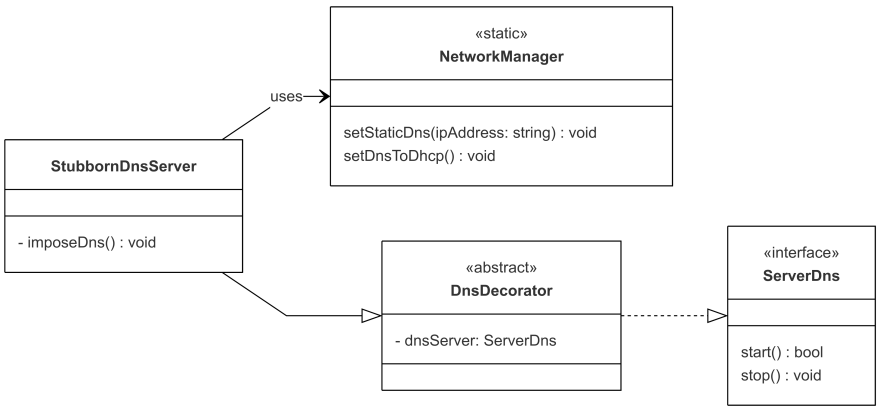

Figura 3.5: Decorator pattern per Server DNS

Infine, l'ultimo e cruciale compito affidato al servizio Windows è impostare il server DNS locale come predefinito di sistema. Questa fase è decisamente delicata, perché rappresenta il punto in cui ClientShield interagisce direttamente con il sistema operativo, diventando di fatto dipendente dalla piattaforma.

La soluzione utilizzata in questo caso, illustrata in figura 3.5, sfrutta il pattern *Decorator*. Questa scelta permette di isolare la logica di interazione con Windows in una classe secondaria, migliorando il riuso. Ad interagire con il sistema operativo è la classe statica `NetworkManager`, che consente di configurare il DNS di sistema come statico, specificando un indirizzo, oppure di ripristinarlo in modalità dinamica. Il decoratore `StubbornDnsServer` permette di aggiungere le funzionalità di `NetworkManager` a `DnsServer`. In questo modo è possibile estendere i metodi per avviare e interrompere il server, garantendo, rispettivamente, l'aggiornamento e il ripristino automatico del DNS predefinito sul dispositivo. L'adozione del pattern *Decorator* risulta particolarmente utile, in quanto consente di isolare l'utilizzo di librerie specifiche per Windows a due sole classi, aprendo la possibilità ad un futuro sviluppo multi-platform.

3.4 Interfaccia grafica

L'interfaccia grafica è l'unica parte del software con cui l'utente interagirà direttamente. Il suo compito è essenzialmente quello di permettere all'utilizzatore

l'accesso ai comandi della protezione, esposti tramite il server pipe dal servizio Windows. L'interfaccia agisce come client nella connessione tramite named pipe, inviando le richieste effettuate dall'utente e attendendo la risposta dal server.

Poiché il software è soltanto un prototipo, l'obiettivo dell'analisi è proporre una soluzione che permetta una facile modifica della parte visiva. L'interfaccia sarà sviluppata molto semplicemente, dando solo una rapida rappresentazione grafica ai comandi che è possibile effettuare. Dal punto di vista aziendale, il prototipo risulta efficace se la modifica dell'interfaccia, che verrà senza dubbio sottoposta al consiglio di grafici esperti, richiede poche o nulle modifiche di backend.

3.4.1 Model-View-Presenter

Per lo sviluppo dell'interfaccia grafica è stato utilizzato il pattern architetturale *Model-View-Presenter (MVP)*. MVP è un'architettura software che deriva dal famosissimo pattern *Model-View-Controller (MVC)*, nata con lo scopo di separare la logica di presentazione dall'interfaccia, in modo da migliorare la manutenibilità e il testing del codice.

Si compone di tre elementi principali:

- **Model:** offre un'interfaccia per reperire i dati che devono essere visualizzati, l'ottenimento dei quali può coinvolgere eventualmente anche la comunicazione tramite API.

- **View:** è la parte responsabile della rappresentazione dei dati. La *View* si occupa esclusivamente di eseguire chiamate al suo *Presenter* ogni qual volta l'utente interagisca con l'interfaccia. Tipicamente è un componente totalmente passivo, le cui funzionalità sono ridotte al minimo e contengono poca, o nessuna logica. Non comunica direttamente con la parte di *Model*, tutti i dati passano attraverso il *Presenter*, lasciando alla *View* il solo compito di rappresentarli visivamente.

- **Presenter:** implementa l'intera logica di presentazione, occupandosi di reperire i dati dal *Model* ed eventualmente modificarli in modo che possano essere rappresentati. Il *Presenter*, inoltre, riceve i dati in input dalla

View, li valida e si occupa della loro elaborazione. Svolge il ruolo di mediatore tra *Model* e *View*, occupandosi di verificare sempre la coerenza tra dati e rappresentazione.

A causa della loro stretta interconnessione, Presenter e View detengono riferimenti reciproci, consentendo ad ognuno di chiamare i metodi dell'altro. Ogni View necessita il suo specifico Presenter e tipicamente lo instanzia essa stessa.

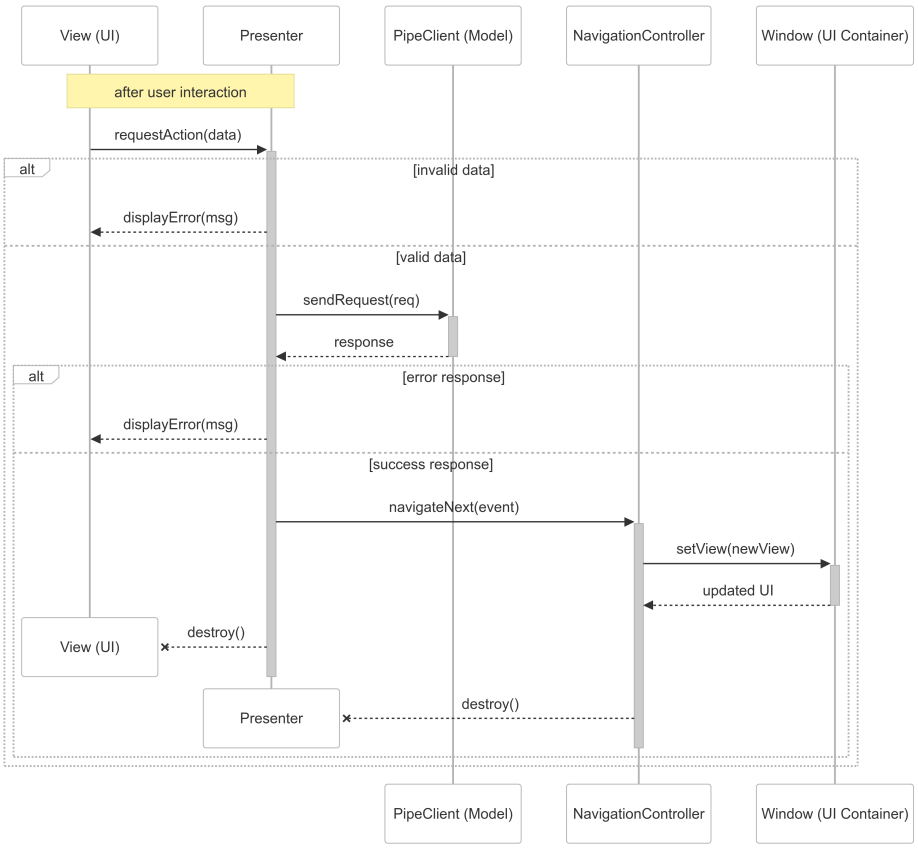

Figura 3.6: Diagramma di sequenza pattern MVP

In figura 3.6 è possibile visualizzare i componenti dell'architettura proposta, che trae ispirazione dell'articolo [Ram10]. L'immagine offre una rappresentazione delle interazioni che dovrebbero avvenire in una corretta implementazione dell'architettura.

Nell'interfaccia grafica di ClientShield non esiste un vero e proprio Model, i dati da rappresentare risultano essere gli stati della protezione. Pertanto, a svolgere

tale ruolo è `PipeClient`, che è il responsabile della comunicazione con il servizio Windows, dal quale ottiene risposte sull'esito delle operazioni e sullo stato della protezione.

Stando a quanto riportato in sezione 2.4, le possibili `View` sono tre e rispecchiano gli stati di ClientShield: non registrato, protezione attiva, protezione disattivata. Ognuna di esse dovrà avere il proprio `Presenter` che consenta di validare e processare i dati delle richieste utente.

L'architettura proposta sostituisce all'implementazione di una coda di eventi, utilizzata in [Ram10], l'utilizzo della classe `NavigationController`. Quest'ultima si configura come un *observer* per i `Presenter`, che notificano gli eventi scatenati dall'interfaccia grafica. Tali eventi, che rappresentano tipicamente il passaggio da uno stato all'altro, necessitano una visione globale sull'applicazione ed una gestione centralizzata, compito che, pertanto, non può essere svolto da un semplice `Presenter`.

Capitolo 4

Implementazione

La successiva, ed ultima, fase del progetto ha riguardato lo sviluppo, che ha compreso la scelta delle tecnologie da utilizzare e l'effettiva scrittura di codice. Questa fase è stata indubbiamente resa più veloce e diretta grazie all'attento lavoro di documentazione, analisi e progettazione descritto nei capitoli precedenti.

4.1 Scelta del linguaggio

Il linguaggio scelto per lo sviluppo di ClientShield è stato C# con .NET 8. Questa scelta è stata piuttosto naturale, sia perché è lo stesso linguaggio utilizzato dalla precedente implementazione di FlashStart, ma soprattutto grazie alle caratteristiche del linguaggio, che si adeguano perfettamente ai requisiti del software. Innanzitutto C# è un linguaggio moderno, che consente uno sviluppo ed una manutenzione più semplici, grazie alla sua sintassi chiara e funzionalità avanzate. L'utilizzo di questo linguaggio garantisce un ottimo equilibrio tra efficienza e semplicità, garantendo una gestione automatica della memoria tramite la garbage collection[1] ed una vasta disponibilità di librerie e framework che facilitano lo sviluppo.

Analogamente a Java, C# supporta la programmazione multi-platform, ma risulta più adatto per ClientShield grazie alla perfetta integrazione dell'ecosistema

[1]La **garbage collection** identifica e recupera automaticamente la memoria occupata da oggetti non più utilizzati, evitando che il programma esaurisca lo spazio disponibile.

.NET con Windows, che rappresenta il target principale del software. C# , infatti, ha accesso facilitato alle API di Windows grazie a librerie ufficiali sviluppate da Microsoft, che fornisce inoltre guide e documentazione utile all'implementazione e installazione di servizi Windows sviluppati in .NET.

Il linguaggio supporta anche l'implementazione di interfacce utente native attraverso Windows Presentation Foundation (WPF), che permette l'applicazione del pattern MVP scelto in fase di analisi. Inoltre la comunicazione tramite named pipes è supportata nativamente e vengono offerti diversi strumenti per la creazione di un installer. Ciò consente l'uso del medesimo linguaggio per tutti i progetti definiti in sezione 3.1.1, garantendo coerenza nell'intera codebase, oltre a permettere una struttura assolutamente simile a quella prevista.

La scelta del linguaggio conduce all'ovvia adozione di Visual Studio come ambiente di sviluppo integrato (IDE)[2], che offre suggerimenti di codice, editor integrato di interfacce utente, ottimi strumenti di debugging, supporto per il version control di Git e tantissime altre funzionalità che lo rendono una scelta ottime per questo linguaggio.

La directory di ClientShield segue la struttura illustrata in figura 3.1, è stata creata un'unica soluzione composta da tre dei quattro progetti previsti:

- **Service:** è un progetto di tipo Worker Service, template presente in Visual Studio e adatto allo sviluppo di un servizio Windows.

- **Client:** è un progetto di tipo WPF, adatto allo sviluppo di una moderna interfaccia grafica per sistemi Desktop.

- **Common:** questo progetto contiene i messaggi e le configurazioni della comunicazione inter-processo, utilizzabili sia da Service, che da Client.

L'installer non è stato sviluppato nella versione prototipale.

[2]Un **IDE** è un software che fornisce strumenti come editor di testo, compilatori e debugger per facilitare la scrittura e la gestione del codice.

4.2 Tecnologie di supporto

NuGet

Per lo sviluppo di ClientShield sono state utilizzate diverse librerie esterne, perciò, allo scopo di gestire le dipendenze del progetto, è stato utilizzato NuGet[3], il gestore di pacchetti ufficiale per l'ecosistema .NET.

NuGet permette di aggiungere, aggiornare e rimuovere pacchetti esterni con facilità, evitando la necessità di includere manualmente le librerie. Ha un ruolo chiave nel processo di sviluppo, in quanto, grazie al controllo di versione sulle dipendenze, segnala problemi di incompatibilità tra le librerie di un progetto, evitando potenziali errori difficili da individuare manualmente.

Git

Per lo sviluppo è stato fondamentale l'utilizzo di Git[4], un sistema di versionamento, che permette un efficace controllo sulle modifiche, consentendo di preservare l'integrità del codice. Git è senza dubbio uno dei sistemi di versionamento più utilizzati, grazie alla capacita di gestire in modo molto efficiente anche progetti particolarmente complessi.

Il software consente di tenere traccia di ogni modifica apportata al codice, permettendo di ripristinare versioni precedenti in caso di errori o di confrontare diversi stati del progetto. Nel contesto di ClientShield, trattandosi di un prototipo sviluppato da zero, il lavoro si è concentrato prevalentemente su un unico branch principale, evitando la necessità di gestire rami multipli tipici di progetti più strutturati. Per migliorare la leggibilità dello storico delle modifiche è stata adottata una struttura standardizzata per i commit, preceduti da tag identificativi che ne indicano il tipo: ad esempio i tag `[DOC]` per la documentazione o `[NEW]` per le nuove funzionalità.

Un ulteriore vantaggio di Git è la grande compatibilità con piattaforme cloud come GitHub[5], consente di lavorare su computer diversi con codice sempre

[3]Sito ufficiale NuGet: `http://nuget.org`
[4]Sito ufficiale di Git: `http://git-scm.com`
[5]Sito ufficiale di GitHub: `http://github.com`

sincronizzato.

Librerie di gestione JSON

In fase di analisi si è scelto di utilizzare JSON per la comunicazione inter-processo. Tale scelta ha reso possibile l'utilizzo di un meccanismo di deserializzazione che avesse un modello di ereditarietà, in modo da poter effettuare un casting intelligente.

A tal scopo è stata utilizzata la libreria JsonSubTypes[6], che consente di definire una classe base comune per tutte le richieste, semplificando la gestione dei messaggi. Invece di trattare ogni richiesta come entità separata, il sistema utilizza un tipo di richiesta generico, che viene poi convertito in richieste specifiche, ad esempio di autorizzazione o registrazione. Il controllo e la conversione da messaggio generico a specifico sono gestiti direttamente dalla libreria, eliminando la necessità di controlli manuali sulla struttura delle richieste e migliorando la pulizia del codice.

La serializzazione e la deserializzazione sono gestite attraverso la libreria Newtonsoft.Json[7]. Questa libreria è in grado di lavorare assieme a JsonSubTypes per la corretta conversione da classe a JSON e viceversa, offrendo semplicità, ottime prestazioni e individuazione automatica di errori di conversione.

4.3 Implementazione progetto Service

Il progetto per l'implementazione del servizio Windows è stato sviluppato a partire dalla guida [Mic21] realizzata da Microsoft. Il servizio è stato sviluppato, come suggerito dalla guida, attraverso un `BackgroundService` che, all'avvio, instanzia un `ServerNamedPipe` e un `ProtectionManager`, adeguatamente introdotti nella sezione 3.3.1 ed illustrati in figura 3.3.

ServerNamedPipe è una classe implementata facendo uso delle librerie di sistema e funge da server per la comunicazione inter-processo. Il server sarà sempre in ascolto sulla pipe di nome "FlashStart-ClientShield-Service-Pipe" finché

[6]Sito ufficiale JsonSubTypes: `http//manuc66.github.io/JsonSubTypes/`
[7]Sito ufficiale Newtonsoft.Json: `http://www.newtonsoft.com/json`

il servizio Windows è attivo. Il costruttore necessita il passaggio di un oggetto `ProtectionManager`, col quale il server si interfaccia per eseguire i comandi ricevuti dalla named pipe.

ProtectionManager è la classe responsabile della gestione globale del servizio di protezione, coordinando la comunicazione con le API FlashStart, l'esecuzione del server DNS locale e la gestione delle credenziali. Questa classe funge da punto di raccordo tra le classi che offrono i suddetti servizi:

- **CredentialManager:** viene utilizzato all'instanziazione di ProtectionManager per controllare l'esistenza di un cliente già registrato. Durante le fasi di registrazione e logout questa classe di occupa di salvare o eliminare le credenziali sul dispositivo.
- **FlashStartClient:** viene utilizzato durante tutta l'esecuzione del servizio per interfacciarsi con le API di FlashStart.
- **DnsServer:** classe dedicata alla gestione del server DNS locale, viene inizializzato da `ProtectionManager` all'avvio della protezione.

CredentialManager utilizza il gestore di credenziali Windows per l'archiviazione del Client ID, cioè il codice utente univoco necessario per l'autenticazione con i server FlashStart. Questa classe è sviluppata come wrapper dell'omonimo modulo appartenente alla libreria Meziantou.Framework.Win32, gestendo automaticamente le configurazioni necessarie per consentire una più facile interazione da parte di `ProtectionManager`. I metodi esposti consentono in particolare la registrazione, l'eliminazione, l'aggiornamento e la visualizzazione (se presente) del codice utente.

FlashStartClient utilizza un client HTTP fornito dalle librerie di sistema per effettuare richieste ai server FlashStart, impostandone automaticamente l'header di autenticazione. Questa classe fornisce metodi per permettere di effettuare le richieste di *registrazione*, *autorizzazione* e *check*.

4.3.1 Implementazione server DNS

Il server DNS rappresenta il cuore del progetto e la sua implementazione è un aspetto cruciale, poiché le future modifiche del software riguarderanno prevalentemente questa parte. È di fondamentale importanza aderire alla struttura proposta in fase di analisi e rendere il codice chiaro ed organizzato, in modo tale da facilitarne la modifiche e, soprattutto, l'estensione. L'implementazione descritta di seguito segue le linee guida delineate nella sezione 3.3.2, sfruttando i pattern progettuali individuati per ottenere gli obiettivi di manutenibilità, estensibilità ed efficienza.

ClientShield utilizza `StubbornDnsServer` come server DNS. Questa classe funge da decorator per l'implementazione di base `DnsServer`, utilizzando il pattern per estendere le funzionalità del server DNS, consentendogli di imporsi come predefinito di sistema. Per consentire l'operazione, `StubbornDnsServer` si avvale dei metodi esposti dalla classe `NetworkManager`, responsabile della gestione delle impostazioni di rete.

NetworkManager è una classe statica che si occupa della configurazione del DNS del dispositivo. Interfacciandosi direttamente con le librerie di sistema di Windows, `NetworkManager` consente di impostare il DNS su uno specifico indirizzo statico e di ripristinarlo alla modalità automatica quando necessario. Il codice prodotto sfrutta la libreria *System.Management* per interagire direttamente con il sistema operativo. Questa libreria consente di configurare gli indirizzi dei server DNS utilizzati per la risoluzione dei domini. La soluzione proposta imposta come unico indirizzo valido l'IP 127.0.0.1, dove opera il server gestito da ClientShield.

StubbornDnsServer estende i metodi *start* e *stop* dell'implementazione di base per gestire correttamente la configurazione del DNS di sistema. All'avvio del server locale, `StubbornDnsServer` utilizza l'evento *NetworkChange*, appartenente alle librerie di sistema, per monitorare ogni cambiamento alle configurazioni di rete, assicurandosi costantemente che il server DNS rimanga impostato correttamente. All'arresto, la gestione dell'evento viene disattivata e il DNS viene ripristinato alla modalità dinamica, garantendo che il sistema torni al suo stato originale.

L'implementazione del server DNS di base segue uno sviluppo molto modulare sfruttando vari componenti che ne consento la facile estensione e modifica. Non solo, lo sviluppo modulare consente di suddividere i compiti da svolgere in routine eseguibili da thread in concorrenza, rendendo la risoluzione dei domini più efficiente. Il server necessita di due componenti fondamentali:

- **DnsResolver:** interfaccia di risoluzione delle richieste DNS.
- **SocketListener:** interfaccia di gestione di un socket, consente di attendere richieste su una porta ed eventualmente rispondere.

Tali componenti, oltre a garantire flessibilità al sistema, contribuiscono anche a mantenere il codice della classe `DnsServer` più semplice e, dunque, facile da manutenere.

Al suo avvio il server utilizza la classe `TaskFactory`, appartenente alle librerie di sistema, per schedulare l'esecuzione parallela dei task di ascolto e risoluzione delle richieste DNS. Queste routine sono svolte per un periodo di tempo indefinito, infatti, fintanto che il servizio non riceve una richiesta di stop dal server named pipe o il computer non viene spento, il server deve procedere con la sua esecuzione. Per questo motivo entrambi i task sono creati specificando alla `TaskFactory` il parametro *LongRunning*, ciò assicura che entrambi abbiano dei thread dedicati, prevenendo l'uso di quelli appartenenti al ThreadPool, riservati ai task di breve durata. Per gestire in maniera ottimale la concorrenza causata dai task, il server mantiene una coda di messaggi bloccante. La coda consente a `SocketListener` e `DnsResolver` di cooperare tra loro in modo sicuro, evitando race conditions.

La routine di ascolto delle richieste è completamente affidata alla classe `UdpListener`, un'implementazione dell'interfaccia `SocketListener` che utilizza il client UDP delle librerie di sistema per monitorare la porta 53, dedicata alle comunicazioni DNS. Infatti, come riportato in [Moc87]: "I messaggi inviati utilizzando UDP usano la porta server 53 (decimale). [...] UDP non è accettabile per i trasferimenti di zona, ma è il metodo raccomandato per le query standard su Internet.".

Alla ricezione di una richiesta DNS, `UdpListener` la aggiunge alla coda di messaggi. Ciò consente di rendere il thread di ascolto sempre reattivo, in quanto

viene scaricato di qualsiasi lavoro di gestione della richiesta, minimizzando i tempi di reazione alla ricezione.

La routine di risoluzione del dominio fa uso della classe `DoHResolver`, implementazione dell'interfaccia `DnsResolver` che utilizza il client HTTP delle librerie di sistema per inviare richieste di risoluzione DNS ai server FlashStart. La classe si occupa di assicurarsi che venga utilizzata la versione 2 di HTTP, garantendo la crittografia della comunicazione. La classe espone un metodo per la risoluzione dei domini: la richiesta viene redirezionata ai server FlashStart fornendo il client ID, che verrà utilizzato per l'applicazione del filtro. La risposta ottenuta può contenere l'indirizzo IP corrispondente al dominio richiesto o, in caso di blocco, una stop page indicante il motivo della restrizione.

La risoluzione del dominio da parte del server prevede tre fasi:

1. Prima essere affidata al resolver, il nome del dominio viene estrapolato dalla richiesta, permettendo l'eventuale esclusione di domini specifici dal filtro. A tal scopo viene utilizzata la libreria ARSoft.Tools.Net[8], che offre efficaci strumenti per una decodificazione veloce di richieste DNS.

2. Decodificato il dominio, se appartenente alla lista dei domini esclusi dal filtro, la richiesta viene risolta utilizzando un server DNS pubblico, altrimenti viene utilizzato `DoHResolver` per la risoluzione.

3. Una volta ottenuta la risposta dal server DNS esterno, viene inviata al processo che aveva richiesto la risoluzione.

Infine, all'arresto, il server si occupa di interrompere correttamente l'esecuzione parallela dei task.

[8]Sito ufficiale ARSoft.Tools.Net: `https://github.com/alexreinert/ARSoft.Tools.Net`

Bibliografia

[DNS25] DNSPerf. Classifica dei servizi dns più veloci al mondo - febbraio 2025. Technical report, 2025.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Reading, MA, USA, 1994.

[Mic21] Microsoft. Microsoft docs - interprocess communications, 2021.

[Mic24] Microsoft. Microsoft docs - how to: Create windows services, 2024.

[Moc87] P. Mockapetris. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. Request for Comments 1035, RFC Editor, 1987.

[PH18] Patrick McManus e Eric Rescorla Paul Hoffman. Dns queries over https (doh). Request for Comments 8484, Internet Engineering Task Force, Ottobre 2018.

[plDA20] Agenzia per l'Italia Digitale (AgID). *Vademecum per lavorare online in sicurezza*. Agenzia per l'Italia Digitale, 2020. 11 raccomandazioni per la sicurezza nello smart working.

[plSI24] Clusit Associazione Italiana per la Sicurezza Informatica. Rapporto clusit 2024 sulla sicurezza ict in italia - aggiornamento ottobre 2024. Rapporto di ricerca CLUSIT-2024-10, Clusit, Ottobre 2024.

[Ram10] Chris Ramsdale. *Building MVP apps: MVP Part I*, March 2010.

Riconoscimenti

Optional. Max 1 page.