



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA
E DELL'AUTOMAZIONE

Software Security and Blockchain

Sviluppo di un'applicazione per la gestione di dati sensibili in
ambito sanitario

Autori:

Beccerica Sara

Attili Loris

Anno Accademico 2023/2024

Contents

1	Valutazione del rischio	4
1.1	Analisi dei requisiti	4
1.2	Identificazione del rischio	5
1.2.1	Identificazione degli asset	5
1.3	Analisi del rischio	6
1.4	Scomposizione del rischio	7
1.4.1	Valutazione dell'attacco	8
1.5	Riduzione del rischio	26
2	Design sicuro	28
2.1	Architettura	28
2.1.1	Database	28
2.1.2	Client	29
2.1.3	Blockchain	29
2.2	Architettura distribuita ridondante e diversificata	29
2.2.1	Architettura distribuita	30
2.2.2	Architettura ridondante	30
2.2.3	Architettura diversificata	30
2.3	Monitoraggio isolamento e offuscamento	31
2.3.1	Monitoraggio	31
2.3.2	Isolamento	32
2.3.3	Offuscamento	35
2.4	Sicurezza del design	35
2.4.1	Principi di design OWASP	36
2.4.2	Linee guida di Sommerville	37
2.4.3	Linee guida di Saltzer & Schroeder	38
2.4.4	Scelte tecnologiche	39
2.4.5	Python	39
2.4.6	MySQL	39
2.4.7	Ganache	40
2.4.8	Docker	40
2.4.9	Solidity	40
2.4.10	GitHub	41
3	Programmazione sicura	42
3.1	Python	42
3.2	Solidity	44
3.3	User's action flow	45
3.3.1	Login	45
3.3.2	Registrazione	46
3.3.3	Home del paziente	46
3.3.4	Cartella clinica del paziente	47
3.3.5	Conferma cure	48
3.4	Modifica profilo	48

3.4.1	Home amministratore	49
3.4.2	Registrazione di un dottore	49
3.4.3	Registrazione di un caregiver	50
3.4.4	Home dottore	50
3.4.5	Profilo dottore	51
3.4.6	Fascicolo del paziente	51
3.4.7	Aggiornamento storia clinica	52
3.4.8	Modifica delle prescrizioni	52
3.4.9	Aggiornamento dei trattamenti	53
3.4.10	Home caregiver	53
3.4.11	Profilo caregiver	53
3.4.12	Fascicolo del paziente	54
3.4.13	Inserimento di una nuova nota	55
3.5	Test	56
3.5.1	Test statico	56
3.5.2	Test dinamico	57

List of Figures

1	Prima versione del diagramma i*	4
2	Versione finale del diagramma i*	5
3	Asset individuati	7
4	Prima parte tabella stride	8
5	Fascicolo paziente - abuse case <i>Intercettazione dei dati del fascicolo</i>	9
6	Fascicolo paziente - misuse case <i>Aggiornamento errato del fascicolo del paziente</i>	10
7	Fascicolo paziente - misuse case <i>Recapito errato della prescrizione medica</i>	11
8	Fascicolo paziente - misuse case <i>Condivisione imprevista delle informazioni</i>	12
9	Monitoraggio delle attività - abuse case <i>Manipolazione del monitoraggio delle attività</i>	13
10	Backup dei dati - abuse case <i>Manipolazione dei file di backup</i>	14
11	privilegi utente - abuse case <i>Manipolazione dei privilegi utente</i>	14
12	Autenticazione - abuse case <i>Addescamento paziente</i>	15
13	Autenticazione - abuse case <i>Compromissione autenticazione</i>	16
14	Autenticazione - misuse case <i>Utilizzo di credenziali di default</i>	17
15	Autorizzazione - abuse case <i>Compromissione autorizzazione</i>	18
16	Conferma paziente - misuse case <i>Mancanza conferma paziente</i>	19
17	Abuse e misuse case in i*	20
18	Attack tree attaccante esterno	21
19	Attack tree di un utente maldestro	22
20	Attack tree medico maldestro	23
21	Attack tree utente malintenzionato	24
22	Attack tree ex utente	24

23	Attack tree paziente maldestro	25
24	Terza parte della tabella stride	26
25	Quarta parte della tabella stride	27
26	Schematizzazione dell'architettura	28
27	Funzione di ascolto degli eventi	32
28	Funzione di monitoraggio delle risorse hardware utilizzate	32
29	Funzione di registrazione delle operazioni effettuate	32
30	Un esempio di servizio presente nel file docker-compose.yml, in questo caso ganache	34
31	Contenuto del dockerfile	34
32	Contenuto del file dei requisiti	35
33	Codice software che implementa la tecnica dei canarini	35
34	Funzione di sanificazione dell'input	42
35	Esempio di gestione delle eccezioni	43
36	Esempio di utilizzo di un timeout in corrispondenza della chiamata ad un servizio esterno	44
37	Funzione di sanificazione	45
38	Schermata di login	46
39	Schermata di registrazione per i pazienti	46
40	Schermata home del paziente autenticato	47
41	Finestra di visualizzazione della cartella clinica per il paziente	48
42	Finestra di inserimento del codice fiscale del caregiver che ha somministrato le cure al paziente	48
43	Finestra di visualizzazione del profilo del paziente	49
44	Finestra home dell'amministratore	49
45	Finestra di registrazione di un dottore	50
46	Finestra di registrazione di un caregiver	50
47	Finestra home del dottore	51
48	Finestra profilo del dottore	51
49	Finestra di visualizzazione del fascicolo di un paziente da parte del dottore	52
50	Finestra di aggiornamento della storia clinica di un paziente	52
51	Finestra di aggiornamento delle prescrizioni di un paziente	52
52	Finestra di aggiornamento dei trattamenti di un paziente	53
53	Finestra della home per il caregiver	53
54	Finestra del profilo del caregiver	54
55	Finestra fascicolo di un paziente	54
56	Notifica di corretta elaborazione della conferma	55
57	Finestra di inserimento di una nuova nota	55
58	Risultati del test sui contratti	58

1 Valutazione del rischio

1.1 Analisi dei requisiti

L'analisi dei requisiti è un passaggio cruciale per comprendere e documentare le necessità funzionali e non funzionali del sistema. In questo progetto, è stato utilizzato un approccio iterativo, con diagrammi i* per modellare le relazioni tra gli attori coinvolti (dispositivo medico, pazienti, caregiver e medici) e le loro dipendenze reciproche. La modellazione ha attraversato vari sprint, affinando progressivamente il sistema, aggiungendo attori come la dispositivo medico e sistema e softgoal legati a dati medici e autenticazioni.

Dopo l'analisi dei requisiti richiesti per il progetto si è arrivati a una prima definizione del diagramma i*, che è quella che possiamo vedere in Figura 1. Successivamente si è deciso di togliere figure che non erano rilevanti al fine ultimo del progetto, come ad esempio i fornitori o la farmacia locale, e si è arrivato al diagramma finale che è quello che possiamo vedere in Figura 2. All'interno di quest'ultimo, come accennato precedentemente, sono state aggiunte le varie dipendenze e relazione tra le entità che si sono decise di tenere e sono stati aggiunti i softgoal.

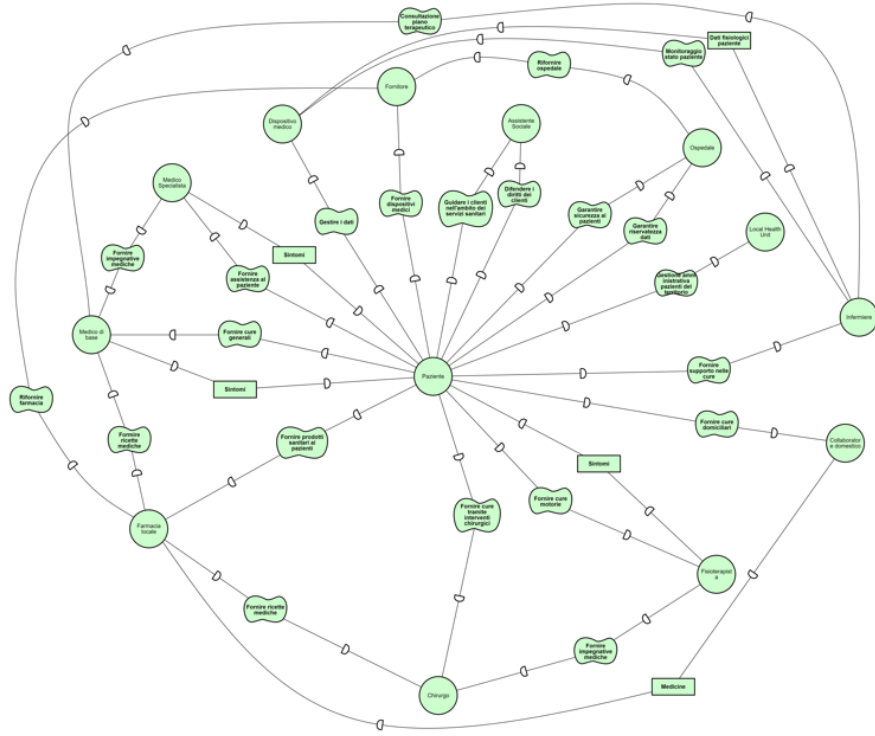


Figure 1: Prima versione del diagramma i*

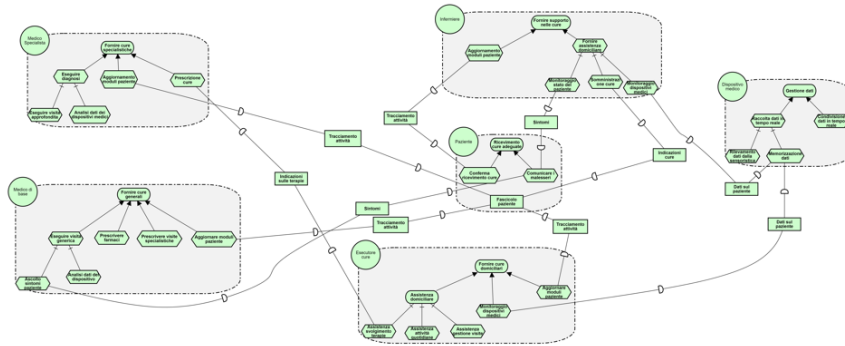


Figure 2: Versione finale del diagramma i*

1.2 Identificazione del rischio

La fase dell'identificazione del rischio è la prima fase del processo che porta alla gestione completa del rischio. Questo processo prevede quattro fasi principali:

- *identificazione del rischio;*
- *analisi del rischio;*
- *scomposizione del rischio;*
- *riduzione del rischio.*

La prima fase prevede di individuare e classificare tutti i possibili rischi a cui il sistema in questione potrebbe essere soggetto. In questa fase la prima cosa da effettuare è individuare tutti gli asset, cioè le risorse più importanti e più sensibili per il sistema.

1.2.1 Identificazione degli asset

Gli asset che sono stati identificati nel nostro progetto sono i seguenti:

- *fascicolo paziente:* questo asset riguarda i dati sullo stato del paziente, le terapie a lui assegnate e le sue informazioni personali; gli obbiettivi da rispettare sono: Integrità, Confidenzialità, Disponibilità, Affidabilità, Resilienza;
- *monitoraggio delle attività:* questo asset riguarda un'attività che consiste nel verificare che l'operato dei vari attori rispetti le policy del settore; gli obbiettivi da rispettare sono: Responsabilità, Disponibilità, Affidabilità;
- *backup dei dati:* questo asset riguarda la risorsa che contiene una copia dei dati sulle attività e sui pazienti; gli obbiettivi da rispettare sono: Integrità, Confidenzialità, Affidabilità, Resilienza;

- *privilegi utente*: questo asset riguarda la risorsa che definisce i protocolli di autorizzazione degli utenti; gli obiettivi da rispettare sono: Integrità, Confidenzialità;
- *autenticazione*: questo asset riguarda il processo da effettuare per poter accedere alle varie funzionalità; gli obiettivi da rispettare sono: Autenticazione, Disponibilità, Affidabilità;
- *autorizzazione*: questo asset riguarda il processo che verifica se un utente ha i privilegi necessari per poter accedere ad una funzionalità; gli obiettivi da rispettare sono: Autorizzazione, Responsabilità, Confidenzialità, Affidabilità;
- *conferma paziente*: questo asset riguarda l'attività con cui il paziente certifica il ricevimento delle cure adeguate; gli obiettivi da rispettare sono: Responsabilità, Affidabilità.

1.3 Analisi del rischio

L'analisi del rischio è la fase successiva alla fase dell'identificazione del rischio e ha come obiettivo quello di valutare la probabilità e l'impatto che ciascun rischio individuato ha sul sistema considerato, cioè si concentra sul determinare quanto sia grave ciascun rischio e quali conseguenze potrebbero susseguire alla sua materializzazione.

Generalmente l'analisi del rischio comprende:

- *valutazione del valore degli asset*: si assegna un valore a ciascun asset sulla base dell'importanza che questo ha per il sistema;
- *valutazione all'esposizione*: si stima il danno potenziale che potrebbe derivare dalla compromissione di ciascun asset;
- *identificazione delle minacce*: in questa fase vengono valutati i diversi tipi di minacce che potrebbero influenzare i vari asset, le categorie di minacce utilizzate nel diagramma dual-stride utilizzato sono:
 - **spoofing**,
 - **tampering**,
 - **repudation**,
 - **information disclosure**,
 - **denial of service**,
 - **elevation of privilege**,
 - **danger**,
 - **unreliability**,
 - **absence of resilience**;

- *probabilità di materializzazione del rischio*: viene stimata la probabilità che una minaccia possa effettivamente verificarsi;
- *calcolo dell'impatto*: si analizzano le conseguenze del verificarsi del rischio in termini qualitativi o quantitativi;
- *classificazione del rischio*: dopo aver valutato la probabilità e l'impatto i rischi vengono classificati in base alla loro gravità determinando così quali sono i più critici che richiedono azioni immediate di mitigazione;
- *decisione di contromisure*: in base alla criticità del rischio si individuano le contromisure da adottare per mitigare o eliminare il rischio.

Nella Figura 3 possiamo vedere gli asset individuati. Inizialmente compiliamo solo le colonne iniziali della tabella DUAL-STRIDE per poi completarlo una volta effettuata la scomposizione del rischio. Nella Figura4 possiamo vedere la prima parte dell'analisi degli asset individuati.

Asset	Value	Objective
Fascicolo paziente	E' costituito dai dati sullo stato del paziente, le terapie a lui assegnate e le sue informazioni personali.	Integrità, Confidenzialità, Disponibilità, Affidabilità, Resilienza
Monitoraggio delle attività	E' un'attività che consiste nel verificare che l'operato dei vari attori rispetti le policy del settore.	Responsabilità, Disponibilità, Affidabilità
Backup dei dati	E' la risorsa che contiene una copia dei dati sulle attività e sui pazienti.	Integrità, Confidenzialità, Affidabilità, Resilienza
Privilegi utente	E' la risorsa che definisce i protocolli di autorizzazione degli utenti.	Integrità, Confidenzialità
Autenticazione	E' il processo da effettuare per poter accedere alle varie funzionalità.	Autenticazione, Disponibilità, Affidabilità
Autorizzazione	E' il processo che verifica se un utente ha i privilegi necessari per poter accedere ad una funzionalità.	Autorizzazione, Responsabilità, Confidenzialità, Affidabilità
Conferma paziente	E' l'attività con cui il paziente certifica il ricevimento delle cure adeguate.	Responsabilità, Affidabilità

Figure 3: Asset individuati

1.4 Scomposizione del rischio

La scomposizione del rischio è la terza fase della gestione del rischio e consiste nel suddividere un rischio complessivo in componenti più piccole e specifiche, per analizzarle in dettaglio e comprendere meglio la natura del rischio. Questa fase aiuta a identificare le cause sottostanti e i fattori che contribuiscono al rischio, facilitando così lo sviluppo di misure di mitigazione più mirate ed efficaci.

Asset	Value	Spoofing	Tampering	Repudiation	Information disclosure	DOS	Elevation of privilege	Danger	Unreliability	Absence of Resilience	Exposure
Fascicolo paziente	150.000-200.000€		X		X	X			X	X	Impatto elevato 50.000€
Monitoraggio delle attività	130.000-150.000€			X		X			X		Impatto notevole 30.000€
Backup dei dati	100.000-130.000€		X		X				X	X	Impatto elevato 50.000€
Privilegi utente	100.000-150.000€		X		X						Impatto elevato 60.000€
Autenticazione	200.000€-220.000€	X				X			X		Impatto molto elevato
Autorizzazione	180.000-200.000€			X	X		X		X		Impatto elevato 60.000€
Conferma paziente	50.000-80.000€			X					X		Impatto contenuto 15.000€

Figure 4: Prima parte tabella stride

1.4.1 Valutazione dell'attacco

La valutazione dell'attacco è una fase della gestione del rischio in cui si valutano i potenziali attacchi informatici che un sistema potrebbe subire. L'obiettivo principale è comprendere in che modo un attacco potrebbe concretizzarsi, quali vulnerabilità potrebbe sfruttare, e quali conseguenze potrebbe avere sul sistema e sui dati.

In questa fase si individuano vari scenari possibili che vengono classificati in **abuse case**, che sono gli scenari comprendenti azioni svolte in modo intenzionale, e **misuse case** che sono gli scenari comprendenti azioni svolte in modo accidentale. Questi casi sono stati descritti tramite delle tabelle che sono gli **Schemi di Jacobson**.

In sintesi, questa fase permette di esaminare il sistema dal punto di vista di un potenziale attaccante, identificando i punti deboli e sviluppando strategie per prevenire o mitigare gli attacchi.

Asset fascicolo paziente

Per l'asset **fascicolo paziente** sono stati individuati vari casi. Il primo è un caso di abuso, cioè **intercettazione dei dati del fascicolo** (Figura 5), questo caso descrive la situazione in cui quando un utente richiede l'accesso a un fascicolo di un paziente un attaccante intercetta il trasferimento dei dati e ne entra in possesso. Come possiamo vedere ci sono due possibili scenari di attacco che

sono: l'attaccante, una volta aperto il file in lettura o scrittura, acquisisce le informazioni contenute nel file e può modificarle; un monitor della rete, una volta aperto il file, lo intercetta e l'attaccante può accedere a una copia del file.

Case Type	Abuse Case	Case ID	1
Case Name	Intercettazione dei dati del fascicolo		
Actors	Utente del sistema, attaccante		
Description	Un utente del sistema richiede l'accesso al fascicolo del paziente per inserire nuovi dati oppure per leggere quelli già presenti. Un attaccante intercetta il trasferimento dei dati entrandone in possesso (CAPEC-94).		
Data	Fascicolo del paziente.		
Stimulus and preconditions	L'utente è autenticato. L'utente è autorizzato. L'utente sceglie il file da aprire.		
Attack Flow 1	1. L'attaccante accede ai file in lettura o scrittura. 2. L'attaccante acquisisce le informazioni del paziente e può modificare i file.		
Attack Flow 2	1. L'utente apre il file scelto 2. Un monitor della rete intercetta il file inviato dal sistema all'utente e l'attaccante può accedere ad una copia del file.		
Response and Postconditions	L'attaccante ha ottenuto le informazioni contenute nel file trasferito.		
Non Functional Requirements			
Mitigations	I dati scambiati nelle comunicazioni devono essere criptati. L'accesso ai file deve essere concesso solo dopo verifica dell'identità dell'utente e dell'autorizzazione.		
Comments			

Figure 5: Fascicolo paziente - abuse case *Intercettazione dei dati del fascicolo*

Il caso di misuso **aggiornamento errato del fascicolo del paziente** (Figura 6), descrive la situazione in cui quando qualcuno aggiorna un fascicolo di un paziente lo fa in modo errato e quindi i dati contenuti nel fascicolo sono compromessi.

Case Type	Misuse Case	Case ID	1
Case Name	Aggiornamento errato del fascicolo del paziente		
Actors	Medico di base, medico specialista, infermiera, esecutore		
Description	Un attore quando aggiorna il fascicolo di un paziente immette per sbaglio dei dati errati e lo modifica in modo irreversibile.		
Data	Fascicolo paziente		
Stimulus and preconditions	Un attore immette i dati errati.		
Attack Flow 1	Un attore immette dei dati pensando che siano corretti. I dati immessi quindi non sono più modificabili.		
Response and Postconditions	I dati contenuti nel fascicolo sono compromessi e non sono più affidabili.		
Non Functional Requirements			
Mitigations	Per evitare che questo accada si può aggiungere una doppia conferma al salvataggio delle aggiunte effettuate, inoltre andrà introdotta una funzionalità per inserire correzioni tenendo comunque traccia delle varie versioni del fascicolo.		
Comments			

Figure 6: Fascicolo paziente - misuse case *Aggiornamento errato del fascicolo del paziente*

Il caso di misuse **recapito errato della prescrizione medica** (Figura 7), descrive la situazione in cui un medico che ha fatto la prescrizione a un paziente la manda poi al paziente errato.

Case Type	Misuse Case	Case ID	2
Case Name	Recapito errato della prescrizione medica		
Actors	Medico di base, medico specialista		
Description	Un attore, quando deve consegnare una prescrizione medica ad un paziente, la manda al paziente sbagliato.		
Data	Prescrizione medica		
Stimulus and preconditions	Un attore immette il codice identificativo errato.		
Attack Flow 1	Un attore immette il codice errato e invia la prescrizione medica.		
Response and Postconditions	I dati sensibili dei due pazienti sono compromessi.		
Non Functional Requirements			
Mitigations	Per evitare che questo accada si può aggiungere un tasto di conferma quando l'attore immette il codice di un determinato paziente.		
Comments			

Figure 7: Fascicolo paziente - misuse case *Recapito errato della prescrizione medica*

Il terzo caso di misuse è **condivisione imprevista delle informazioni** (Figura 8), descrive la situazione in cui un attore cattura un'immagine o trasporta del testo nel suo computer non intenzionalmente, e quindi renderebbe accessibili delle informazioni sensibili ai malintenzionati.

Case Type	Misuse Case	Case ID	3
Case Name	Condivisione imprevista delle informazioni		
Actors	Medico di base, medico specialista, infermiera, esecutore cure		
Description	Un attore, catturando un'immagine o trasportando il testo di un file all'esterno del software, potrebbe involontariamente rendere accessibili ai malintenzionati le informazioni contenute nel fascicolo di un paziente.		
Data	Fascicolo del paziente		
Stimulus and preconditions	L'attore è autenticato.		
Attack Flow 1	1. L'attore ha aperto un file contenente informazioni riservate del paziente 2. L'attore trasferisce il contenuto del file all'esterno dell'applicazione, ad esempio fotografandolo o copiando e incollando il testo al suo interno altrove.		
Response and Postconditions	I dati sono al di fuori del software e non possono più essere protetti o controllati, sono dunque potenzialmente accessibili da attaccanti o malintenzionati.		
Non Functional Requirements			
Mitigations	Non deve essere possibile eseguire l'operazione "copia e incolla" con i dati dei pazienti, inoltre gli utenti devono essere informati dei rischi relativi a questo utilizzo improprio del software.		
Comments			

Figure 8: Fascicolo paziente - misuse case *Condivisione imprevista delle informazioni*

Asset monitoraggio delle attività

Per l'asset **monitoraggio delle attività** è stato individuato un caso di abuso, cioè **manipolazione del monitoraggio delle attività** (Figura 9), che descrive la situazione in cui un attaccante prende possesso dei file di log e ne modifica il contenuto rendendoli così non più integri.

Case Type	Abuse Case	Case ID	2			
Case Name	Manipolazione del monitoraggio delle attività					
Actors	Attaccante					
Description	Un attaccante con accesso ai file di log li modifica compromettendo la loro integrità.					
Data	Log delle attività					
Stimulus and preconditions	L'attaccante ha accesso ai file di log					
Attack flow 1	L'attaccante modifica i file di log per coprire le tracce di attività non conformi alle policy, oppure indurre in errore un controllo di tali file.					
Response and Postconditions	I file di log non rispettano più le proprietà di integrità e affidabilità.					
Non Functional Requirements						
Mitigation	L'input inserito dagli utenti deve essere sanificato. L'accesso ai file di log deve essere concesso solo dopo verifica dell'identità dell'utente e dell'autorizzazione.					
Comments						

Figure 9: Monitoraggio delle attività - abuse case *Manipolazione del monitoraggio delle attività*

Asset backup dei dati

Per l'asset **backup dei dati** è stato individuato un caso di abuso, cioè **manipolazione dei file di backup** (Figura 10); questo caso di abuso descrive una situazione in cui l'attaccante ottiene l'accesso ai file di backup e li modifica compromettendo il loro contenuto.

Case Type	Abuse Case	Case ID	3
Case Name	Manipolazione dei file di backup		
Actors	Attaccante		
Description	Un attaccante con accesso ai file di backup li modifica compromettendo la loro integrità.		
Data	Backup dei dati		
Stimulus and preconditions	L'attaccante ha accesso ai file di backup		
Attack Flow 1	1. L'attaccante modifica i file di backup; 2. Quando viene attivata la funzione di ripristino il sistema ripristina dati errati.		
Response and Postconditions	I dati di backup e il relativo servizio sono inutilizzabili, compromettendo la resilienza del sistema.		
Non Functional Requirements			
Mitigation	Il sistema deve verificare l'integrità di un file prima di processarlo. Le informazioni contenute all'interno dei file di backup devono essere crittografate.		
Comments			

Figure 10: Backup dei dati - abuse case *Manipolazione dei file di backup*

Asset privilegi utente

Per l'asset **privilegi utente** è stato individuato il caso di abuso **manipolazione dei privilegi utente** (Figura 11), che si verifica in caso un attaccante modifica i privilegi utente ottenendo così accesso a delle funzionalità che non dovrebbe avere.

Case Type	Abuse Case	Case ID	4
Case Name	Manipolazione dei privilegi utente		
Actors	Attaccante		
Description	Un attaccante con accesso ai file che descrivono i privilegi degli utenti li modifica compromettendo la loro integrità e ottenendo accesso a delle funzionalità che non dovrebbe usare.		
Data	Privilegi utente		
Stimulus and preconditions	L'attaccante deve avere accesso ai file che definiscono i privilegi degli utenti.		
Attack Flow 1	L'attaccante modifica i file che descrivono i privilegi degli utenti ottenendo privilegi che non dovrebbe avere o togliendoli ad altri utenti.		
Response and Postconditions	Il sistema di gestione delle autorizzazioni non funziona più correttamente e viene meno la sua reliability.		
Non Functional Requirements			
Mitigation	Le informazioni contenute all'interno dei file di backup devono essere crittografate		
Comments			

Figure 11: privilegi utente - abuse case *Manipolazione dei privilegi utente*

Asset autenticazione

Per l'asset **autenticazione** sono stati individuati due casi di abuso e un caso di misuse.

Il caso di abuso **adescamento paziente** (Figura 12), descrive la situazione in cui il paziente riceve un messaggio o un'e-mail che pensa essere del suo medico e inserisce i propri dati sensibili, così l'attaccante ha accesso ai dati del paziente.

Case Type	Abuse Case	Case ID	5			
Case Name	Addescamento paziente					
Actors	Paziente, attaccante					
Description	Il paziente riceve una e-mail oppure un messaggio che sembra essere del suo medico (o dall'ospedale) la apre e inserisce i suoi dati sensibili. L'attaccante quindi ha accesso ai dati inseriti.					
Data	Dati sensibili del paziente					
Stimulus and preconditions	Il paziente apre un'e-mail o il messaggio malevoli e inserisce i suoi dati (ad esempio le sue credenziali).					
Attack Flow 1	L'attaccante ha accesso ai dati che il paziente ha inserito.					
Response and Postconditions						
Non Functional Requirements						
Mitigation	Con un'adeguata formazione del paziente si evita che esso apra i messaggi malevoli e esponga i suoi dati sensibili.					
Comments						

Figure 12: Autenticazione - abuse case *Addescamento paziente*

Il secondo caso di abuso è **compromissione autenticazione** (Figura 13), in cui l'attaccante riesce ad eludere i meccanismi di autenticazione del sistema e riesce ad accedere ai dati sensibili dell'utente.

Case Type	Abuse Case	Case ID	6			
Case Name	Compromissione autenticazione					
Actors	Attaccante					
Description	La compromissione dell'autenticazione avviene quando un attaccante riesce ad eludere i meccanismi di autenticazione del sistema. Una volta dentro l'attaccante può avere accesso ai dati sensibili e ai servizi di quell'utente.					
Data	Dati sensibili.					
Attack Flow 1	<p>L'attaccante cerca di intercettare i dati di sessione, che possono essere contenuti in cookie, URL o altri mezzi di trasmissione dati.</p> <p>Si possono utilizzare diverse tecniche per intercettare le sessioni, inclusi attacchi di tipo "Man-in-the-Middle", in cui l'attaccante monitora il traffico tra l'utente e il server.</p> <p>Una volta ottenute le informazioni di sessione, l'attaccante può utilizzarle per impersonare l'utente e accedere alle risorse o ai servizi protetti da autenticazione senza la necessità di conoscere le credenziali di accesso originali.</p>					
Attack Flow 2	<p>L'attaccante esegue ripetutamente tentativi di accesso con varie combinazioni di username e password, partendo da un elenco predefinito o generando tutte le possibili combinazioni.</p>					
Response and Postconditions						
Non Functional Requirements						
Mitigation	Entrambi gli attacchi possono essere sventati utilizzando tecniche di crittografia avanzata.					
Comments						

Figure 13: Autenticazione - abuse case *Compromissione autenticazione*

Il caso di misuse **utilizzo di credenziali di default** (Figura 14) avviene quando un utente utilizza delle credenziali semplici da individuare rendendo così facile acquisire il suo profilo.

Case Type	Misuse Case	Case ID	5			
Case Name	Utilizzo di credenziali di default					
Actors	Medico di base, medico specialista, infermiera, esecutore cure.					
Description	Un utente del sistema utilizza delle credenziali di default o particolarmente banali.					
Data	Fascicolo del paziente, account utente					
Attack Flow 1	L'utente non modifica le credenziali di default, oppure lo fa sostituendole con credenziali banali.					
Response and Postconditions	Il profilo dell'utente, e quindi il software, diventa particolarmente vulnerabile ad attacchi di tipo brute force.					
Non Functional Requirements						
Mitigations	Il sistema, durante la procedura di definizione delle credenziali, deve accettarle solo se hanno una complessità minima prestabilita.					
Comments						

Figure 14: Autenticazione - misuse case *Utilizzo di credenziali di default*

Asset autorizzazione

Per l'asset **autorizzazione** è stato individuato il caso di abuso **compromissione autorizzazione** (Figura 15), questo succede quando l'attaccante riesce ad ottenere accesso non autorizzato a risorse o funzionalità del sistema superando le restrizioni presenti.

Case Type	Abuse Case	Case ID	7			
Case Name	Compromissione autorizzazione					
Actors	Attaccante					
Description	La compromissione dell'autorizzazione avviene quando l'attaccante riesce ad ottenere privilegi o accesso non autorizzato a risorse o funzionalità di un sistema, superando le restrizioni di autorizzazione stabilite					
Data	Dati sensibili del paziente					
Attack Flow 1	Gli attaccanti cercano vulnerabilità nelle funzionalità di inclusione di file di un'applicazione. Successivamente, sfruttando la vulnerabilità identificata, inseriscono URL o percorsi di file che contengono codice malevolo all'interno delle richieste di inclusione di file dell'applicazione. Se l'attacco ha successo, il codice malevolo incluso viene eseguito nel contesto dell'applicazione, permettendo così agli attaccanti di ottenere il controllo del sistema ed eseguire comandi senza averne realmente i privilegi.					
Attack Flow 2	Gli attaccanti potrebbero analizzare i modelli di generazione delle credenziali, la durata delle sessioni o altri parametri correlati per dedurre le credenziali di sessione attese. Una volta dedotte le credenziali di sessione previste, l'attaccante potrebbe utilizzarle per ottenere accesso ad autorizzazioni di sistema che non gli spettano.					
Response and Postconditions						
Non Functional Requirements						
Mitigation	Il primo attacco può essere sventato con l'utilizzo di librerie controllate così che l'attaccante non abbia la possibilità di inserire codice all'interno i file di sistema. Il secondo può essere sventato con l'utilizzo di Firewall che monitorano i tentativi di accesso a un sistema.					
Comments						

Figure 15: Autorizzazione - abuse case *Compromissione autorizzazione*

Asset conferma paziente

Infine, per l'asset **conferma paziente** è stato individuato un caso di misuse, cioè **mancanza conferma paziente** (Figura 16), questo caso si verifica quando il paziente non inserisce la conferma di aver ricevuto le cure e quindi viene meno la proprietà del non ripudio.

Case Type	Misuse Case	Case ID	4
Case Name	Mancanza conferma paziente		
Actors	Paziente		
Description	Il paziente non inserisce la conferma di ricevimento delle cure previste.		
Data	Prescrizione medica		
Stimulus and preconditions	Il paziente deve confermare il ricevimento delle cure.		
Attack Flow 1	Il paziente non inserisce all'interno del sistema la sua conferma.		
Response and Postconditions	Non avendo accertato che il paziente abbia ricevuto le cure previste, oppure che abbia letto delle notifiche inviate attraverso il software, in caso di problemi non è possibile identificare le cause; viene quindi meno la proprietà del non ripudio.		
Non Functional Requirements			
Mitigations	Il paziente non deve poter utilizzare alcuna funzionalità del software finché non conferma di aver ricevuto le cure previste.		
Comments			

Figure 16: Conferma paziente - misuse case *Mancanza conferma paziente*

La valutazione dell'attacco procede con l'analisi dei rischi attraverso l'utilizzo degli attack tree, strumenti che permettono di rappresentare in modo strutturato i diversi scenari di attacco. Questi alberi consentono di visualizzare i passaggi che un potenziale aggressore potrebbe seguire per portare a termine un attacco.

Un attack tree si compone di una struttura gerarchica, in cui il nodo principale rappresenta l'attacco generale. Da qui si diramano diversi nodi e rami secondari, ciascuno dei quali descrive le azioni e le decisioni che l'attaccante deve intraprendere per raggiungere il suo obiettivo finale.

Questo metodo facilita l'identificazione delle vulnerabilità del sistema, dei punti deboli nel design e delle possibili azioni malevole che potrebbero essere sfruttate. Grazie agli attack trees, diventa possibile mappare i rischi e le minacce in modo dettagliato, offrendo così una base concreta per pianificare e attuare efficaci contromisure di sicurezza.

Nella Figura 17 possiamo vedere gli abuse e misuse case nell'i*.

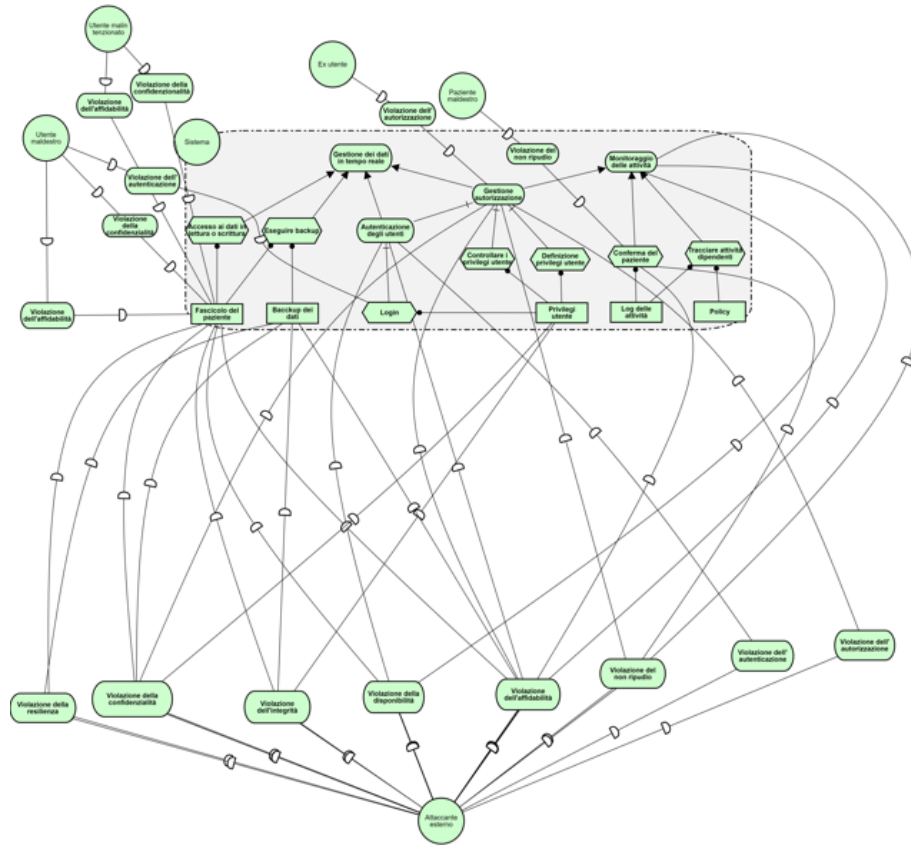


Figure 17: Abuse e misuse case in i*

Successivamente sono stati definiti i vari attack tree, uno per ogni minaccia identificata.

Attack tree attaccante esterno

Il nodo radice di questo attack tree è l'attaccante esterno, cioè il punto di partenza è un aggressore che tenta di compromettere il sistema dall'esterno. Nella Figura 18 possiamo vedere che ci sono vari possibili attacchi, in primis si può vedere che **l'autenticazione** può essere compromessa con varie tecniche:

- *monitoraggio della rete*: l'attaccante monitora il traffico di rete per intercettare credenziali o altre informazioni sensibili;
- *intercettazione dei cookie*: l'attaccante sfrutta i cookie di sessione per ottenere accesso non autorizzato;

- *attacchi brute force*: vengono effettuati tentativi ripetuti di indovinare le credenziali di autenticazione tramite prove automatizzate;
- *attività di phishing*: l'attaccante inganna gli utenti legittimi per ottenere le loro credenziali di accesso.

Invece, la **violazione della confidenzialità** deriva dalla compromissione dell'autenticazione, cioè se l'attaccante riesce a ottenere accesso illegittimo, può accedere a informazioni riservate e confidenziali.

La **violazione dell'integrità** avviene quando l'attaccante altera i dati o compromette la coerenza delle informazioni dopo aver ottenuto l'accesso non autorizzato. Inoltre, si collega alle attività di phishing, che possono portare a modifiche malevole dei dati.

La **violazione del non ripudio** deriva dalla violazione dell'integrità, cioè, l'attaccante potrebbe manipolare i dati o le azioni nel sistema in modo tale che non sia possibile dimostrare chi abbia eseguito determinate operazioni, compromettendo la possibilità di legare determinate attività alla persona che le ha eseguite.

La **violazione dell'affidabilità** deriva dalla violazione dell'integrità, cioè, i dati non possono più essere considerati affidabili se sono stati compromessi.

La **violazione dell'autorizzazione** avviene quando l'attaccante ottiene permessi o privilegi non autorizzati per accedere a risorse sensibili, una possibilità che questo avvenga è tramite l'attacco di command injection, che permette all'attaccante di eseguire comandi arbitrari su un sistema sfruttando vulnerabilità nell'autorizzazione.

La **violazione della disponibilità** avviene se l'attaccante, riuscendo ad ottenere privilegi elevati, rende il sistema indisponibile.

Infine, la **violazione della resilienza** avviene se l'attaccante riesce ad influenzare la capacità del sistema di resistere agli attacchi, compromettendo la disponibilità del servizio.

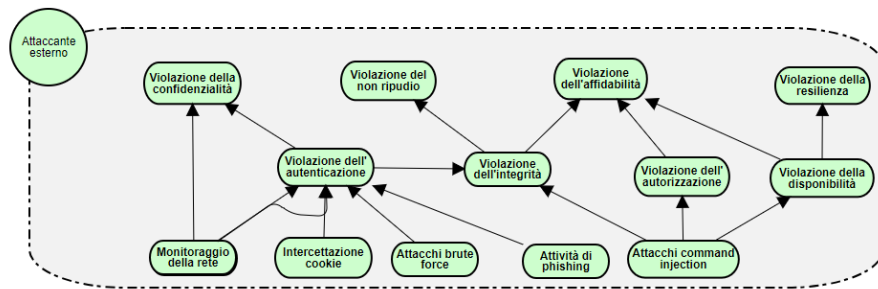


Figure 18: Attack tree attaccante esterno

Attack tree utente maldestro

Il nodo radice di questo attack tree è un utente maldestro, cioè il punto di partenza è un utente che, pur non agendo intenzionalmente, compromette la sicurezza del sistema attraverso comportamenti sbagliati o sbrigativi.

Come possiamo vedere in Figura 19 le vulnerabilità che sono state individuate sono:

- **violazione dell'affidabilità:** questa violazione potrebbe avvenire tramite la modifica o la cancellazione di dei dati sensibili; un utente potrebbe erroneamente alterare i dati, causando una perdita di coerenza e compromettendo l'affidabilità del sistema;
- **violazione dell'integrità:** anch'essa come la violazione dell'affidabilità avviene tramite una modifica o cancellazione involontaria di alcuni dati sensibili;
- **violazione della confidenzialità:** questa violazione potrebbe avvenire in due modi distinti, tramite la condivisione dei dati, cioè l'utente potrebbe utilizzare canali di condivisione non sicuri o non idonei, esponendo dati sensibili; oppure tramite il trasporto dei dati all'esterno del software, cioè dati protetti potrebbero essere salvati in luoghi non sicuri o non regolamentati, aumentando il rischio di esposizione.
- **violazione dell'autenticazione:** questa violazione potrebbe avvenire tramite la scelta di password deboli e facilmente intuibili (ad esempio, password come "12345" o "password"), questo riduce drasticamente la sicurezza del sistema.

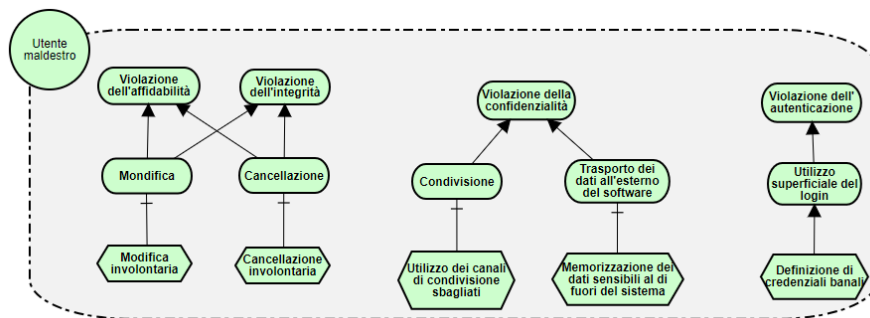


Figure 19: Attack tree di un utente maldestro

Attack tree medico maldestro

Il nodo radice di questo attack tree è un medico maldestro, cioè il punto di partenza è un medico che, senza malizia, commette errori che portano a vulnerabilità nel sistema.

Come possiamo vedere in Figura 20 le vulnerabilità che sono state individuate sono: la **violazione della confidenzialità** e la **violazione dell'integrità**, entrambe queste violazioni avvengono quando un medico, dopo aver creato un file contenente informazioni riservate, lo assegna al paziente errato creando così una situazione in cui i dati del paziente potrebbero essere visualizzati da personale che non ha il diritto di accedervi, inoltre, le informazioni errate dovendo essere attribuite a un paziente diverso, alterano l'accuratezza della sua documentazione clinica.

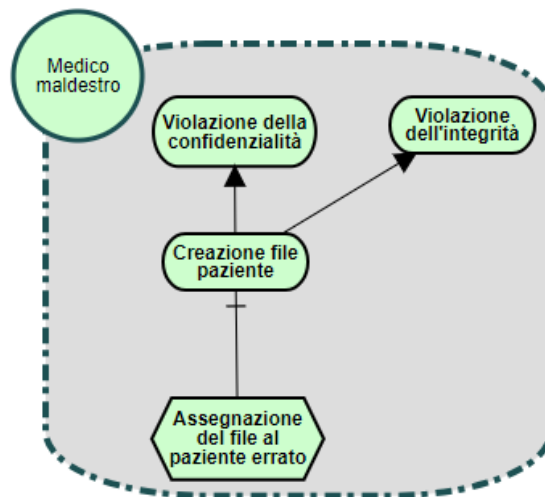


Figure 20: Attack tree medico maldestro

Attack tree utente malintenzionato

Il nodo radice di questo attack tree è un utente malintenzionato, cioè rappresenta un utente che, con intento malevolo, cerca di compromettere diversi aspetti del sistema.

Nella Figura 21 possiamo vedere due possibili vulnerabilità:

- **violazione della confidenzialità:** avviene quando l'utente estrae dati sensibili dal sistema protetto per condividerli o memorizzarli in ambienti non sicuri o non autorizzati;
- **violazione dell'affidabilità:** avviene quando l'utente modifica, inserisce o cancella volontariamente dei dati sensibili, questo comprometterebbe l'affidabilità di tutti i dati del sistema.

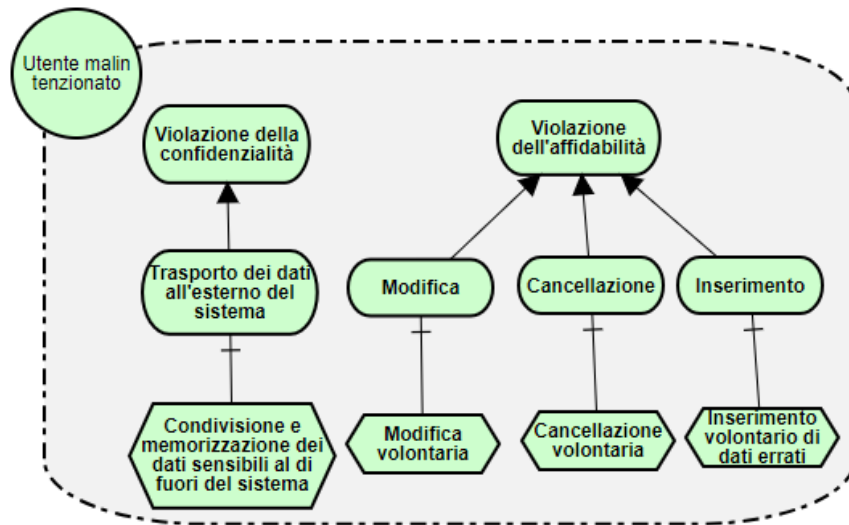


Figure 21: Attack tree utente malintenzionato

Attack tree ex utente

Il nodo radice di questo attack tree è un ex utente, cioè il punto di partenza è una persona che in passato aveva accesso al sistema, ma che ora non dovrebbe più poter eseguire operazioni all'interno di esso.

Come possiamo vedere in Figura 22 la vulnerabilità individuata è la **violazione dell'autorizzazione**: questo avviene quando un ex utente tenta di eseguire operazioni nel sistema per cui non ha più le credenziali necessarie o i permessi autorizzati; può trattarsi di tentativi di accesso a risorse o esecuzione di comandi che dovrebbero essere limitati a utenti attivi e autorizzati.

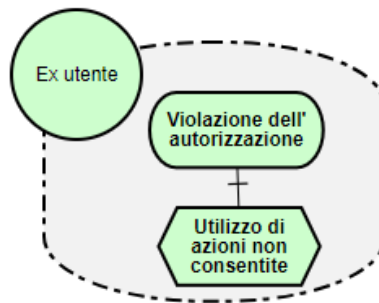


Figure 22: Attack tree ex utente

Attack tree paziente maldestro

Il nodo radice di questo attack tree è un ex utente, cioè il punto di partenza è un paziente che, involontariamente, non segue le procedure o le regole stabilite dal sistema.

Come possiamo vedere in Figura 23 la vulnerabilità individuata è la **violazione del non ripudio**: questa violazione avviene quando il paziente si dimentica di eseguire l'attività di "conferma ricevimento cure" cioè che non conferma di aver ricevuto le cure prescritte o il trattamento eseguiti dal caregiver associatogli; questo può causare problemi di responsabilità o di tracciabilità, dove il sistema non è in grado di garantire che il paziente abbia effettivamente ricevuto le cure.

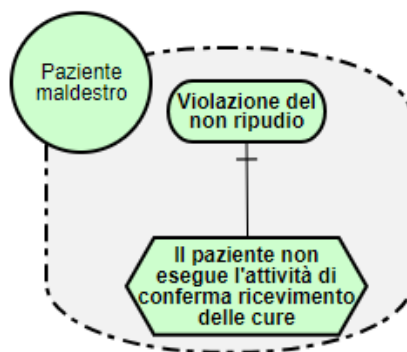


Figure 23: Attack tree paziente maldestro

Dopo le seguenti analisi è stata definita un'altra parte della tabella DUAL-STRIDE come possiamo vedere in Figura 24.

Asset	Exposure	Attack	Inherent Probability	Inherent Risk
Fascicolo paziente	Impatto elevato 50.000€	CAPEC-165 CAPEC-94	18%	9.000 €
Monitoraggio delle attività	Impatto notevole 30.000€	CAPEC-268 CAPEC-242	20% 35%	6.000€ 10.500€
Backup dei dati	Impatto elevato 50.000€	CAPEC-165	10%	5.000 €
Privilegi utente	Impatto elevato 60.000€	CAPEC-165	10%	6.000 €
Autenticazione	Impatto molto elevato	CAPEC-593 CAPEC-112	35%	24.500 €
Autorizzazione	Impatto elevato 60.000€	CAPEC-175 CAPEC-59	20% 30%	12.000€ 18.000€
Conferma paziente	Impatto contenuto 15.000€	CAPEC-98	30%	4.500 €

Figure 24: Terza parte della tabella stride

1.5 Riduzione del rischio

La riduzione del rischio si concentra sull'adozione di misure per mitigare o eliminare i rischi identificati e analizzati nelle fasi precedenti. L'obiettivo è ridurre la probabilità che un rischio si materializzi o ridurre l'impatto che esso avrebbe sul sistema.

Partendo dall'analisi dei rischi e degli attacchi associati, sono stati individuati meccanismi di controllo per limitare i danni. Non è sufficiente elencare soluzioni singole; è fondamentale anche considerare combinazioni di soluzioni per ottenere un approccio più completo e sinergico nella gestione dei rischi.

Ogni meccanismo di controllo è stato sottoposto a una valutazione dettagliata che ha considerato non solo i costi di implementazione, ma anche la fattibilità e la praticità d'uso. È stata esaminata la concreta realizzabilità di ogni soluzione e la sua sostenibilità a lungo termine. Inoltre, è stata valutata l'influenza dei nuovi meccanismi sull'usabilità del software per garantire una buona esperienza utente. Poiché la soluzione economicamente più vantaggiosa non sempre coincide con quella che minimizza maggiormente il rischio, è stato introdotto il concetto di Return of Control (RoC).

A seguito della valutazione di fattibilità, sono state chiaramente stabilite le misure di controllo da implementare. In base ai risultati ottenuti, è stata completata la tabella STRIDE come si può vedere in Figura 25.

Asset	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C	Overall Cost
Fascicolo paziente	Blockchain Crittografia	800€ 100€	Alta Alta	3% 3%	50.000 €	1.500 €	9,4	2.400 €
Monitoraggio delle attività	Input validation	500 €	Alta	0,50%	30.000 €	1.500 €	17	2.000 €
Backup dei dati	Crittografia	100 €	Alta	2%	50.000 €	1.000 €	39	1.100 €
Privilegi utente	Crittografia	100 €	Alta	2%	60.000 €	1.200 €	47	1.200 €
Autenticazione	Crittografia	1.000 €	Alta	2%	70.000 €	1.400 €	22,5	2.400 €
Autorizzazione	Librerie controllate, Firewall	1.000 €	Alta	1%	60.000 €	600 €	16,4	1.600 €
Conferma paziente	Formazione	500 €	Alta	1%	15.000 €	150 €	7,7	650 €

Figure 25: Quarta parte della tabella stride

2 Design sicuro

La progettazione è un aspetto cruciale nello sviluppo di un software sicuro. Un'architettura ben progettata consente di identificare e mitigare le vulnerabilità prima che il codice venga scritto, riducendo i rischi di attacchi informatici. Durante questa fase, si stabiliscono le linee guida per l'uso di tecnologie sicure, la gestione dei dati sensibili e i controlli di accesso. Inoltre, analizzare i componenti dell'architettura prima di implementarla, consente di verificare che essi rispettino i requisiti o gli standard di sicurezza che si vogliono raggiungere. Pianificare la sicurezza sin dall'inizio minimizza, infine, i costi di correzione successiva e assicura un prodotto più affidabile e resistente agli attacchi.

2.1 Architettura

L'architettura progettata è composta da due macro-elementi rappresentati dal software on-chain, il quale utilizza la blockchain Ganache, e il software off-chain, il quale è poi divisibile in due moduli software, cioè il client dell'applicazione, che contiene la logica di comunicazione tra i componenti, e il database, che gestisce i dati sensibili degli utenti (Figura 26).

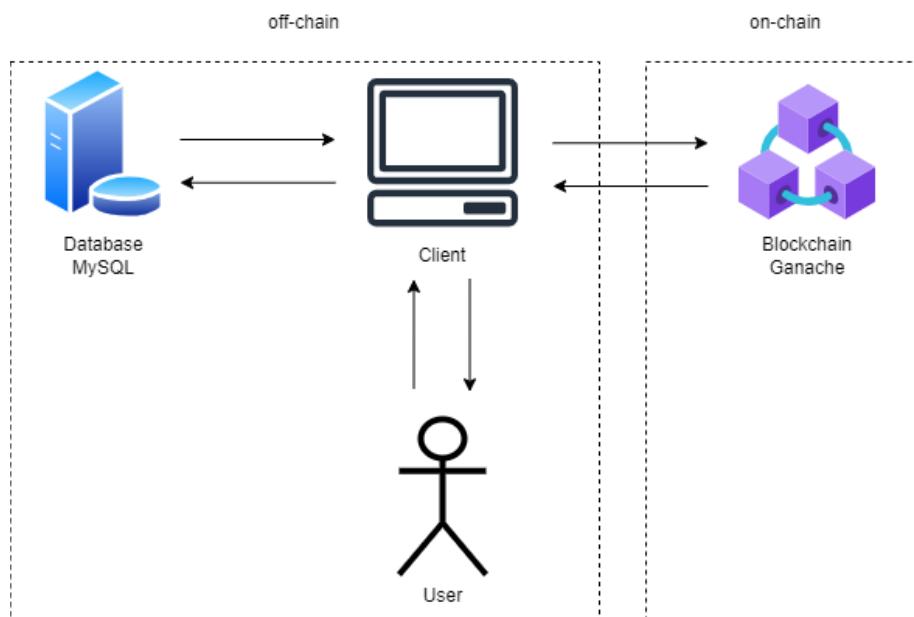


Figure 26: Schematizzazione dell'architettura

2.1.1 Database

Il ruolo del database MySQL è quello di memorizzare e gestire l'accesso alle informazioni sugli utenti che utilizzano il software; essi infatti, una volta registrati,

possiedono ognuno un account e una chiave privata necessarie per effettuare le operazioni sulla blockchain. Queste informazioni devono essere gestite in modo diverso dalle altre, quindi l'opzione più sicura si è ritenuto che fosse quella di realizzare un database comunicante con il client, il quale, riconoscendo le credenziali di accesso dell'utente, fornisce le informazioni necessarie a consentire la connessione con la blockchain.

2.1.2 Client

Il modulo centrale del software è senz'altro quello che implementa il client dell'applicazione; in esso, infatti, è stato implementato in Python con l'utilizzo del pattern architetturale MVC e include:

- l'interfaccia grafica: implementata con PySimpleGUI, molto utile per generare dinamicamente delle finestre grafiche per consentire all'utente di eseguire in modo semplice le operazioni;
- l'interfaccia con il database: implementata con il pacchetto "mysql.connector", che consente di connettersi da remoto al database e accedere alle credenziali degli utenti;
- l'interfaccia con la blockchain: la quale è stata implementata con il pacchetto "web3.py", che mette a disposizione metodi di comunicazione con la blockchain.

L'elemento più complesso è l'interfaccia di comunicazione con la blockchain, essa infatti è stata realizzata con l'uso di classi Python che fungono da model per l'utilizzo delle funzioni della blockchain. Tali model vengono richiamati dalle view e gestiti attraverso il controller; quest'ultimo poi, è utilizzato anche per avviare, come thread paralleli all'esecuzione dell'applicazione, delle routine di sicurezza che consentono la realizzazione di un log delle operazioni e il monitoraggio dell'utilizzo delle risorse hardware.

2.1.3 Blockchain

Il software relativo alla blockchain è stato realizzato in solidity e viene utilizzato per gestire le informazioni dell'applicazione. Le informazioni relative agli utenti e alle cartelle mediche sono contenute all'interno degli smart contract, i quali contengono i metodi di accesso ai dati e la gestione degli eventi relativi alla scrittura, modifica o eliminazione delle informazioni. Per la realizzazione della blockchain è stato utilizzato Ganache, un software sviluppato da Truffle Suite che consente di configurare in pochi click una blockchain privata e locale, compatibile con ethereum e particolarmente indicata per lo sviluppo.

2.2 Architettura distribuita ridondante e diversificata

Le architetture distribuite, ridondanti e diversificate giocano un ruolo cruciale nel rafforzare la sicurezza informatica in un'epoca in cui le minacce digitali

sono sempre più sofisticate e pervasive. L'integrità e la disponibilità dei sistemi sono priorità fondamentali per le aziende, e l'adozione di queste architetture contribuisce in modo determinante a proteggere le infrastrutture critiche da attacchi, guasti e vulnerabilità.

2.2.1 Architettura distribuita

Le architetture distribuite frammentano l'elaborazione dei dati su diversi nodi o server, rendendo più difficile per gli attaccanti compromettere l'intero sistema con un singolo punto di attacco. Questa suddivisione permette di ridurre l'impatto di eventuali intrusioni o attacchi mirati, come i Distributed Denial of Service (DDoS), perché il sistema può ridistribuire il carico e mantenere operativa una parte della rete anche in caso di attacco. Questa proprietà è stata implementata grazie alla blockchain; essa rappresenta, infatti, un sistema di gestione dei dati basato sulla distribuzione delle loro copie su diversi nodi, il che rende difficile per un eventuale attaccante esterno apportare una modifica che venga ritenuta valida e applicata su tutti i nodi del sistema. La blockchain inoltre, essendo un registro immutabile, consente di tracciare tutte le modifiche effettuate sui dati, cosa che facilita eventuali indagini a seguito di un attacco.

2.2.2 Architettura ridondante

La ridondanza è essenziale per garantire la disponibilità del servizio anche in caso di guasti hardware o software, ma è altrettanto importante per mitigare attacchi come il ransomware, che possono compromettere una porzione del sistema. Se un componente o un server viene compromesso, grazie alla ridondanza un altro può immediatamente prendere il suo posto, garantendo la continuità del servizio. Questa proprietà è fondamentale in settori critici come quello finanziario o governativo, dove l'interruzione di un servizio può avere conseguenze disastrose. Anche questa proprietà è garantita dalla blockchain; i dati applicativi, infatti, hanno una copia per ogni nodo della rete, e questo garantisce che il malfunzionamento, la corruzione o l'attacco che influenzano un nodo non impattino i dati dell'intero sistema.

2.2.3 Architettura diversificata

La diversificazione, cioè l'uso di piattaforme e tecnologie diverse all'interno della stessa infrastruttura, è una strategia chiave per ridurre il rischio di vulnerabilità sistemiche. Utilizzando software, sistemi operativi o fornitori cloud differenti, si minimizza l'esposizione a exploit specifici, che potrebbero colpire una sola piattaforma. In un ambiente diversificato, anche se un singolo sistema viene compromesso, il danno può essere contenuto, migliorando la resilienza contro attacchi zero-day e vulnerabilità non ancora note. Questa proprietà è stata implementata tramite l'utilizzo di Docker, una piattaforma open-source che automatizza il processo di creazione, distribuzione e gestione di applicazioni all'interno di container. I container sono ambienti isolati e leggeri che includono

tutto il necessario per eseguire un'applicazione, come il codice, le librerie e le dipendenze, garantendo che il software funzioni in modo coerente su qualsiasi sistema. In questo modo l'applicazione può essere utilizzata con sistemi operativi, compilatori e hardware qualsiasi, riducendo la presenza e l'impatto di vulnerabilità dovute al sistema di elaborazione.

2.3 Monitoraggio isolamento e offuscamento

Monitoraggio, isolamento e offuscamento sono tre pilastri essenziali per rafforzare la sicurezza informatica in un ambiente sempre più complesso e minacciato da attacchi sofisticati. Questi principi aiutano a proteggere i sistemi, rilevare attività sospette e minimizzare i rischi di compromissione, garantendo una difesa più completa contro attacchi esterni e interni.

2.3.1 Monitoraggio

Il monitoraggio (monitoring) continuo delle reti, dei sistemi e delle applicazioni consente di rilevare anomalie e comportamenti sospetti in tempo reale. Strumenti di intrusion detection e SIEM (Security Information and Event Management), possono individuare tentativi di accesso non autorizzato, modifiche sospette e attività insolite. Un monitoraggio efficace permette di reagire prontamente a potenziali minacce, riducendo il tempo di esposizione e limitando i danni. Nell'applicazione sono state implementate delle routine di monitoraggio attraverso l'uso di thread paralleli all'esecuzione dell'applicazione ed eseguiti all'interno del controller:

- monitoraggio degli eventi: è stato implementato attraverso una funzione in ascolto di alcuni eventi particolari (Figura 27), definiti all'interno di un filtro, la quale esegue azioni di risposta ad essi. Il modo in cui è stata definita consente di inserire facilmente nuovi eventi tra quelli in ascolto e di avviare routine differenti in base all'evento;
- monitoraggio uso risorse: è stato implementato mediante una funzione che, grazie alla libreria "psutil" registra periodicamente la percentuale utilizzata di CPU, memoria e disco all'interno di un log apposito (Figura 28). Questo consente di immagazzinare informazioni riguardo eventuali malfunzionamenti o alterazioni della normale computazione dell'applicazione;
- monitoraggio delle attività: grazie alla libreria "logging" di Python, è stato implementato un sistema di registrazione di tutte le attività eseguite sul client, sia riguardo le modifiche ai dati, sia riguardo alla visualizzazione degli stessi, registrando anche il codice fiscale di chi ha eseguito tale operazione; ad esempio se un dottore accede al fascicolo di un paziente anche solo per visualizzarne il contenuto, verrà registrato il codice fiscale del dottore e del paziente di cui ha visualizzato le informazioni.


```
def listen_to_events(doctorContracts, caregiverContracts, patientContracts):
    filters = {
        '1': doctorContracts.contract.events.DoctorRegistered().create_filter(fromBlock='latest'),
        '2': patientContracts.contract.events.PatientRegistered().create_filter(fromBlock='latest'),
        '3': caregiverContracts.contract.events.CaregiverRegistered().create_filter(fromBlock='latest')
    }
    while True:
        for event_name, event_filter in filters.items():
            for event in event_filter.get_new_entries():
                handle_event(event)
```

Figure 27: Funzione di ascolto degli eventi

```
def monitor_system():
    file_path = 'offChain/monitoring/systemMonitoring.txt'
    with open(file_path, 'w') as file:
        while True:
            file.write(f"CPU usage: {psutil.cpu_percent()}%")
            file.write(f"Memory usage: {psutil.virtual_memory().percent}%")
            file.write(f"Disk usage: {psutil.disk_usage('/').percent}%")
            file.write(f"VRAM usage: 0,1%")
            file.write("Data e ora formattata: " + time.strftime("%d-%m-%Y %H:%M:%S", time.localtime()))
            file.write("\n")
            file.flush()
            time.sleep(10)
            if fine.is_set():
                break
```

Figure 28: Funzione di monitoraggio delle risorse hardware utilizzate

```
if event == '0k':
    healthFile = healthFileResearch(cf, healthFileContracts)
    if healthFile:
        windowHome['-OUTPUT-'].update('')
        windowHome.Hide()
        logging.info('Fascicolo con codice: ' + healthFile.cf +
                    ' aperto dal dottore: ' + doctor.lastname + ' ' + doctor.name + ' con codice fiscale: '
                    + doctor.cf)
```

Figure 29: Funzione di registrazione delle operazioni effettuate

2.3.2 Isolamento

L'isolamento (isolation) dei sistemi è una pratica che prevede la separazione di ambienti critici o potenzialmente vulnerabili dal resto dell'infrastruttura. Tecniche come la sandboxing e la containerizzazione limitano il movimento laterale di eventuali attaccanti, impedendo che una compromissione locale si diffonda al resto del sistema. Isolare applicazioni e dati sensibili riduce l'impatto di attacchi e aumenta la resilienza del sistema. L'isolamento è stato implementato grazie a Docker, che implementa il concetto di isolamento attraverso l'uso dei container, i quali separano le applicazioni e i loro ambienti di esecuzione dal sistema operativo sottostante e dalle altre applicazioni. Ogni container include il proprio

set di risorse, come file system, librerie, variabili di ambiente e processi, garantendo che funzioni in modo indipendente e isolato dagli altri. La configurazione del container Docker avviene attraverso tre file principali, il docker-compose, il Dockerfile e il file requirements.txt:

- il docker-compose è uno strumento che consente di definire e gestire applicazioni multi-container tramite un file YAML (solitamente chiamato docker-compose.yml) (Figura 30). In questo file, vengono descritti i vari servizi (container) di cui l'applicazione ha bisogno, come database, backend e frontend, permettendo di eseguire più container insieme con un solo comando. È utile per orchestrare l'intero stack applicativo e semplificare l'avvio e la gestione dei container;
- il Dockerfile è uno script che definisce come costruire un'immagine Docker personalizzata (Figura 31). Contiene istruzioni come l'installazione di dipendenze, la copia dei file necessari, la configurazione delle variabili di ambiente e l'impostazione del comando principale da eseguire all'avvio del container. In pratica, è il file che descrive la "ricetta" per creare l'immagine Docker che verrà utilizzata nei container;
- il requirements.txt è un file utilizzato principalmente nei progetti Python per elencare le dipendenze del progetto, come librerie e moduli necessari (Figura 32). Docker lo utilizza spesso in combinazione con il Dockerfile per installare automaticamente tutte le dipendenze all'interno del container Python durante la fase di build. Questo garantisce che l'ambiente del container abbia tutte le librerie necessarie per far funzionare correttamente l'applicazione.

```

version: '3.8'
#la versione non so quale mettere per ora lascio questa
services:
  ganache:
    image: trufflesuite/ganache-cli
    command: ganache-cli --host 0.0.0.0 --port 8080 --accounts 10
    #qui specifica i parametri in esecuzione su ganache
    ports:
      - 8080:8080
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8545"]
      interval: 10s
      timeout: 10s
      retries: 5

```

Figure 30: Un esempio di servizio presente nel file docker-compose.yml, in questo caso ganache

```

FROM python:3.11

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
# Copia le immagini di riferimento nel container

#dobbiamo scegliere la nostra directory per ora è HealthChain
WORKDIR /HealthChain
COPY . /HealthChain

#pure qui possiamo lasciare o cambiare
EXPOSE 8000

#qui dobbiamo far partire il nostro progetto quindi ad esempio ora sarebbe HealthChain/offChain/main.py
CMD [ "python", "/HealthChain/offChain/main.py" ]

```

Figure 31: Contenuto del dockerfile

```

web3 == 6.20.2
mysql-connector-python
py-solc-x
PySimpleGUI
pyautogui
psutil
faker

#qui dentro dobbiamo inserire tutto quello che c'è da installare per far funzionare l'applicazione
#ad esempio se usiamo pyGui dobbiamo mettere anche quella etc.

```

Figure 32: Contenuto del file dei requisiti

2.3.3 Offuscamento

L'offuscamento (obfuscation) è una tecnica che maschera o rende più complesso l'accesso alle informazioni sensibili o al codice sorgente, rendendo più difficile per gli attaccanti comprendere il funzionamento di un'applicazione o di un sistema. Questo processo può includere l'offuscamento del codice sorgente, l'uso di crittografia per dati sensibili o la creazione di falsi bersagli per confondere potenziali aggressori. L'offuscamento non previene gli attacchi, ma aumenta significativamente la difficoltà per gli aggressori di sfruttare vulnerabilità. Questa tecnica è stata applicata nella comunicazione tra la blockchain e il client dell'applicazione; le transazioni infatti subiscono il processo di hashing prima di essere trasmesse alla blockchain, così come le informazioni che fanno il percorso contrario. Un altro esempio di applicazione dell'offuscamento è l'utilizzo della tecnica dei canarini all'interno del file di log che monitora l'utilizzo delle risorse hardware (Figura 33). In tale log, infatti, le informazioni che vengono registrate sull'utilizzo della scheda grafica sono in realtà generate dal codice, perciò in caso vengano analizzate e non risultino congruenti con quanto il codice dovrebbe aver generato, si può giungere alla conclusione che qualcuno le abbia manomesse.

```

def monitor_system():
    file_path = 'offChain/monitoring/systemMonitoring.txt'
    with open(file_path, 'w') as file:
        while True:
            file.write(f"CPU usage: {psutil.cpu_percent()}%")
            file.write(f"Memory usage: {psutil.virtual_memory().percent}%")
            file.write(f"Disk usage: {psutil.disk_usage('/').percent}%")
            file.write(f"VRAM usage: 0,1%")
            file.write("Data e ora formattata: " + time.strftime("%d-%m-%Y %H:%M:%S", time.localtime()))
            file.write("\n")
            file.flush()

```

Figure 33: Codice software che implementa la tecnica dei canarini

2.4 Sicurezza del design

Le linee guida per la progettazione del software, come quelle fornite da OWASP, Saltzer & Schroeder e Sommerville, sono fondamentali per garantire lo sviluppo di software sicuro, affidabile e di alta qualità. Queste linee guida offrono principi

e best practice che aiutano a prevenire vulnerabilità, ridurre i rischi di attacchi informatici e migliorare la robustezza e la manutenibilità del codice. Di seguito sono riportate i punti chiave su cui si è fatta maggior attenzione durante la progettazione del software.

2.4.1 Principi di design OWASP

Gli elementi descritti nelle linee guida OWASP che sono stati presi in considerazione sono:

- **minimizzare la superficie di attacco:** questo principio è particolarmente importante quando vengono implementati software gestionali come quello in esame, questo perchè solitamente si hanno molte informazioni eterogenee e un alto grado di interfacciamento tra utente e sistema software; ciò è anche amplificato dalla presenza di un interfaccia grafica come quella presentata. Per rispettare il principio di superficie di attacco minima è stato molto importante l'utilizzo di Docker; il docker-compose, infatti, come descritto precedentemente, consente di dichiarare quali servizi sono attivi e su quali porte, questo minimizza la possibilità di interagire esternamente con le interfacce di comunicazione del sistema. In merito all'interfaccia grafica inoltre, sono stati implementati i componenti di interazione minimi, i quali sono inoltre perennemente sanificati e monitorati dal sistema.
- **Sicurezza di default:** le informazioni trattate dal software sono estremamente sensibili, per questo motivo non si può fare affidamento sul rispetto delle policy di sicurezza da parte degli utenti. Per mettere in pratica questo principio, quindi, il sistema non accetta credenziali che non abbiano una complessità sufficiente, e non consente di disabilitare le procedure di autenticazione o autorizzazione.
- **privilegi minimi:** questo principio consiste nell'impedire la privilege escalation e ridurre i danni che possono essere commessi quando un utente viene compromesso minimizzando le risorse concesse ad ogni utente. Questo principio è stato seguito dando accesso alle sole informazioni indispensabili per ogni utente; il caregiver, ad esempio, per svolgere i suoi compiti non necessita di conoscere la storia clinica di un paziente, ma solo delle cure e i trattamenti che deve ricevere, per questo motivo i caregiver non hanno accesso alla cartella clinica dei loro pazienti.
- **fallire in modo sicuro:** consiste nel garantire che in presenza di un'attività che non va a buon fine il sistema non si arresti, bensì continui a funzionare correttamente. Questo principio è stato implementato utilizzando dei messaggi di errore che vengono restituiti ad un utente che esegue un'operazione che non termina correttamente, senza arrestare il software e dandogli la possibilità di tentare nuovamente l'operazione o eseguire altre attività.
- **Separazione dei compiti:** prevede la divisione di responsabilità e privilegi tra più persone o processi per prevenire abusi o frodi. L'idea centrale è che

nessun singolo individuo o sistema dovrebbe avere il controllo completo su tutte le fasi critiche di un processo. Questo principio è stato implementato ad esempio facendo sì che l'amministratore possa gestire dottori e caregiver, ma non abbia modo di accedere ai dati dei pazienti poiché ciò è compito esclusivo delle altre due figure.

- Evitare la sicurezza basata sull'oscuramento: è un principio che afferma che la sicurezza di un sistema non deve dipendere esclusivamente dal mantenere segreti i dettagli della sua implementazione poiché questa è una pratica di scarsa efficacia. In sostanza bisogna fare in modo che, anche se un avversario dovesse conoscere il funzionamento interno di un sistema, questo dovrebbe rimanere sicuro grazie alle misure di protezione implementate. Questo principio è stato implementato evitando di utilizzare nel codice delle procedure che se analizzate consentono di entrare nel sistema.
- Mantenere i sistemi di sicurezza semplici: è un principio che si basa sull'evitare architetture o meccanismi di sicurezza complessi, poiché, sebbene possano sembrare più sicuri, spesso causano errori di manutenzione o implementazione, perciò vanno evitati.

2.4.2 Linee guida di Sommerville

Gli elementi descritti nelle linee guida di Sommerville che sono stati presi in considerazione sono:

- Basare le decisioni su politiche di sicurezza ben definite: si basa sul concetto per cui tutte le decisioni relative alla sicurezza di un sistema devono essere basate su una politica di sicurezza chiara e ben definita. Questa politica dovrebbe specificare in modo esplicito i requisiti di sicurezza, gli obiettivi e le misure da adottare, offrendo una guida per le scelte tecniche e organizzative. In questo modo, le decisioni non sono arbitrarie o lasciate all'interpretazione dei singoli, ma seguono un insieme di regole coerenti e trasparenti. Ciò riduce il rischio di vulnerabilità dovute a decisioni incoerenti o non allineate con gli obiettivi di sicurezza complessivi. Questo principio è stato realizzato effettuando l'analisi degli asset e dei rischi tramite il modello dual-stride;
- Bilanciare sicurezza e usabilità: consiste nel trovare un equilibrio tra proteggere il sistema da attacchi e garantire che gli utenti possano utilizzarlo senza difficoltà eccessive. Se la sicurezza è troppo restrittiva, può rendere il sistema complicato o frustrante da usare, spingendo gli utenti a trovare soluzioni alternative che possono compromettere la sicurezza stessa. D'altro canto, un'eccessiva enfasi sull'usabilità a scapito della sicurezza può lasciare il sistema vulnerabile agli attacchi. È quindi importante progettare un sistema che sia sicuro ma anche intuitivo e facile da usare. Per implementare questo principio sono state implementate poche procedure di sicurezza da eseguire per gli utenti, tuttavia quelle presenti

non sono evitabili o sostituibili; in effetti questo principio è stato ritenuto più importante del "Evitare un singolo punto di fallimento" il quale fa riferimento all'implementazione di procedure di sicurezza alternative nel caso una non funzioni; tuttavia questo rende il sistema più complesso sia da implementare che da utilizzare, per cui si è preferito mettere in secondo piano questo elemento.

- Mantenere un registro delle azioni degli utenti: suggerisce di registrare le azioni degli utenti all'interno di un sistema per aumentare la sicurezza e facilitare la gestione delle operazioni. Tenere traccia delle attività utente consente di monitorare comportamenti sospetti, rilevare potenziali minacce e indagare su incidenti di sicurezza. Questo principio è stato implementato utilizzando dei log sia per le operazioni on-chain, gestite automaticamente da ganache, che dalle operazioni del client, come descritto al paragrafo 2.3.1;
- Utilizzare ridondanza e diversità per ridurre i rischi: suggerisce di utilizzare sistemi ridondanti e diversificati per ridurre la probabilità di fallimenti e minimizzare i rischi. L'applicazione di questi principi è stata effettuata come già trattato ai paragrafi 2.2.2 e 2.2.3.
- Specificare il formato degli input: afferma che ogni input di sistema dovrebbe avere un formato ben definito e chiaramente specificato. Questo riduce il rischio di errori o vulnerabilità legate all'inserimento di dati imprevisti o malevoli. Stabilire regole precise per il formato degli input aiuta a prevenire attacchi come l'SQL injection o il buffer overflow, migliorando la sicurezza e l'affidabilità del sistema. Inoltre, specificare formati chiari contribuisce a una migliore interoperabilità tra i componenti del sistema e una maggiore facilità di gestione. Questo principio è stato implementato come trattato al paragrafo 3.1.

2.4.3 Linee guida di Saltzer & Schroeder

Gli elementi descritti nelle linee guida di Saltzer & Schroeder che sono stati presi in considerazione rappresentano una combinazione dei principi descritti precedentemente, perciò non verranno spiegati ulteriormente per evitare ripetizioni, ma vengono solamente elencati di seguito:

- privilegi minimi;
- "KISS" (Mantenere i sistemi di sicurezza semplici);
- evitare la sicurezza basata sull'oscuramento;
- fallire in modo sicuro;
- separare i privilegi;
- semplicità di utilizzo.

2.4.4 Scelte tecnologiche

Le scelte tecnologiche sono fondamentali per la sicurezza informatica, poiché determinano il livello di protezione contro attacchi e vulnerabilità. La selezione accurata di strumenti, piattaforme e infrastrutture può ridurre i rischi, prevenire violazioni dei dati e garantire la continuità operativa. Ignorare l'importanza della sicurezza nelle decisioni tecnologiche può esporre le organizzazioni a minacce costose e difficili da gestire. Le tecnologie utilizzate sono state quindi analizzate per valutarne eventuali criticità che potessero compromettere la sicurezza del sistema. Di seguito vengono elencate le tecnologie utilizzate con una breve spiegazione dei pro e dei contro:

2.4.5 Python

- **Punti di forza:** Python offre un'ampia gamma di librerie per la sicurezza (es. `cryptography`, `hashlib`), facilitando l'implementazione di algoritmi crittografici e funzioni di hashing. Grazie alla vasta comunità open-source, gli aggiornamenti e le patch di sicurezza sono tempestivi, riducendo la vulnerabilità a minacce emergenti. La sua facilità d'uso lo rende ideale per test di sicurezza (es. `pentesting`) e lo sviluppo di strumenti per il monitoraggio e la protezione delle reti.
- **Punti di debolezza:** Python è un linguaggio interpretato, quindi può essere meno efficiente rispetto a linguaggi compilati quando si tratta di gestire operazioni critiche di sicurezza in tempo reale. Dipendenze esterne: la gestione di librerie di terze parti può introdurre vulnerabilità se non sono aggiornate o verificate adeguatamente. Tipizzazione dinamica: l'assenza di una tipizzazione statica può portare a errori imprevedibili o a problemi di sicurezza difficili da individuare prima dell'esecuzione del codice.

2.4.6 MySQL

- **Punti di forza:** MySQL offre un sistema di gestione degli utenti robusto, con controlli granulari sui permessi a livello di database, tabelle e colonne, riducendo il rischio di accesso non autorizzato. MySQL supporta la crittografia delle connessioni attraverso SSL/TLS, proteggendo i dati in transito tra client e server. È possibile integrare MySQL con firewall e sistemi di monitoraggio per individuare tentativi di accesso sospetti e prevenire attacchi.
- **Punti di debolezza:** le impostazioni predefinite di MySQL possono essere meno sicure, richiedendo una configurazione personalizzata per rinforzare la sicurezza (es. disabilitazione dell'accesso remoto non necessario). Le vulnerabilità di sicurezza possono emergere rapidamente, e la mancata applicazione tempestiva delle patch di sicurezza può lasciare i sistemi esposti ad attacchi. Sebbene MySQL offra meccanismi di protezione, è ancora suscettibile a attacchi di SQL injection se le applicazioni non validano correttamente gli input o non usano query preparate.

2.4.7 Ganache

- Punti di forza: Ganache permette di creare una blockchain locale isolata, ideale per sviluppare e testare contratti intelligenti senza rischiare la compromissione di una blockchain reale. Consente di simulare le condizioni di rete, inclusi i gas fee e il mining, offrendo agli sviluppatori un ambiente sicuro per testare i comportamenti dei contratti in diverse situazioni, prima di implementare sulla mainnet. Ganache fornisce strumenti di debug per tracciare e identificare facilmente eventuali bug o vulnerabilità nei contratti intelligenti, aiutando a migliorare la sicurezza del codice.
- Punti di debolezza: poiché Ganache è destinato esclusivamente a un ambiente di test locale, non offre protezioni di sicurezza avanzate per un ambiente di produzione. Non riflette tutte le minacce reali presenti su blockchain pubbliche come attacchi DoS o exploit di rete. La sicurezza dell'ambiente di sviluppo è legata alle competenze e alla configurazione dell'utente. Una configurazione errata o una mancata gestione delle chiavi private può portare a vulnerabilità, anche in fase di test. Anche se Ganache simula bene molte condizioni della blockchain, non può riprodurre completamente alcune situazioni complesse di rete e sicurezza che si verificano nelle blockchain pubbliche decentralizzate.

2.4.8 Docker

- Punti di forza: Docker utilizza container per isolare le applicazioni e i loro processi, riducendo il rischio che una vulnerabilità in un'applicazione comprometta l'intero sistema. Docker permette la gestione precisa dei permessi con strumenti come Docker Compose e Docker Swarm, limitando i privilegi delle applicazioni containerizzate. Docker consente di aggiornare rapidamente i container, migliorando la sicurezza grazie all'applicazione rapida delle patch e degli aggiornamenti senza interruzioni prolungate del servizio.
- Punti di debolezza: i container Docker condividono lo stesso kernel del sistema host. Una vulnerabilità nel kernel potrebbe permettere a un attaccante di compromettere l'host o altri container. Configurazioni predefinite eccessivamente permissive, come l'utilizzo di privilegi di root o porte esposte non sicure, possono aumentare i rischi di attacchi. Le immagini Docker possono contenere vulnerabilità se non vengono mantenute aggiornate o se provengono da fonti non affidabili. Un'immagine compromessa può diffondere malware o exploit all'interno di un ambiente di produzione.

2.4.9 Solidity

- Punti di forza: Solidity consente di creare contratti intelligenti che vengono eseguiti su blockchain come Ethereum, garantendo trasparenza e immutabilità. Una volta distribuito, il contratto è pubblico e non può essere

modificato, riducendo il rischio di manomissioni. Ecosistema di strumenti per la sicurezza: esistono numerosi strumenti per l'analisi della sicurezza del codice Solidity, come MythX e Slither, che permettono agli sviluppatori di individuare bug e vulnerabilità nei contratti prima della distribuzione. Solidity supporta funzionalità come modificatori e require/assert per controllare e convalidare le condizioni del contratto, contribuendo a prevenire comportamenti indesiderati e vulnerabilità comuni.

- Punti di debolezza: poiché i contratti sono immutabili una volta distribuiti su blockchain, eventuali bug o vulnerabilità rimangono irreparabili, esponendo i fondi degli utenti a rischi elevati. Solidity è un linguaggio complesso, e la sua struttura può portare facilmente a errori nello sviluppo. Gli attacchi come il reentrancy o l'overflow aritmetico sono esempi di problemi di sicurezza derivati da codici mal progettati. Le funzionalità di debugging di Solidity sono limitate, e il processo di rilevamento di bug in una blockchain pubblica è complesso. Ciò rende difficile individuare problemi di sicurezza una volta che il contratto è attivo.

2.4.10 GitHub

- Punti di forza: GitHub permette un tracciamento preciso delle modifiche al codice tramite il versioning, facilitando il controllo e l'individuazione di vulnerabilità introdotte nel tempo. GitHub offre funzionalità di scansione automatica delle vulnerabilità, come Dependabot, che identifica e avvisa su dipendenze vulnerabili, oltre a CodeQL, per l'analisi statica del codice. È possibile gestire in modo granulare i permessi degli utenti per accedere ai repository, limitando chi può visualizzare o modificare il codice, e integrando l'autenticazione a due fattori (2FA) per una maggiore protezione degli account.
- punti di debolezza: gli utenti possono accidentalmente committare file contenenti credenziali sensibili, come chiavi API o password, rendendo queste informazioni vulnerabili se il repository è pubblico. Gli attacchi alla catena di fornitura software possono avvenire tramite dipendenze di codice open-source ospitate su GitHub, se non adeguatamente monitorate o mantenute. Se non configurati correttamente, i permessi sui repository possono lasciare accessi non autorizzati, specialmente se le policy di sicurezza aziendali non sono rigorose.

3 Programmazione sicura

Sebbene l'architettura ideata in fase di progettazione sia fondamentale per garantire un buon livello di sicurezza, è fondamentale anche rispettare le pratiche e le linee guida di sicurezza del software anche in fase di implementazione. Per fare questo è fondamentale rispettare gli standard di programmazione o le linee guida relative ai linguaggi di programmazione utilizzati, che nel nostro caso sono Python e Solidity.

3.1 Python

Per Python non è ancora stato realizzato uno standard di programmazione sicura, perciò nella fase di implementazione si è cercato di rispettare le linee guida generali descritte di seguito.

- Limitare la visibilità delle informazioni: consiste nel dividere i moduli del software in modo che ognuno di essi possa accedere alle sole informazioni di cui ha bisogno. Questa proprietà è stata implementata implicitamente con l'architettura del software. Le classi implementate, infatti, sono delle interfacce per la blockchain dove sono memorizzate le informazioni, e possono essere utilizzate solo dopo un controllo sull'autorizzazione.
- Controllare la validità degli input: consiste nel controllare cosa può essere inserito dall'utente al fine di evitare comportamenti inaspettati del programma causati da input non previsti. Questa linea guida è stata implementata attraverso la chiamata ad una funzione di sanificazione in presenza di ogni elemento di input dell'interfaccia grafica; tale funzione esegue un controllo sulle parole pericolose note attraverso una "black list" e successivamente un controllo sui caratteri utilizzati attraverso una "white list" di caratteri utilizzabili. Per le stringhe che rappresentano informazioni brevi, come il nome e il cognome, è presente anche un controllo sulla lunghezza direttamente nel campo di inserimento; un altro caso particolare si ha per il codice fiscale, che deve avere una lunghezza fissa di 16 caratteri alfanumerici.

```
def sanitizeInput(value):  
  
    sanitizedValue = re.sub( pattern: r'\b(import|exec|eval)\b', repl: '', value, flags=re.IGNORECASE)  
  
    if re.search( pattern: r'^[a-zA-Z0-9\s]', sanitizedValue):  
        return ''  
  
    return sanitizedValue
```

Figure 34: Funzione di sanificazione dell'input

- Realizzare un handler per ogni eccezione: consiste nell'utilizzare i costrutti di gestione delle eccezioni per evitare di fare controlli condizionali numerosi

e frequenti, che, oltre a rappresentare un overhead rilevante per la computazione, sono anche prone agli errori di scrittura del codice. Questa linea guida è stata implementata utilizzando il costrutto Python "try-except" per gestire eventuali malfunzionamenti senza portare stati inaspettati del sistema.

```
try:
    doctorContracts = Doctor(provider_url, conn)
    caregiverContracts = Caregiver(provider_url, conn)
    patientContracts = Patient(provider_url, conn)
    healthFileContracts = HealthFile(provider_url, conn)
except Exception as err:
    print('connessione alla blockchain fallita')
```

Figure 35: Esempio di gestione delle eccezioni

- Minimizzare dell'uso di costrutti "error-prone": consiste nell'evitare costrutti che possono rendere la logica del programma difficile da comprendere e con alta probabilità introdurre bug, ad esempio l'uso del "go to", delle interrupt o dei puntatori; anche l'uso del parallelismo e dei numeri in virgola mobile può causare problematiche di questo tipo. Per rispettare questa linea guida sono state implementate funzioni senza gli elementi sopra citati, fatta eccezione per il parallelismo, il quale è stato ritenuto indispensabile al fine di poter eseguire delle routine di sicurezza che non andassero a inficiare sull'esperienza dell'utente. La realizzazione del log delle attività eseguite e il monitoraggio delle risorse utilizzate, per fare degli esempi, sono due processi fondamentali per garantire la sicurezza del sistema, per cui vanno eseguiti costantemente in parallelo all'utilizzo dell'utente. Questi processi, tuttavia, sono completamente indipendenti dall'esecuzione ordinaria dell'applicativo, per cui, anche grazie ai test eseguiti, ci si è assicurati che non intacchino il resto del sistema.
- Prevedere la proprietà di riavvio: consiste nell'implementare una routine che consenta all'utente, dopo un possibile guasto, di riavviare il sistema e riprendere le attività che stava eseguendo senza dover ricominciare o senza perdere informazioni. Questa proprietà è stata implementata grazie alla comunicazione tra la blockchain e il sistema sulla macchina locale. Nel caso in cui sulla macchina locale si abbiano dei problemi, dato che il software è installato in un container virtuale di docker, sarà sufficiente riavviare il container per riportare il software locale ad uno stato sicuro. Il resto delle attività, anche parziali, eseguite prima dell'ipotetico crash, verranno poi reperite dalla blockchain dopo il riavvio.

- Controllare i limiti dei vettori: una vulnerabilità nota nei programmi in cui non vengono controllati i limiti dei vettori, è quella per cui un attaccante scrive deliberatamente oltre il limite di memoria allocata ad un vettore. Per evitare questa problematica è indispensabile controllare la dimensione dei vettori utilizzati ed eseguire dei controlli quando vengono aggiunti o rimossi degli elementi. Per rispettare questa linea guida, i vettori vengono utilizzati solo sulla blockchain, dove è più facile controllarne l'uso e non si hanno rischi di sicurezza.
- Utilizzare dei timeout per le chiamate a moduli esterni: consiste nell'impostare dei limiti di tempo per la ricezione dei messaggi di risposta durante la comunicazione tra moduli su macchine diversi; superati tali limiti di tempo bisogna presupporre che il servizio esterno stia subendo un malfunzionamento. Per implementare questa linea guida, nel contesto della comunicazione tra il codice locale in Python e la blockchain, sono stati utilizzati dei metodi del modulo web3.py che in caso di mancata risposta sollevano un errore del tipo "TimeExhausted".

```
def update_caregiver(self, cf, name, lastname):
    data = super().cf_to_address(cf)
    transaction = self.contract.functions.updateCaregiver(name, lastname, cf).build_transaction({
        'from': data['address'],
        'nonce': self.web3.eth.get_transaction_count(data['address']),
        'gas': 2000000,
        'gasPrice': self.web3.to_wei('0', 'gwei')
    })

    signed_txn = self.web3.eth.account.sign_transaction(transaction, private_key=data['private_key'])
    tx_hash = self.web3.eth.send_raw_transaction(signed_txn.rawTransaction)
    receipt = self.web3.eth.wait_for_transaction_receipt(tx_hash)
    return receipt
```

Figure 36: Esempio di utilizzo di un timeout in corrispondenza della chiamata ad un servizio esterno

- Dare un nome alle costanti che rappresentano valori reali: consiste nel fare riferimento al nome delle costanti globali piuttosto che il loro valore numerico, cosa che diminuisce il rischio di commettere errori nella scrittura del codice. Per il software realizzato non è stato necessario utilizzare valori reali, perciò il rispetto di questa regola è dovuto al non utilizzo di valori costanti per la logica del sistema sviluppato.

3.2 Solidity

Il codice Solidity utilizzato per la blockchain è stato testato utilizzando gli strumenti di Remix: "Solidity Analyzer e Solhint", inoltre per verificarne la conformità agli standard di programmazione sicura è stato utilizzato il documento "EEA EthTrust Security Levels Specification version 2" ed è stato verificato il

rispetto di tutte le regole di sicurezza per tutti i tre livelli di conformità presenti. I test successivi sono poi stati eseguiti utilizzando la blockchain "Ganache".

3.3 User's action flow

L'interfaccia utente è stata implementata utilizzando la libreria di python "PySimpleGUI", che mette a disposizione delle classi e metodi per creare e gestire in modo rapido e semplice l'interfaccia grafica. La natura del contesto di applicazione ha reso necessario mettere a disposizione dell'utente numerosi campi di input che potrebbero essere utilizzati per inserire del codice da parte di malintenzionati; per questo motivo ogni elemento dell'interfaccia grafica subisce come prima fase di elaborazione una procedura di sanificazione (Figura 37). L'utilizzo di una funzione indipendente e in comune a tutti i campi di inserimento è utile per evitare duplicazione di codice e per aggiornare il contenuto della funzione se vengono scoperte nuove vulnerabilità o se cambiano le policy di sicurezza.

```
def sanitizeInput(value):  
  
    sanitizedValue = re.sub(pattern: r'\b(import|exec|eval)\b', repl: '', value, flags=re.IGNORECASE)  
  
    if re.search(pattern: r'^[a-zA-Z0-9\s]', sanitizedValue):  
        return ''  
  
    return sanitizedValue
```

Figure 37: Funzione di sanificazione

3.3.1 Login

La prima schermata che appare all'utente è quella di login, in cui si può accedere utilizzando le proprie credenziali, creare un nuovo profilo per i pazienti, oppure uscire dall'applicazione. Al posto dell'username si è scelto di utilizzare il codice fiscale, e come controllo ulteriore per verificarne la correttezza, è presente per il rispettivo campo di input, come in tutti i successivi campi di input in cui ci si aspetta un codice fiscale, un controllo sulla stringa che deve contenere esattamente 16 caratteri esclusivamente alfanumerici.

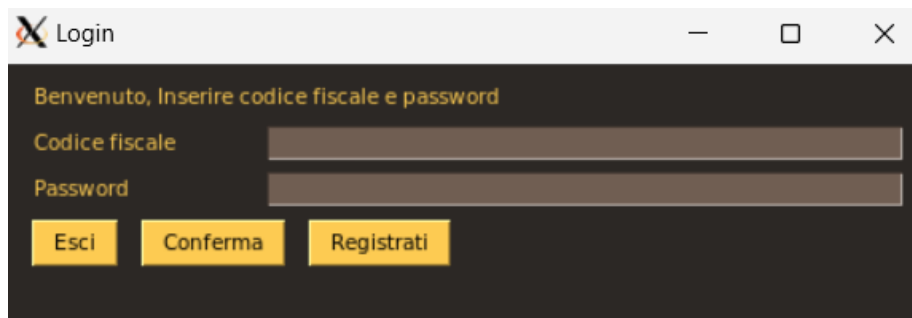


Figure 38: Schermata di login

3.3.2 Registrazione

Cliccando il bottone di registrazione si apre una nuova finestra in cui appare il form di registrazione (Figura 39), il quale richiede le generalità del paziente e chiede di impostare una password per il login, la quale subisce un controllo sulla lunghezza affinché non sia possibile utilizzare password facilmente intuibili. Quando viene effettuata la registrazione il software controlla che non esista un altro paziente con lo stesso codice fiscale, poi genera un nuovo address e una nuova private key per la blockchain, i quali, insieme alla password dell'utente, vengono memorizzati su un database MySQL esterno.

Figure 39: Schermata di registrazione per i pazienti

3.3.3 Home del paziente

Una volta eseguita l'autenticazione, viene presentata al paziente una finestra chiamata "home paziente" (Figura 40) in cui si possono eseguire alcune azioni:

- visualizzare le proprie informazioni contenute nella cartella clinica
- dare conferma delle cure ricevute dal proprio caregiver (funzionalità disponibile solo se al momento della registrazione il paziente viene ritenuto autosufficiente)
- visualizzare le informazioni relative al profilo dell'applicazione ed eventualmente modificarle
- effettuare il logout e uscire dal proprio profilo

Si evidenzia che il bottone per la conferma delle cure è gestito dinamicamente in fase di ricezione delle informazioni dalla blockchain, e nel caso in cui il paziente non sia indipendente (e di conseguenza come da policy non abbia la possibilità di confermare le cure ricevute), tale bottone non viene generato all'interno della schermata.

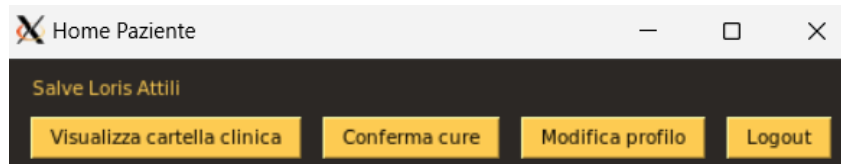


Figure 40: Schermata home del paziente autenticato

3.3.4 Cartella clinica del paziente

Cliccando sul bottone della cartella clinica, il paziente viene presentato ad una nuova finestra in cui può visualizzare le informazioni della sua cartella clinica che possono essergli utili (Figura 41), ovvero le prescrizioni e le note.

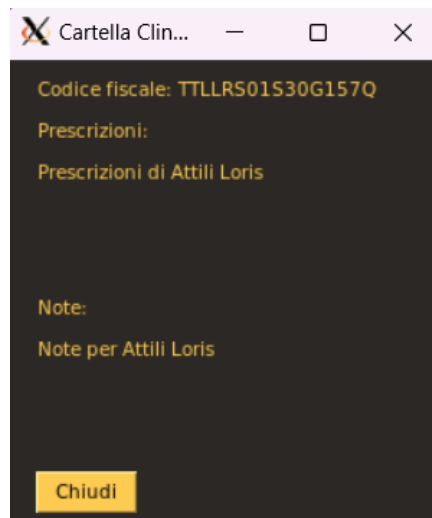


Figure 41: Finestra di visualizzazione della cartella clinica per il paziente

3.3.5 Conferma cure

Cliccando sul bottone per confermare le cure, il paziente viene presentato ad una schermata in cui deve inserire il codice fiscale del caregiver che gli ha somministrato le cure (Figura 42); una volta verificato il codice fiscale, il sistema memorizzerà l'evento con tutte le informazioni necessarie.

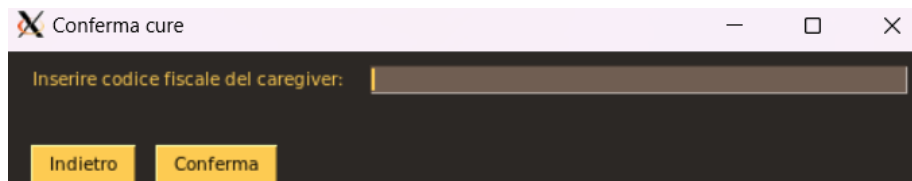
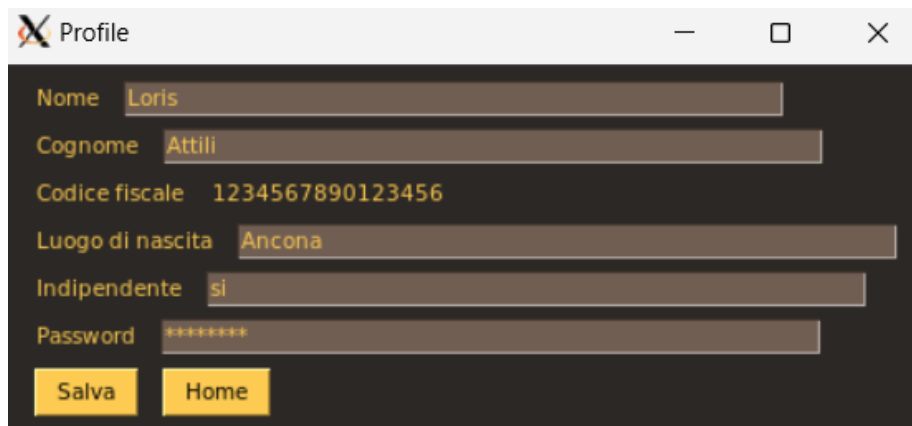


Figure 42: Finestra di inserimento del codice fiscale del caregiver che ha somministrato le cure al paziente

3.4 Modifica profilo

Dalla home del paziente, cliccando sul bottone "Modifica profilo" è possibile accedere alle generalità del paziente e modificarle, fatta eccezione per il codice fiscale (Figura 48). E' possibile anche modificare la password utilizzata per il login; per tutte le modifiche vengono utilizzati gli stessi controlli di sanificazione descritti precedentemente.



The 'Profile' window displays the following information:

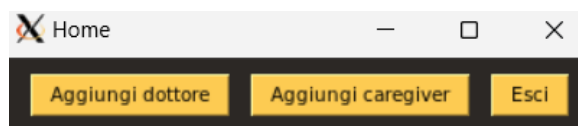
Nome	Loris
Cognome	Attili
Codice fiscale	1234567890123456
Luogo di nascita	Ancona
Indipendente	si
Password	*****

At the bottom, there are two buttons: 'Salva' and 'Home'.

Figure 43: Finestra di visualizzazione del profilo del paziente

3.4.1 Home amministratore

La registrazione di dottori e caregiver non può avvenire allo stesso modo dei pazienti, dato che hanno autorizzazioni di livello più alto; per questo motivo si è scelto di riservare la registrazione dei dottori e dei caregiver all'amministratore, il quale ha accesso a queste sole funzionalità, proprio per rispettare il principio secondo cui ogni utente deve avere accesso al minor numero di privilegi possibile. Per questi motivi la home dell'amministratore possiede, oltre al solito bottone di logout, i bottoni che rimandano ai form di registrazione di dottori e caregiver (Figura 44).



The 'Home' window for the administrator contains three buttons: 'Aggiungi dottore', 'Aggiungi caregiver', and 'Esci'.

Figure 44: Finestra home dell'amministratore

3.4.2 Registrazione di un dottore

Il form di registrazione di un dottore richiede il numero minimo di informazioni necessarie per l'utilizzo del software da parte di un dottore, l'unica nota di rilievo va fatta per la password che deve avere una complessità minima definita e deve essere inserita due volte, dato che una volta registrato un dottore se non si riuscisse a risalire alla password questo non potrebbe più utilizzare il software, salvo richiesta di un nuovo profilo all'amministratore (Figura 45).

The screenshot shows a window titled "Registra dottore" with a standard macOS-style title bar (red, yellow, and green buttons). The window has a dark background. It contains five text input fields with labels in yellow: "Nome" (containing "Loris"), "Cognome" (containing "Attili"), "Codice Fiscale", "Password" (containing seven asterisks), and "Conferma password" (containing seven asterisks). At the bottom, there are two yellow buttons: "Registra dottore" and "Annulla".

Figure 45: Finestra di registrazione di un dottore

3.4.3 Registrazione di un caregiver

La finestra di registrazione di un caregiver è analoga a quella per i dottori (Figura 46).

The screenshot shows a window titled "Registra caregiver" with a standard macOS-style title bar. The window has a dark background. It contains five text input fields with labels in yellow: "Nome", "Cognome", "Codice Fiscale", "Password", and "Conferma password". All fields are currently empty. At the bottom, there are two yellow buttons: "Registra caregiver" and "Annulla".

Figure 46: Finestra di registrazione di un caregiver

3.4.4 Home dottore

Una volta eseguito il login con le credenziali di un dottore, tale utente accede alla home per i dottori, in cui si può accedere al fascicolo di un qualunque paziente inserendo il suo codice fiscale e premendo il bottone "Ok", oppure si può accedere al proprio profilo cliccando l'omonimo bottone; infine si può eserequire il logout (Figura 47).

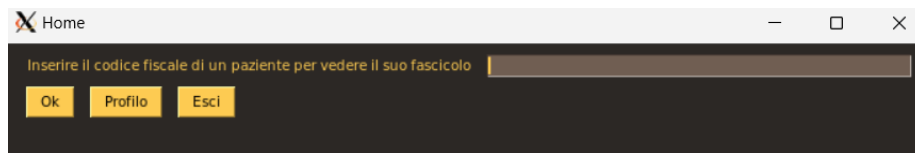


Figure 47: Finestra home del dottore

3.4.5 Profilo dottore

Nel profilo del dottore (Figura 48) si possono visualizzare e modificare il nome e il cognome, che servono solo come personalizzazione della home del dottore, mentre il codice fiscale è solo visualizzabile e non modificabile dato che viene utilizzato per identificare univocamente il dottore.

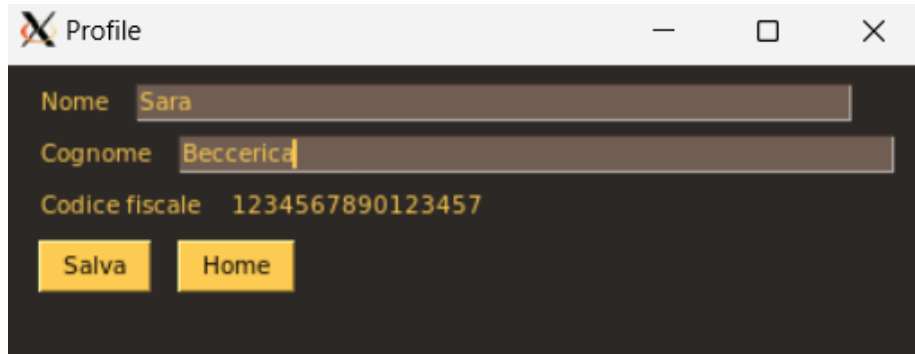


Figure 48: Finestra profilo del dottore

3.4.6 Fascicolo del paziente

Inserendo correttamente il codice fiscale di un paziente dalla home, un dottore ha accesso a tutte le informazioni di tale paziente (Figura 49); inoltre, cliccando sui relativi bottoni, il dottore può aggiornare le informazioni, aggiornando la storia clinica, modificando le prescrizioni o aggiungendo un nuovo trattamento.

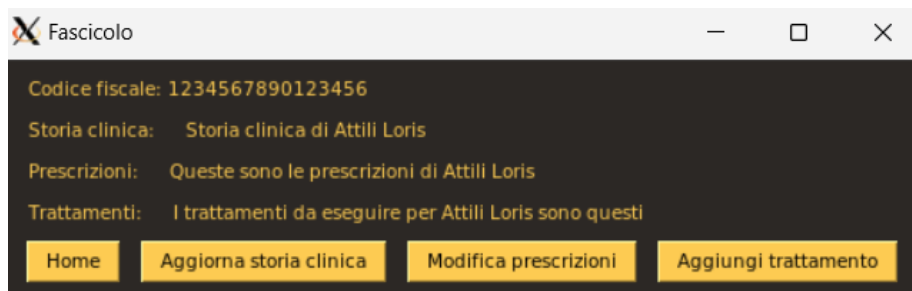


Figure 49: Finestra di visualizzazione del fascicolo di un paziente da parte del dottore

3.4.7 Aggiornamento storia clinica

Dopo aver cliccato sul bottone "Aggiorna storia clinica" si può visualizzare tutto ciò che è stato inserito precedentemente in tale campo e modificarlo (Figura 50). Vengono fatti dei controlli per evitare che la storia clinica modificata sia vuota, per evitare che ci sia una totale perdita di informazioni per un determinato paziente, cosa che non dovrebbe accadere.



Figure 50: Finestra di aggiornamento della storia clinica di un paziente

3.4.8 Modifica delle prescrizioni

Cliccando invece sul bottone "Modifica prescrizioni" dal fascicolo del paziente, il dottore accede ad una finestra analoga a quella dell'aggiornamento della storia clinica in cui possono essere aggiunte o rimosse le cure prescritte al paziente. (Figura 51).

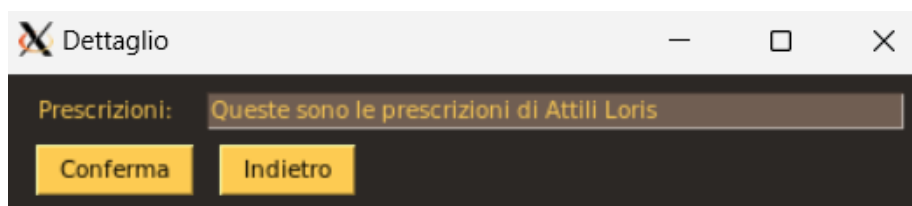


Figure 51: Finestra di aggiornamento delle prescrizioni di un paziente

3.4.9 Aggiornamento dei trattamenti

Cliccando infine sul bottone "Aggiungi trattamento" dal fascicolo del paziente, il dottore accede ad una finestra analoga a quelle precedenti in cui possono essere aggiunti o rimossi i trattamenti assegnati al paziente (Figura 52).

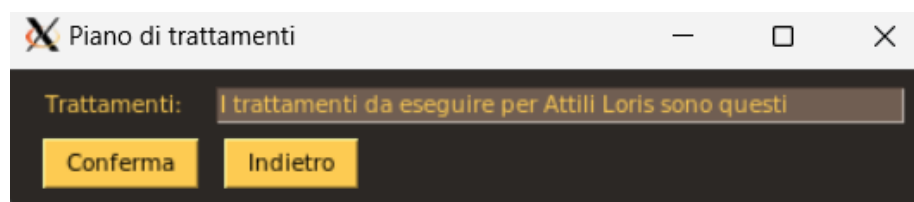


Figure 52: Finestra di aggiornamento dei trattamenti di un paziente

3.4.10 Home caregiver

Eseguendo l'accesso con le credenziali di un caregiver, viene presentata all'utente una schermata in cui è possibile inserire il codice fiscale di un paziente per eseguire le procedure legate ad esso, oppure accedere al proprio profilo (Figura 53).

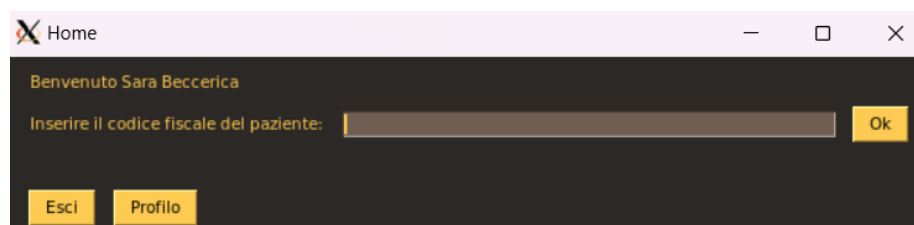


Figure 53: Finestra della home per il caregiver

3.4.11 Profilo caregiver

Nel profilo del caregiver (Figura 54), come per quello del dottore, si possono visualizzare e modificare il nome e il cognome, che servono solo come personalizzazione della home del caregiver, mentre il codice fiscale è solo visualizzabile e non modificabile dato che viene utilizzato per identificare univocamente l'utente.

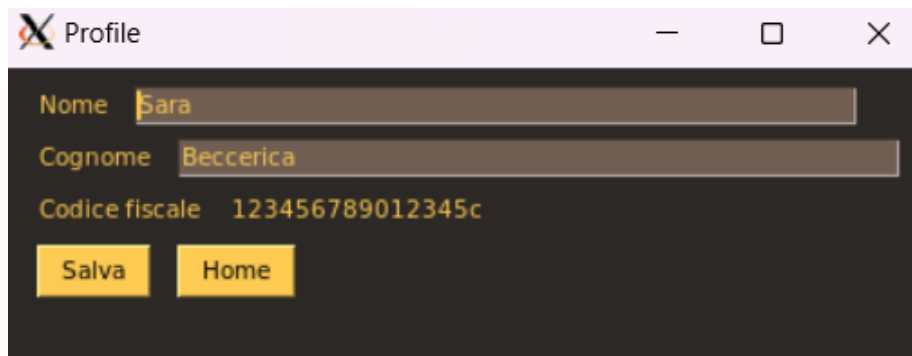


Figure 54: Finestra del profilo del caregiver

3.4.12 Fascicolo del paziente

Inserendo il codice fiscale di un paziente dalla home caregiver è possibile accedere al fascicolo medico di quel paziente (Figura 55); la finestra che mostra il fascicolo permette al caregiver di effettuare 2 operazioni: confermare le cure (disponibile solo se il paziente non è indipendente) o aggiungere delle note relative al paziente.

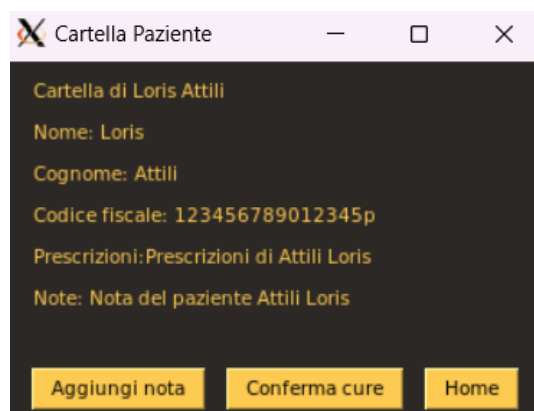


Figure 55: Finestra fascicolo di un paziente

Nel caso il paziente non sia indipendente è visualizzabile il tasto di conferma delle cure, il quale, se cliccato, informa della corretta esecuzione della procedura (Figura 56).

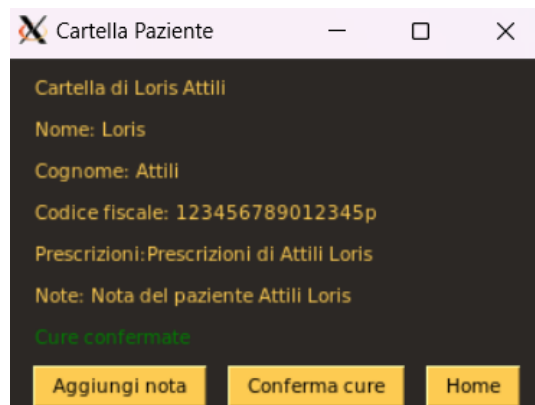


Figure 56: Notifica di corretta elaborazione della conferma

3.4.13 Inserimento di una nuova nota

Dal fascicolo del paziente, cliccando su "Aggiungi nota", il caregiver può inserire delle informazioni aggiuntive che possono essere utilizzate da lui o dal dottore per gestire in modo corretto il paziente (Figura 57).

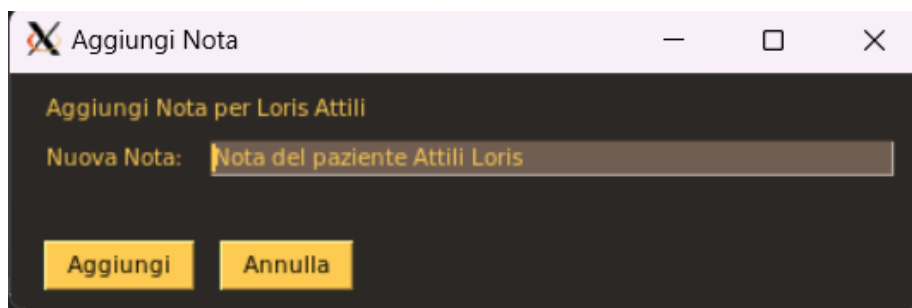


Figure 57: Finestra di inserimento di una nuova nota

3.5 Test

Il testing è una fase cruciale nello sviluppo di software, poiché garantisce la qualità, la funzionalità e l'affidabilità di un programma. Senza test adeguati, anche piccoli errori possono portare a malfunzionamenti, compromettendo la stabilità del software e la soddisfazione dell'utente finale. Il processo di testing consente di individuare e correggere difetti prima che il software venga rilasciato, riducendo i costi di manutenzione e prevenendo problemi futuri. Esistono due principali approcci al testing: il testing statico e il testing dinamico. Sebbene abbiano obiettivi comuni, questi due metodi differiscono nel modo in cui vengono eseguiti e nel tipo di informazioni che forniscono.

- Il testing statico consiste nella revisione del codice, dei documenti e delle specifiche senza eseguire effettivamente il programma. Questa tecnica mira a rilevare errori sintattici, problemi di stile e potenziali vulnerabilità analizzando il codice sorgente. Esempi di testing statico includono le revisioni manuali del codice (code review), l'uso di strumenti di analisi statica e la verifica della conformità a standard di codifica. I vantaggi del testing statico sono molteplici: permette di individuare errori precocemente, prima ancora che il codice venga eseguito, riducendo così i costi di correzione. Inoltre, aiuta a migliorare la leggibilità e la manutenibilità del codice.
- Il testing dinamico, invece, coinvolge l'esecuzione del programma per verificarne il comportamento in tempo reale. Questa tecnica valuta come il software interagisce con i dati di input, come gestisce le eccezioni e se produce i risultati attesi. Il testing dinamico include unit test, test di integrazione, test funzionali e test di accettazione. A differenza del testing statico, il testing dinamico consente di valutare la correttezza del programma in esecuzione, simulando scenari reali e identificando bug che potrebbero sfuggire a un'analisi statica. Tuttavia, richiede un ambiente di esecuzione e, in alcuni casi, può risultare più costoso e complesso.

3.5.1 Test statico

Per l'analisi statica del codice off-chain inizialmente è stato utilizzato il package "pylint" per verificare la correttezza sintattica e stilistica del codice. Per via della logica utilizzata da docker, tuttavia, risultavano dei problemi sui comandi di import, questo perché la struttura del file sistem all'interno dei container è gestita in modo differente da quella della macchina locale; questo ha fatto sì che lo strumento di analisi statica rilevasse degli errori sintattici anche dove non erano presenti. Questo ha portato a compromettere le valutazioni del codice date dall'algoritmo; per tale motivo l'analisi statica è stata eseguita analizzando manualmente la logica delle funzioni e affidandosi alle avvertenze dell'ide PyCharm, utilizzato per programmare in tutto il processo di sviluppo. Per l'analisi del codice on-chain, cioè dei contratti, sono stati utilizzati Remix Analyzer e Solhint, i quali hanno rilevato un errore relativo all'uso del costrutto "require()",

che causava una richiesta infinita di gas per la computazione delle funzioni che lo utilizzavano. Dopo alcune ricerche, tuttavia, è emerso che questo tipo di avvertimento risulta in realtà fuorviante, in quanto indica solo che il codice potrebbe sollevare un'eccezione, cosa che era stata prevista durante l'implementazione, perciò si è deciso di ignorare questo warning. Un'altra criticità sollevata dai tool di analisi è stata la similarità tra i nomi delle variabili, una pratica che può portare errori; in questo caso si è deciso di lasciare i nomi com'erano ma il suggerimento è stato applicato prestando maggiore attenzione nell'uso di tali variabili.

3.5.2 Test dinamico

Il test dinamico del codice è stato realizzato utilizzando la libreria Python "unittest", con la quale si possono creare delle classi di test in cui viene controllato il corretto funzionamento di piccoli moduli di codice o anche singole funzioni. La libreria mette anche a disposizione due metodi speciali chiamati `setUp()` e `tearDown()`, il primo viene eseguito prima di ogni test per impostare un ambiente di test, mentre il secondo viene eseguito dopo per ripulire l'ambiente. Per controllare il corretto funzionamento della comunicazione tra client e blockchain, che rappresenta il punto critico principale dell'architettura, è stata implementata una classe di test che, dopo aver creato una connessione con la blockchain ganache locale, utilizzando le variabili di ambiente, esegue le seguenti operazioni per ogni tipo di contratto:

- compila il contratto partendo dal codice Solidity ad esso associato;
- effettua il deploy del contratto all'interno della blockchain;
- controlla che la blockchain abbia confermato la ricezione del nuovo contratto e abbia istanziato correttamente un oggetto Python che funga da interfaccia tra i due moduli software.

```
✓ Tests passed: 5 of 5 tests – 345 ms

C:\Users\loris\PycharmProjects\HealthChain\venv>
Testing started at 12:58 ...
Launching unittests with arguments python -m

testing deploy of contract patient...
OK
testing deploy of contract caregiver...
OK
testing deploy of contract doctor...
OK
testing deploy of contract healthfile...
OK

Ran 1 test in 0.345s

OK

Process finished with exit code 0
```

Figure 58: Risultati del test sui contratti