UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Relazione di redazione dell'applicativo "Ticket4U"

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Corso di Software Cybersecurity A.A. 2020/2021

Baioni Francesco
Caprari David
Cingolani Cristian
Mori Nicola
Sospetti Mattia

Sommario

Introduzione e caso di studio 3

Progettazione 4

- Analisi dei requisiti 4
  - Early-Requirement Analysis 4
  - Late-Requirement Analysis 4
- Threat identification e Attack assessment 10
  - Risk and Mitigation Table 10
- Abuse e Misuse Cases in i* 15
  - Abuse Case 15
  - Misuse Case 18
- Attack Trees 19
  - Insider/Outsider Attack Tree 19
  - Clumsy Manager/Clumsy Buyer Attack Tree 20
- Abuse Case and Misuse Case Specification 21
- Risk Assessment and Mitigation Table 22
  - Mitigation: STRIDE Model 23
  - Mitigation: Abuse/Misuse Case 26

Implementazione 27

- Sistemi Utilizzati: OS e VM 27
- Infrastruttura Utilizzata: Client-Server e pacchetti NodeJS 27
- Software e Strumenti Utilizzati: Blockchain e Smart-Contract 29
- Implementazione: Smart-Contract e Blockchain 30
- Implementazione: Server e applicativo Web 32
- Testing e collaudi 37

Introduzione e caso di studio

Il progetto in esame viene realizzato per il corso di Software Cybersecurity A.A 2020/2021 previsto per il Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione presso l'Università Politecnica delle Marche.

Il progetto prende in esame il caso di studio di una biglietteria digitale online.
La generica applicazione che ne deve risultare deve poter essere in grado di supportare la generazione, modifica e cancellazione di eventi, ad esempio concerti o incontri sportivi, a cui devono essere associati dei biglietti, acquistabili, spendibili per la partecipazione all'evento. Le funzioni implementate prevedono anche l'assegnazione di un sigillo fiscale e la validazione del biglietto.

Il tutto deve essere realizzato con l'ausilio e l'utilizzo di una blockchain e relativi smart-contracts su cui registrare le eventuali transazioni.

Le soluzioni adottate per la realizzazione del suddetto sono descritte nel dettaglio di seguito.

Progettazione

Analisi dei requisiti

L'analisi dei requisiti si sviluppa in due parti principali: Early-Requirement Analysis e Late-Requirement Analysis.

Early-Requirement Analysis

In questa fase sono state discusse le linee guida da seguire. In particolare, nel caso di studio veniva richiesta la presenza di una biglietteria che ha lo scopo di rivendere i biglietti e quindi di applicare un sigillo, di un manager che possa occuparsi della gestione degli eventi e della loro validazione, di una biglietteria che possa rendere valido il biglietto apponendone un sigillo ed infine di un buyer che possa comprare i ticket per poter partecipare ad un evento. Ciascun soggetto è provvisto di un wallet, un portafogli, all'interno del quale, ad esempio, vengono registrati i ticket acquistati (buyer) o gli eventi creati (manager).

Le interazioni degli agenti tra essi e/o con il sistema devono avvenire tramite una interfaccia web appositamente studiata, sviluppata per garantire e risolvere anche i requisiti funzionali che verranno previsti dalle fasi successive del progetto. È previsto inoltre l'uso di smart-contract e dunque di una relativa blockchain per eseguire alcune delle operazioni fondamentali richieste, come l'acquisto da parte di un buyer di alcuni biglietti e la validazione di questi. Queste operazioni devono poter essere tracciate e registrate nel dettaglio, motivo che implica e suggerisce l'adozione di tale architettura.

Late-Requirement Analysis

1)SD/SR i model*

Di seguito, si è passati alla Late-Requirement Analysis che prevede la messa a terra dei concetti precedentemente discussi.

A tal fine, si è utilizzato il modello *i** che permette di specificare quali sono gli attori principali, le risorse che devono essere usate o ottenute da un attore, i task che devono essere svolti, i softgoal ed i goal che devono essere perseguiti. È possibile, inoltre, indicare relazioni e dipendenze fra questi elementi utilizzando i costrutti grafici specifici del modello.

Dopo diversi incontri e sessioni di confronto, il gruppo ha scelto di ottenere quattro principali attori: Manager, Buyer, Reseller e Sistema.

Il Manager deve avere la possibilità di gestire gli eventi, dunque di crearli, modificarli e cancellarli. L'evento creato deve essere registrato all'interno del Manager's Wallet e a partire da questo vengono creati i relativi ticket. Il manager deve poter validare i biglietti acquistati da un buyer andando a simulare l'ingresso autorizzato da parte di quest'ultimo all'evento.

L'attore Buyer può acquistare i biglietti di un evento accedendo al sito web e registrando l'acquisto all'interno del proprio wallet.

La biglietteria (Ticket Reseller), invece, per emettere i biglietti deve poter applicare il sigillo (simil SIAE) a ciascuno di essi.

Infine, nel diagramma SD/SR i* mostrato di seguito è presente l'attore Credit Card Manager che non è di particolare interesse per questa analisi, visto che l'implementazione non verrà trattata nelle fasi successive, ma è stato inserito per completezza in quanto in un sistema reale usualmente la progettazione prevede una sezione relativa alla gestione dei pagamenti.

La maggior parte delle funzioni descritte in precedenza sono messe a disposizione dei vari attori da parte del Sistema che rappresenta dunque il core del modello e parte dell'applicativo.

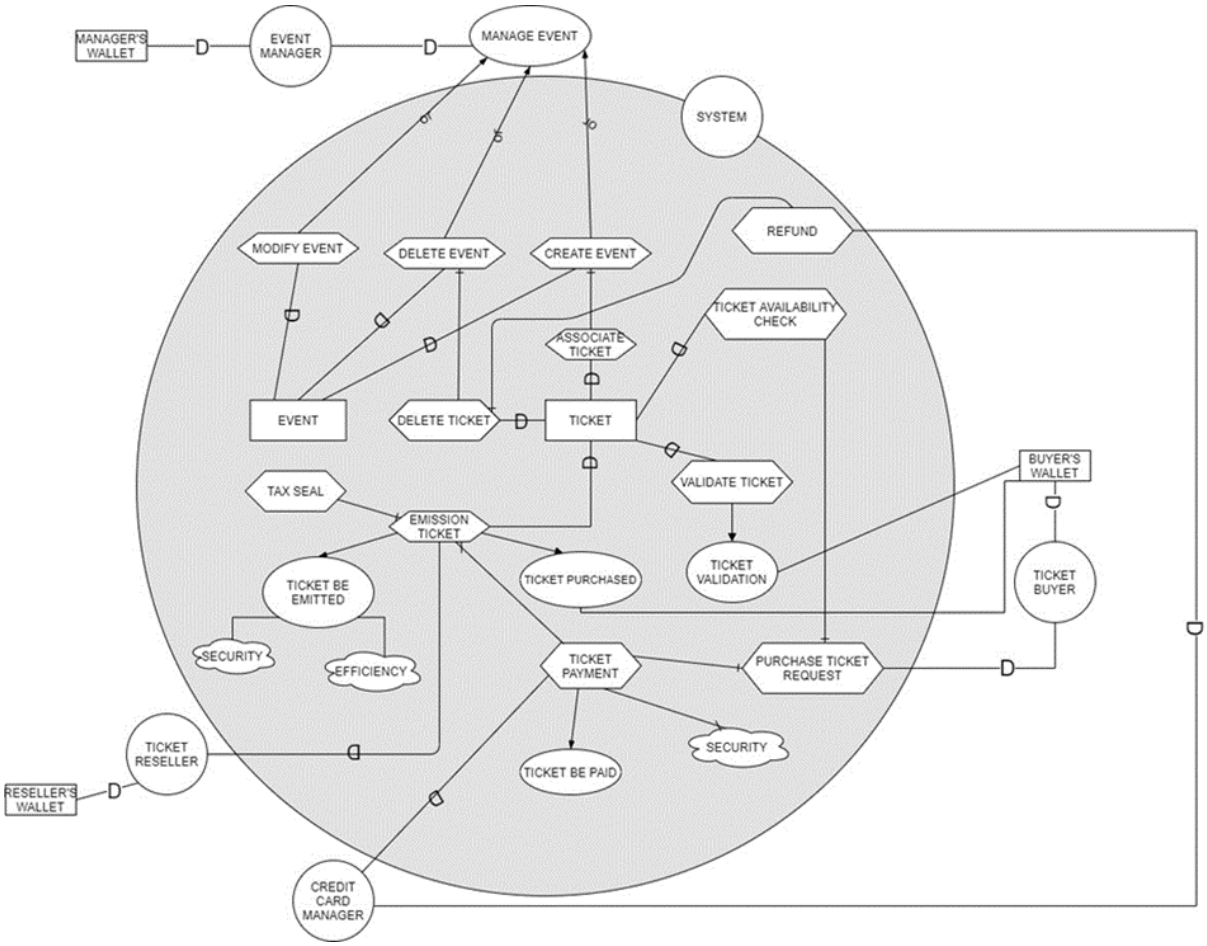

Fig. 1: Modello SD/SR i di prima definizione in cui sono presenti i principali attori del caso in esame e la definizione delle attività e dei task che questi svolgono.*

Sono stati poi individuati gli asset che rivestono un ruolo importante e di più spessore all'interno del progetto. In particolare, sono stati selezionati: la risorsa Manager's Wallet, in quanto all'interno di esso vengono registrati gli eventi creati; il goal Manage Event, in quanto sintetizza le principali azioni che può svolgere il manager ovvero, creazione, modifica e cancellazione di un evento; la risorsa Event che rappresenta l'asset principale attorno al quale ruota un progetto che ha come fine la gestione di eventi; la risorsa Ticket, su cui possono essere applicate azioni di fondamentale importanza come l'acquisto da parte del buyer, la validazione da parte del manager e l'emissione da parte della biglietteria; il task Ticket Payment che seppur non venga gestito in questo progetto è fondamentale qualora dovesse essere previsto il pagamento del biglietto da parte del buyer; la risorsa Buyer's Wallet, altrimenti noto come portafoglio dell'utente, il quale contiene il totale dei biglietti acquistati e quindi di fondamentale importanza al fine di garantire un servizio sicuro.

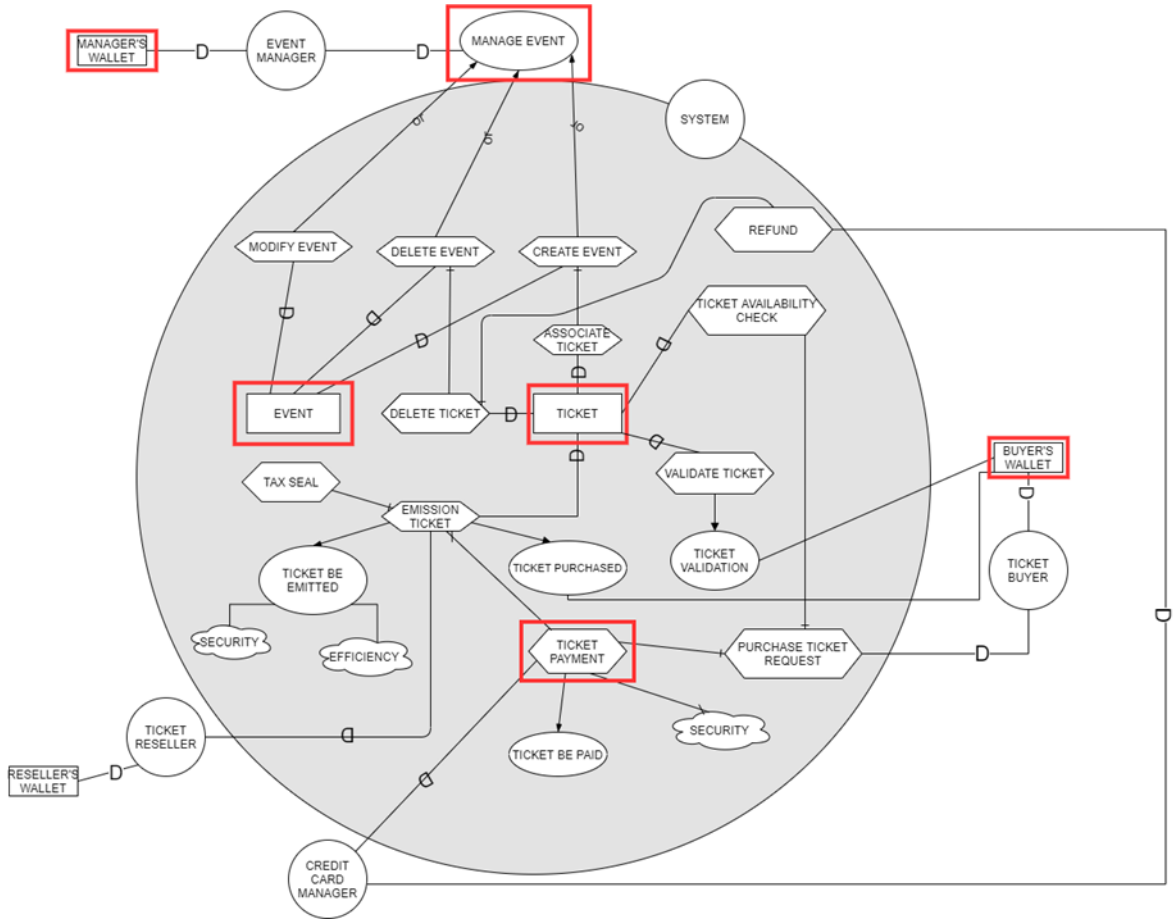

Fig. 2: Al modello precedente viene aggiunta l'individuazione degli asset principali.

2)Use case specification

Per ciascun asset individuato è stata creata una tabella nella quale sono state descritte le specifiche. Ad esempio, per l'asset Manager's Wallet è stato assegnato un ID, sono stati specificati il Manager ed il Sistema come attori coinvolti etc. Appare chiaro come questo asset venga utilizzato qualora il manager decida di eseguire azioni riguardanti la gestione degli eventi. Quest'ultima possibilità, però, è limitata ai soli eventi creati in precedenza dallo stesso manager.

Use case ID:	U-MW-01
Use case name:	Manager's Wallet Utilization
Actors	Manager, System
Description	Manager's Wallet use for ticket registration
Data	Manager's Wallet
Stimulus and Preconditions	Events are registered by managers, system assigns ticket to managers' IDs
Basic Flow	1. Manager manages events (create, delete, modify)
	2. System updates Manager's Wallet with ticket ID
Alternative Flow	
Exception Flow	1. Manager manages events (create, delete, modify)
	2. Wallet is not updated
Response and Postconditions	Confirmation that tickets have been assigned
Non functionals Requirements	CIA
Comments	

Fig. 3: Definizione di esempio dell'annotazione di un caso d'uso

3)Policy

Dopodiché, sono state individuate le policies che devono essere rispettate per ciascun asset. In particolare, sono state largamente utilizzate le policies CIA e AAA.

Prendendo sempre come esempio il Manager's Wallet, devono essere rispettate: confidenzialità, ovvero ai dati presenti in questo wallet non devono avere accesso figure come Buyer o Reseller o attaccanti esterni; integrità, la quale garantisce l'impossibilità di modificare o distruggere i dati in modo improprio; availability, ovvero la possibilità da parte del Manager di accedere ai dati presenti nel proprio wallet in qualsiasi momento.

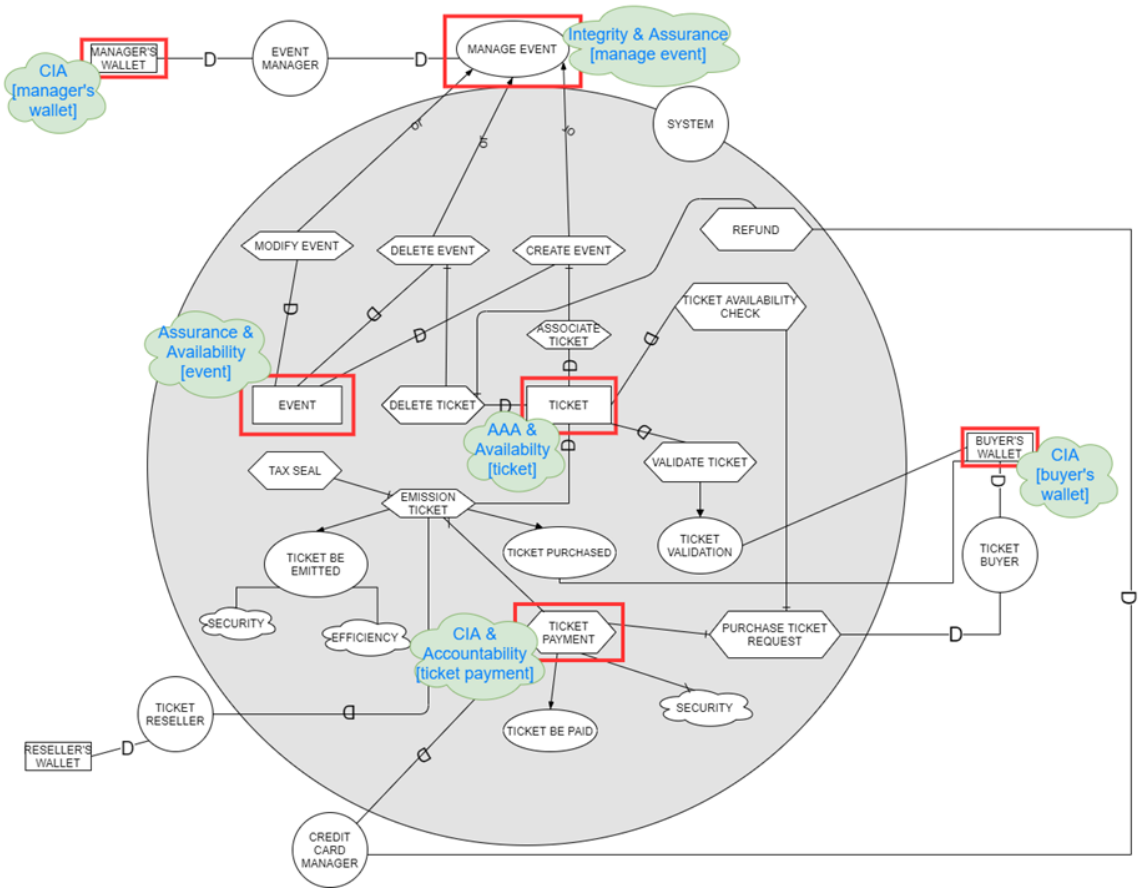

Fig. 4: Inserimento delle policy da mantenere nel modello principale

4)Asset value / Exposure assessment

La Late-Requirement Analysis termina con l'assegnazione per ciascun asset del valore e del livello di esposizione. In questo caso è stata usata una scala di Likert da uno a cinque ed è stata inserita una breve descrizione che motivi tale scelta.

Ad esempio, a *Manage Event* viene assegnato un valore quattro per entrambi i campi in quanto la gestione di eventi ha gran valore per il sistema ed un violazione o una problematica con questo asset produrrebbero altre problematiche a cascata.

ASSET	VALUE	VALUE DESCRIPTION	EXPOSURE	EXPOSURE DESCRIPTION
Manager's Wallet	3	However, it has its own relevance but does not have priority over other assets	3	Interference with ticket sales, organization problems
Manage Event	4	Event management plays a crucial role in the system	4	Deleting or modifying an event affects the entire system
Event	4	The event contains general information such as place, date, available seats ...	3	Changing the parameters of an event affects the entire system
Ticket	5	It has the highest priority for the centrality of the object	4	By modifying the parameters of a ticket, you act on the entire system and directly on the customer, affecting the division of the system and the organizers of the event
Ticket Payment	4	It affects both the buyer and the ticket	5	The presence of money makes this an important factor and contains sensitive data
Buyer's Wallet	3	However, it has its own relevance because it contains the ticket but does not have priority over the other assets	2	Loss of reputation due to customer dissatisfaction

Fig. 5: Dettaglio della Asset Table in cui vengono definiti gli asset principali e viene assegnato un valore in scala di Likert ad ognuno di questi

Terminata la fase di definizione delle policy da utilizzare e gli asset da considerare di valore si passa all'analisi dei rischi, alla definizione di eventuali attacchi ed alle fasi successive di progettazione.

Threat identification e Attack assessment

Risk and Mitigation Table

In seguito alla fase di valutazione degli asset e del loro grado di esposizione ci è concentrati nell'analizzare le possibili minacce che potevano andare a violarne le caratteristiche di sicurezza. Il gruppo ha deciso di effettuare questo task usufruendo della metodologia STRIDE, ovvero sono state classificate le minacce in sei tipologie, ognuna delle quali va a violare un requisito:

- **S**poofing (violazione dell'Authenticity)
- **T**ampering (violazione dell'Integrity)
- **R**epudiation (violazione della Non-repudiation)
- **I**nformation Disclosure (violazione della Confidentiality)
- **D**enial of Service (violazione dell'Availability)
- **E**levation of Privilege (violazione dell'Authorization)

A seconda dell'attacco individuato, nella tabella sono state spuntate le violazioni che esso provoca. Il task è stato svolto in collettivo, così facendo è stato possibile individuare minacce da diversi punti di vista cosicché lo studio non fosse limitato dalla visione di un'unica persona. Di seguito la Risk e Mitigation Table utilizzando il modello STRIDE per individuare le minacce:

ASSET	Spoofing	Tampering	Repudiation	Information Disclosure	Dos	Elevation of Privilege	Attack
Manager's Wallet				X			Intercept
	X	X		X			Unauthorized Access
	X	X		X		X	Vertical Privilege Escalation
		X			X		Denial of service
Event	X	X				X	Privilege Escalation
		X			X		Denial of Service
	X	X					Unauthorized Access
	X	X					Horizontal Privilege Escalation(
		X	X				Accidental Change or Deletion(
	X	X				X	Vertical Privilege Escalation(Clu
Ticket	X	X		X		X	Vertical Privilege Escalation
		X			X		Denial of Service
	X	X		X		X	Vertical Privilege Escalation(Clu
			X				Overbuying(Clumsy Buyer)
	X	X		X			Horizontal Privilege Escalation(Clumsy Manager)
		X	X				Accidental Change or Deletion(Clumsy Manager)
	X	X		X		X	Vertical Privilege Escalation(Clu
Buyer's Wallet				X			Interception
	X	X		X			Unauthorized Access
	X	X		X			Horizontal Privilege Escalation
		X			X		Denial of Service
System	X	X		X		X	Vertical Privilege Escalation
		X			X		Denial of Service

Fig. 6: Dettaglio della Risk Assessment and Mitigation Table in cui ad ogni asset viene associato una possibile vulnerabilità[1]

Piccola nota a venire: nella trattazione successiva, quando si riporteranno esempi che hanno a che fare con la violazione di privilegi verranno fatti spesso con la figura del manager; è ovvio che nel nostro caso di studio anche la biglietteria ha privilegi superiori a quelli dell'utente generico, ma per semplicità di esempi verrà omessa dalla trattazione in questa fase.

[1] Tabella completa nel file excel allegato nel foglio con titolo "RiskAssessmentandMitigationTable".

1) Manager's Wallet

Intercept: Il manager wallet può essere violato con un attacco Intercept che si verifica nel caso in cui utenti non autorizzati hanno accesso alle informazioni del manager, nel caso di studio in particolare possono essere recuperate informazioni come l'e-mail, password o eventi a cui l'utente parteciperà. La violazione è Information Disclosure ovvero potrebbero rese note involontariamente queste informazioni ad utenti non autorizzati.

Unauthorized Access**:** Si verifica quando un utente non autorizzato accede come Manager o Biglietteria. Le violazioni commesse sono: Spoofing, ovvero qualcuno potrebbe identificarsi come un utente che in realtà non è modificandone i dati sensibili; Tampering perché potrebbero essere manomessi i dati scambiati tra Client e Server modificando i permessi che l'utente ha; Information Disclosure poiché un accesso non autorizzato andrebbe a rivelare informazioni sensibili di un manager.

Vertical Privilege Escalation**:** Il manager potrebbe avere più potere di quanto già ne abbia e di conseguenza potrebbe operare con autorizzazioni di sistema, oppure viceversa un Utente normale potrebbe scalare i privilegi e eseguire operazioni solamente accessibili dal manager. Le violazioni sono simili all'unauthorized access con l'Elevation of Privilege ovvero potrebbero essere dati dei privilegi eccessivi per il tipo di utente indicato.

Denial of Service**:** Eventuali operazioni, manomissioni, che vengono eseguite tra Client e Server a livello di manager potrebbero provocare l'arresto del sistema, a causa di una errata configurazione. Le violazioni che vengono commesse sono Tampering e DoS.

2) Event

Privilege Escalation**:** potrebbero avvenire attacchi che consentono ad utenti che hanno un livello di privilegi più basso di effettuare operazioni sull'Evento non consentite. Quindi le violazioni sono Spoofing, fingersi un altro utente per effettuare delle operazioni non consentite, Tampering, manomettere i dati per cercare di effettuare queste operazioni ed avere maggiori privilegi sull'Evento, Elevation of Privilege, si accede a dei privilegi che normalmente un utente non dovrebbe avere rispetto all'evento.

Denial of Service**:** Eventuali operazioni, manomissioni che vengono eseguite tra Client e Server a livello di evento potrebbero provocare l'arresto del sistema, a causa di una errata configurazione. Le violazioni che vengono commesse sono Tampering e DoS.

Unauthorized Access**:** Utenti non autorizzati potrebbero accedere alla risorsa Event, eseguendo operazioni non autorizzate poiché è stato effettuato un accesso da parte di un utente fingendosi un altro, questo con un livello di privilegi che consente la violazione dei dati nell'asset. Attraverso lo Spoofing e il Tampering è possibile effettuare questo tipo di attacco.

Horizontal Privilege Escalation**:** un manager non proprietario dell'evento accidentalmente modifica i dati contenuti in un evento creato da un altro manager, anche qui potrebbero avvenire attacchi come lo Spoofing e il Tampering ovvero, per lo Spoofing un manager potrebbe identificarsi come un altro manager per accedere a quei dati, Tampering potrebbero essere manomessi i dati di quell'evento da un manager non autorizzato.

Accidental Change o Deletion**:** possono essere accidentalmente effettuate operazioni di modifica o eliminazione degli eventi da parte del manager che ha eccesso a quell'evento. Le violazioni commesse sono Tampering, quindi modifica dei dati o anche eliminazione, Repudiation ovvero qualora avvenisse un attacco di questo tipo non si riesce a risalire all'attaccante.

Vertical Privilege Escalation**:** un utente a cui non è consentito effettuare determinate operazioni sull'evento riesce ad eseguirle. Di norma un utente esterno o un utente di basso livello potrebbe andare a modificare, eliminare informazioni su di esso. Avviene se c'è una violazione di Elevation of Privilege, Tampering, ovvero vengono manomesse informazioni, e Spoofing ovvero utenti che si falsificano come manager eseguono operazioni sull'evento.

3) Ticket

Vertical Privilege Escalation: un utente a cui non è consentito effettuare determinate operazioni sul Ticket riesce ad eseguirle. Di norma un utente esterno o un utente di basso livello potrebbe andare a modificare, eliminare informazioni su di esso. Avviene se c'è una violazione di Elevation of Privilege, Tampering, ovvero vengono manomesse informazioni, e Spoofing ovvero utenti che si falsificano come manager eseguono operazioni sull'evento, Information Disclosure ovvero se di un biglietto viene rivelato l'utente che l'ha comprato.

Denial of Service**:** Eventuali operazioni, manomissioni che vengono eseguite tra Client e Server a livello di ticket potrebbero provocare l'arresto del sistema, a causa di una errata configurazione. Le violazioni che vengono commesse sono Tampering e DoS.

Vertical Privilege Escalation (Clumsy Buyer)**:** un utente a cui non è consentito effettuare determinate operazioni sul Ticket riesce ad eseguirle, in modo casuale e non volontario semplicemente navigando all'interno dell'applicazione e riuscendo ad eludere i controlli e i privilegi. Di norma un utente esterno o un utente di basso livello potrebbe andare a modificare, eliminare informazioni su di esso. Avviene se c'è una violazione di Elevation of Privilege, Tampering, ovvero vengono manomesse informazioni, e Spoofing ovvero utenti che si falsificano come manager eseguono operazioni sull'evento, Information Disclosure ovvero se di un biglietto viene rivelato l'utente che l'ha comprato.

Overbuying (Clumsy Buyer)**:** un utente accidentalmente compra un numero troppo elevato di biglietti, sprecando un numero di soldi elevato e compromettendo la disponibilità della risorsa per altri utenti. La violazione è la Repudiation, ovvero potrebbe non essere possibile risalire all'utente che ha effettuato questa operazione.

Horizontal Privilege Escalation (Clumsy Manager)**:** un manager che non ha accesso al ticket modifica i dati dei ticket di altri utenti in particolare di altri manager. Anche qui potrebbero avvenire attacchi come lo Spoofing e il Tampering ovvero, per lo Spoofing un manager potrebbe identificarsi come un altro manager per accedere a quelle informazioni, Tampering potrebbero essere manomessi i dati di quel ticket da un manager non autorizzato andandolo ad assegnare a sé stesso, Information Disclosure ovvero vengono rivelate informazioni sensibili ad un manager che non deve avere accesso a quel ticket.

Accidental Change o Delection (Clumsy Manager)**:** un manager potrebbe modificare i dati di un ticket o anche eliminarlo involontariamente, rendendo il ticket non più valido o non più disponibile. Violazioni che vengono eseguite per questo tipo di attacco Tampering, manomissione delle informazioni e dei dati, Repudiation, non ci sono informazioni su chi ha compiuto questa operazione.

Vertical Privilege Escalation (Clumsy Manager)**:** un manager a cui non è consentito effettuare determinate operazioni sul Ticket riesce ad eseguirle, in modo casuale e non volontario semplicemente navigando all'interno dell'applicazione e riuscendo ad eludere i controlli e i privilegi. Avviene se c'è una violazione di Elevation of Privilege, Tampering, ovvero vengono manomesse informazioni, e Spoofing ovvero utenti che si falsificano come manager eseguono operazioni

sull'evento, Information Disclosure ovvero se di un biglietto viene rivelato l'utente che l'ha comprato.

4) Buyer's Wallet

Intercept: Il buyer wallet può essere violato con un attacco Intercept, ovvero dove utenti non autorizzati hanno accesso alle informazioni del buyer, nel nostro caso in particolare possono essere recuperate informazioni come l'e-mail, password o eventi a cui parteciperà. La violazione è Information Disclosure ovvero potrebbero rese note involontariamente queste informazioni ad utenti non autorizzati.

Unauthorized Access**:** Si verifica quanto un utente non autorizzato accede come Buyer. Le violazioni commesse sono Spoofing, ovvero qualcuno potrebbe identificarsi come un utente che in realtà non è modificando i dati, Tampering perché potrebbero essere manomessi i dati scambiati tra Client e Server modificando i permessi che l'utente ha, Information disclosure poiché un accesso non autorizzato andrebbe a rivelare informazioni sensibili di un utente.

Vertical Privilege Escalation**:** Il Buyer potrebbe avere più potere di quanto già ne ha e di conseguenza potrebbe operare a livello Manager o System, oppure viceversa un utente che non ha effettuato l'accesso potrebbe scalare i privilegi e eseguire operazioni solamente accessibili dal Buyer. Le violazioni sono simili all'unauthorized access con l' Elevation of Privilege ovvero potrebbero essere dati dei privilegi eccessivi per il tipo di utente indicato.

Denial of Service**:** Eventuali operazioni, manomissioni che vengono eseguite tra Client e Server a livello di buyer potrebbero provocare l'arresto del sistema, a causa di una errata configurazione. Le violazioni che vengono commesse sono Tampering e DoS.

5) System

Vertical Privilege Escalation**:** Il System è l'utente che ha i privilegi più alti di tutti e quindi ha accesso alle operazioni più sensibili. Utenti di livello più basso potrebbero accedere a queste funzionalità provocando un aumento dei privilegi. Le violazioni commesse sono Spoofing, ovvero qualcuno potrebbe identificarsi come System ma in realtà non lo è eseguendo operazioni non consentite, Tampering perché potrebbero essere manomessi i dati e informazioni accessibili solamente dal System, Information disclosure poiché un accesso non autorizzato andrebbe a rivelare informazioni sensibili di un utente, Elevation of Privilege.

Denial of Service**:** Eventuali operazioni, manomissioni che vengono eseguite tra Client e Server a livello di System potrebbero provocare l'arresto del sistema, a causa di una errata configurazione. Le violazioni che vengono commesse sono Tampering e DoS.

Questa fase è stata di particolare importanza poiché ha permesso di evidenziare in maniera schematica quali fossero le possibili minacce e i legami di quest'ultime ad ogni asset, così facendo è stata facilitata notevolmente la discussione successiva sulle tecniche di difesa da adottare per proteggere il sistema da eventuali attacchi.

Abuse e Misuse Cases in i*

Una volta completata la fase di individuazione delle possibili vulnerabilità, ci si è spostati sul modello i*, che era momentaneamente lasciato in sospeso, ponendo l'attenzione sugli *Abuse Case* e *Misuse Case*.

Abuse Case

Partendo dagli Abuse Case, dopo aver individuato quali sono le interazioni del sistema, i principali asset e le policy da adottare per ogni asset nell'i*, sono stati valutate quali potessero essere le opportunità che un utente malintenzionato aveva di influenzare negativamente il funzionamento del sistema. Dall'immagine si può notare che abbiamo suddiviso gli Abuse Case in 3 diversi possibili attori che possono eseguirli ovvero l'Outsider Attacker e l'Insider Attacker, suddiviso nei due tipi Manager o Buyer.

1) Outsider Attacker

Manager's Wallet

L'outsider attacker può compiere violazione di Availability, Integrity e Confidentiality rispetto al Manager's Wallet. La disponibilità può essere violata se al manager non viene reso più disponibile l'utilizzo dell'account, se si vanno a violare le informazioni e i dati del manager compromettendo quindi password e e-mail per l'accesso, infine si può violare la Confidenzialità ovvero i dati del manager possono essere letti da parti non autorizzate e quindi resi pubblici.

Event

L'outsider attacker può violare l'asset Event, quindi compromettere una risorsa cruciale per l'intera applicazione. Le violazioni indicate sono Integrity, Availability e Assurance. Un attaccante esterno potrebbe rendere nascosto l'evento e quindi non permettere agli utenti di visualizzarlo e di conseguenza potrebbe venire a sapere quali sono le informazioni legate ad esso; potrebbe anche andare a modificare o eliminare l'evento o i dati legati ad esso andando a comprometterne l'integrità; potrebbe anche andare ad effettuare operazioni errate sull'evento compromettendo l'Assurance. Le violazioni sull'Evento che abbiamo indicato vanno anche a comprendere l'asset *Manage Event* che quindi può ripercuotersi sull'evento stesso.

System

L'outsider attacker potrebbe andare a violare le policy stabilite a livello di sistema violando Accountability, Assurance, Availability e Authenticity. Può andare a compromettere l'intero sistema andando a violare l'Availability, può avere intenzione di eseguire azioni non consentite quindi violare l'Assurance, potrebbe non essere tracciato nel mentre effettua operazioni a livello di sistema, quindi l'Accountability, ed infine può violare l'Authenticity ovvero che sia lui ad operare falsamente come un altro utente.

Ticket

L'outsider attacker potrebbe compromettere anche i dati relativi ai ticket, potrebbe compromettere i biglietti disponibili e quindi violare l'Availability dei ticket, potrebbe effettuare operazioni non consentite sulla risorsa e quindi violare l'Assurance, potrebbe effettuare operazioni sull'asset senza essere tracciato e quindi verrebbe violata l'Accountability ed infine potrebbe associare un biglietto ad una persona o account diverso violando l'Authenticity.

Buyer's Wallet

Parallele alle violazioni che vengono fatte per il Manager's Wallet.

2) Insider Attacker Manager

Gli attacchi che può effettuare un manager dall'interno sono molto simili a quelli dell'outsider attacker con la differenza che non può andare ad attaccare un Buyer's Wallet. La differenza sostanziale è che un utente che già ha effettuato l'accesso come manager avrà dei vantaggi per attaccare in quanto ha già ottenuto privilegi maggiori.

3) Insider Attacker Buyer

Gli attacchi che può effettuare un manager dall'interno sono molto simili a quelli dell'outsider attacker con la differenza che non può andare ad attaccare un Manager's Wallet. Qui un utente che ha effettuato l'accesso potrebbe avere delle facilitazioni data la sua posizione per attaccare il sistema.

Questi tipi di valutazione sono state fatte da tutti i componenti del progetto in modo tale che si individuassero tutte le possibili azioni che potevano essere eseguite, così da avere più punti di vista e allinearsi su quali fossero le vere problematiche per ogni asset.

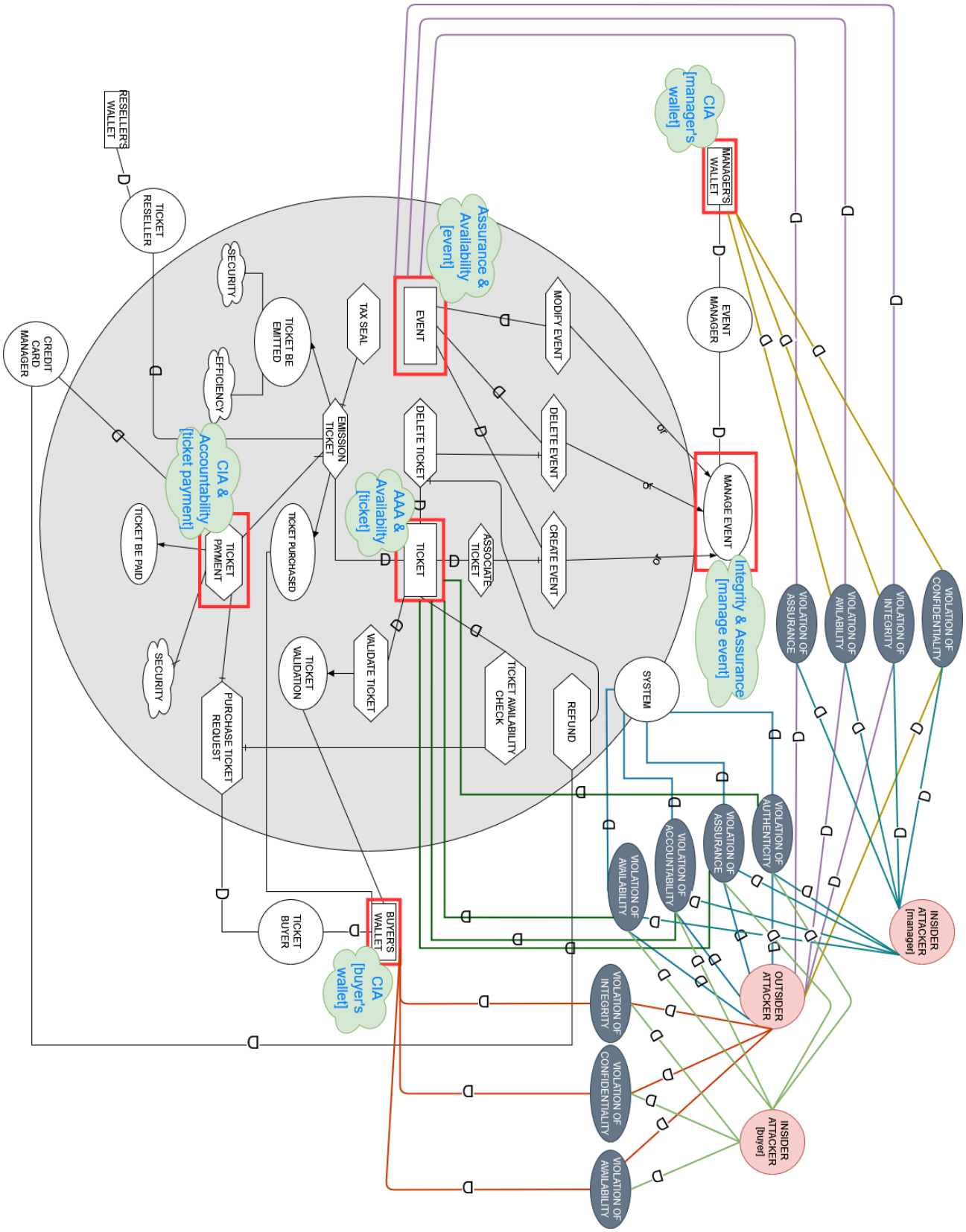

Fig. 7: Al modello i principale vengono aggiunte le possibili violazioni dovute agli Abuse Case*

Misuse Case

Per quanto riguarda i Misuse Case sono state individuate le figure di Clumsy Manager e Clumsy Buyer. Il **Clumsy Manager,** ad esempio, può compiere violazioni di integrità associate sia all'evento che al biglietto poiché possono essere modificate informazioni e dati accidentalmente su questi due Asset compromettendo l'integrità dello stesso. In aggiunta a questo, si possono compiere violazioni di Disponibilità sia, anche qui, per l'evento e il ticket, dal momento che si potrebbe rendere l'evento non visibile agli altri utenti o comprare un numero eccessivo di biglietti, sia modificare il numero di biglietti disponibili. Il **Clumsy Buyer,** invece, potrebbe comprare un numero troppo elevato di biglietti andando a violare ancora l'Availability, in modo tale che altri utenti poi non possano comprare biglietti a loro volta perchè non più disponibili.

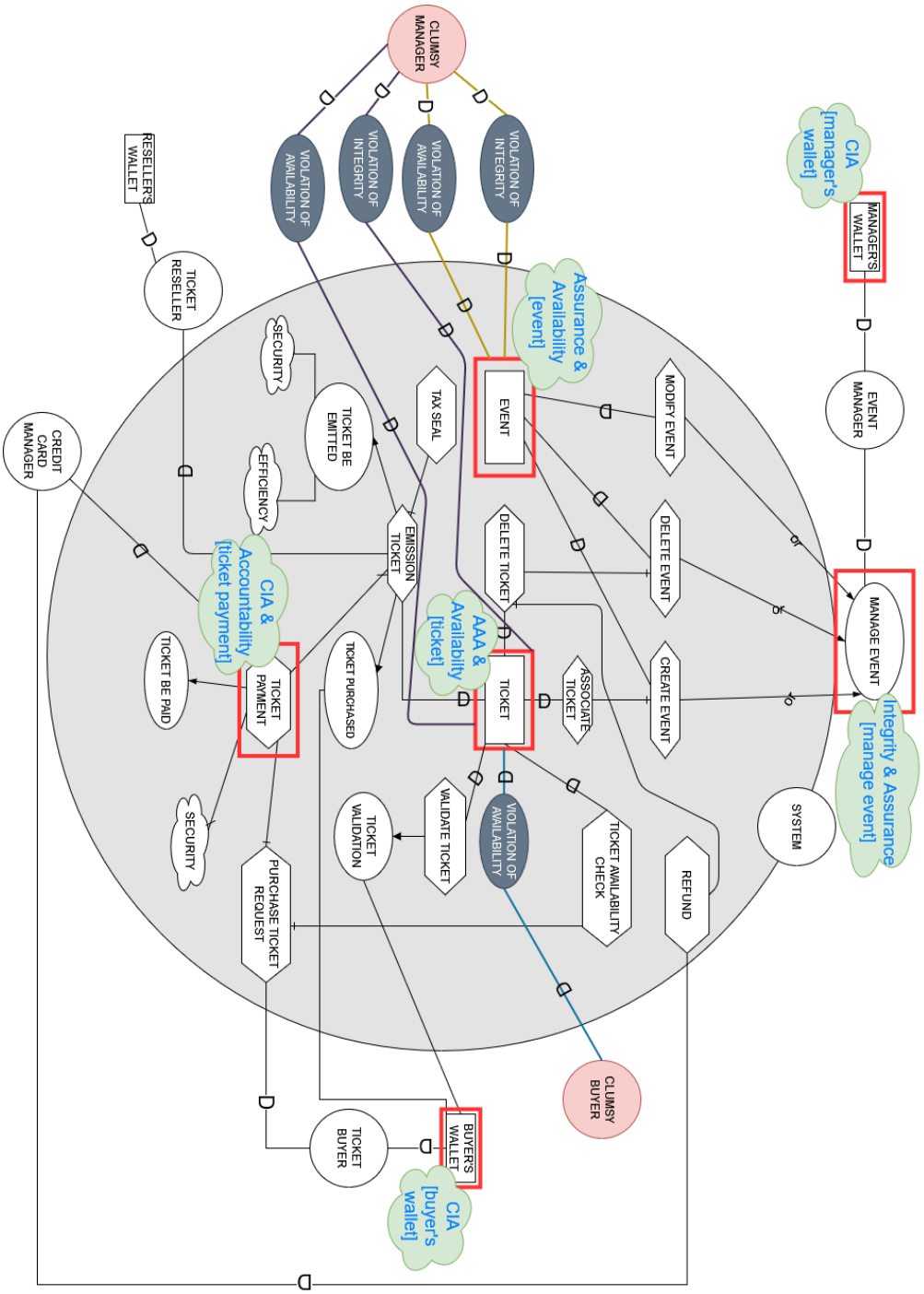

Fig. 8: Al modello i principale vengono aggiunte le possibili violazioni dovute ai Misuse Case*

Attack Trees

Dopo aver aggiornato con i misuse case e gli abuse case i diagrammi i*, ci si è concentrati sugli Attack Trees per descrivere le minacce che gravano sul sistema e possibili attacchi per realizzare tali minacce.

Insider/Outsider Attack Tree

È stato costruito un albero di attacco dove partendo dalla base, possibile attacco, si giunge alla foglia in cui viene indicata la violazione commessa, passando attraverso diversi stadi di attacco.

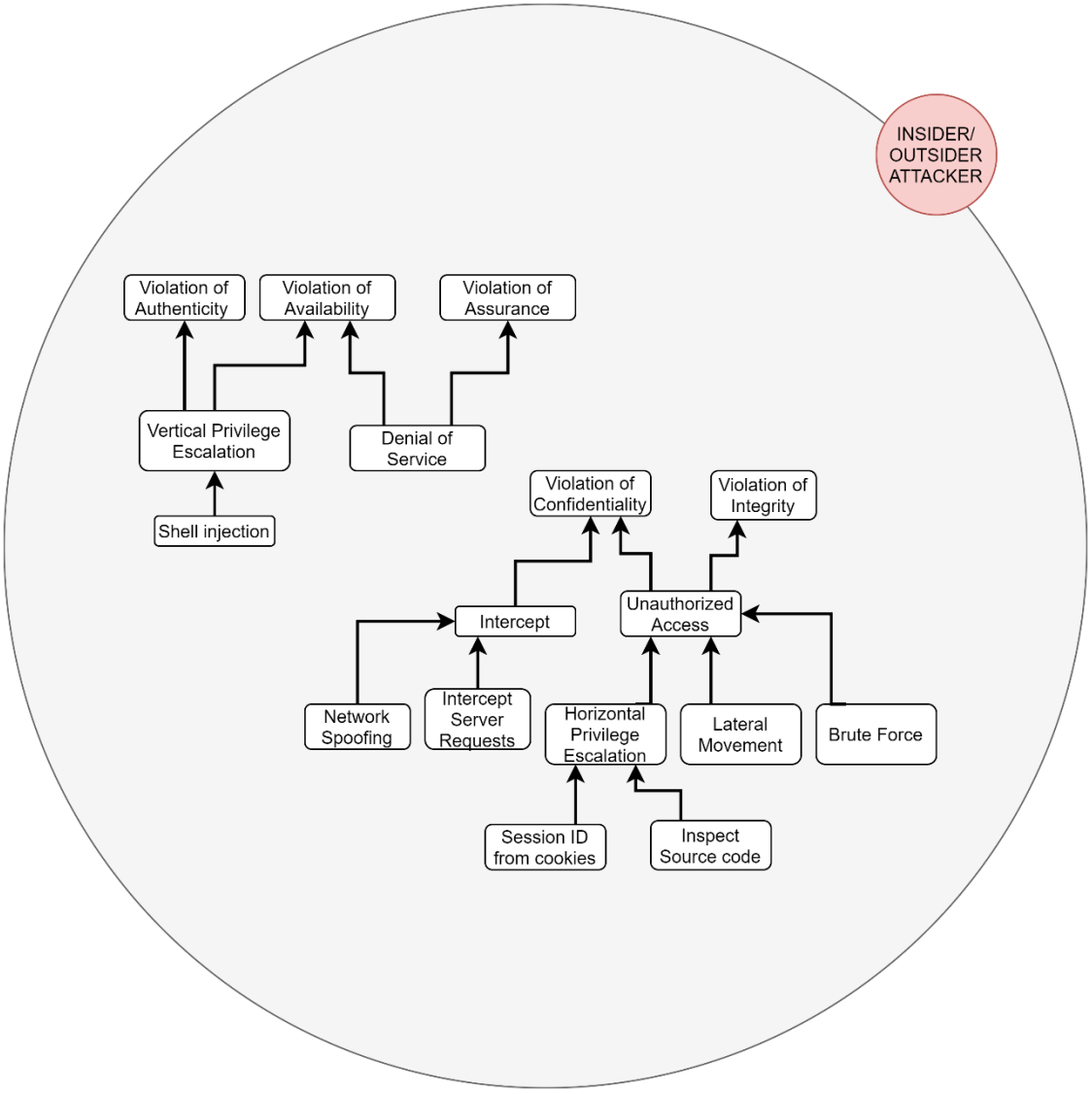

Fig. 9: Modello dell'Insider/Outsider Attack Tree

Shell injection: un utente malintenzionato può utilizzare una reverse shell per inviare e far eseguire comandi al server, sviluppando un'escalation dei privilegi sulla macchina e la capacità di caricare, eliminare, scaricare ed eseguire file da e verso il server Web. Questo porterebbe ad una violazione di *Authenticity* e *Availability* nei confornti dell'applicazione.

Denial of Service**:** un attaccante potrebbe provocare malfunzionamenti in cui si fa esaurire le risorse del web server, fino a renderlo non più in grado di erogare il servizio ai client richiedenti, provocando quindi una violazione dell'*Availability* e dell'*Assurance*.

Network Spoofing/Intercept Server Request**:** vengono inviati pacchetti a qualcuno indicando un indirizzo di ritorno non corretto o semplicemente vengono sniffate informazioni sensibili dal traffico dati (Network Spoofing); in alternativa, può avvenire un'intercettazione ad una richiesta lato server (Intercept Server Request), così facendo l'attaccante può accedere a informazioni e dati privati, violando la *Confidentiality*.

Session ID from cookies/Inspect Source Code**:** potrebbe essere recuperato l'ID dai cookies come le informazioni di sessione, oppure potrebbero essere lette informazioni investigando nel codice sorgente del server, provocando problemi a livello di privilegi orizzontali, il che implica un accesso non autorizzato, violando la *Confidentiality* e l'*Integrity* dei dati. Potrebbero essere modificati dati a cui non si dovrebbe accedere e allo stesso tempo potrebbero essere rivelati dati da utenti non autorizzati.

Lateral Movement**:** un utente potrebbe accedere inizialmente alla rete per poi spostarsi in profondità. Questo tipo di attacco non dovrebbe essere consentito ed è un accesso non autorizzato, il quale può trasformarsi in una violazione di *Confidentiality* e *Integrity* nel momento in cui l'attaccante ha accesso a maggiori privilegi.

Brute Force**:** possono verificarsi attacchi di forza bruta per trovare possibili combinazioni per accedere all'applicazione, nel momento in cui viene eseguito e si ha successo abbiamo un accesso non autorizzato dove potrebbe poi trasformarsi in una violazione della *Confidentiality* e *Integrity.*

Clumsy Manager/Clumsy Buyer Attack Tree

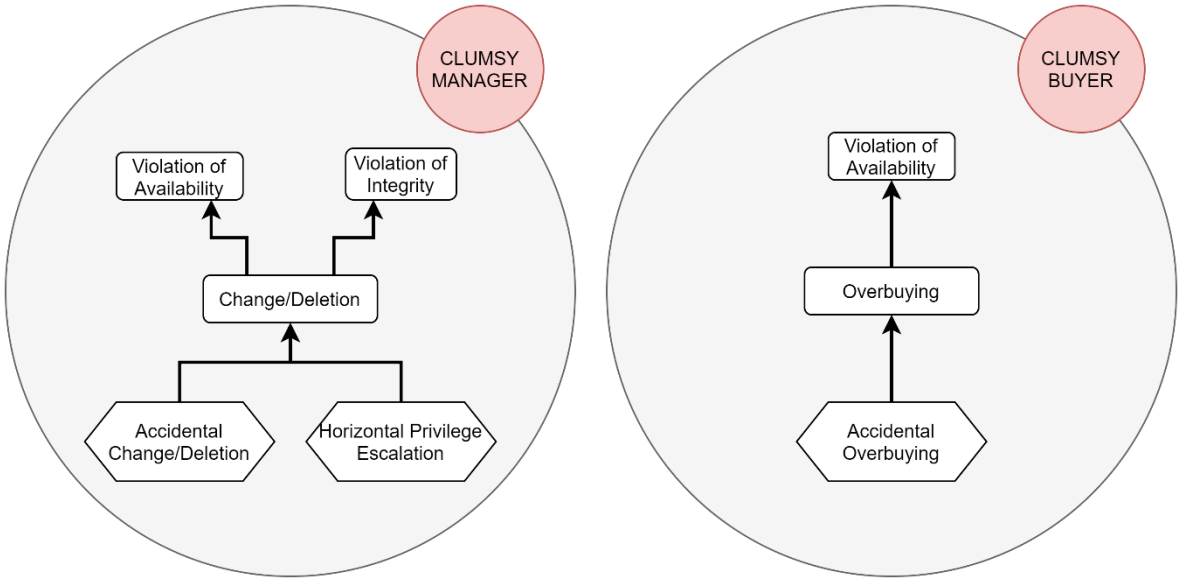

Fig. 10: Modelli degli Attack Tree dovuti ai Clumsy Users

Clumsy Manager

Accidental Change/Deletion**:** un manager potrebbe modificare o eliminare accidentalmente i dati e le informazioni dell'applicazione compromettendo quindi la l'*Availability* e *Integrity*.

Horizontal Privilege Escalation**:** un manager non autorizzato a modificare quell'evento tenta di cambiarne le proprietà, quindi vengono superati i privilegi poiché solo il manager che crea l'evento può modificare i dati dell'evento da lui creato. Se questo accade vengono violate l'*Availability* e *Integrity*.

Clumsy Buyer

Accidental Overbuying**:** un utente maldestro potrebbe comprare un numero di biglietti elevato compromettendo quindi l'*Availability* dei biglietti.

Abuse Case and Misuse Case Specification

In seguito, sono stati approfonditi gli Abuse e i Misuse Case, assegnando ad ogni threat un ID, un nome, gli attori coinvolti, la descrizione, il dato, gli stimoli e le condizioni che possono favorire l'attacco, il flow d'attacco e la risposta. I campi della Mitigazione e dei Requirements non funzionali verranno completati in una fase successiva.[2]

Abuse Case Specification

Abuse Case	
Use case ID:	A-EV-01
Use case name:	Event Privilege Escalation
Actors	Internal/Outsider Attacker, Manager
Description	An internal attacker gets higher credentials to manage the event
Data	Event
Stimulus and Pre.	An attacker gets holds of an unsanified input and conquers the manager's credentials to modify event informations
Attack 1 flow	Not sanitized input could bring to a shell injection with a consequent manager's privilege escalation
Response and Post.	System should restore access credentials and revoke violated certificates
Mitigations	
Non Functionals Requirements	

Fig. 11: Esempio di completamento dell'Abuse Case legato all'Event Privilege Escalation

Misuse Case Specification

Misuse Case Spefication	
Use case ID:	M-M-01
Use case name:	Event Horizontal Privilege Escalation
Actors	Clumsy Manager
Description	Manager can accidentally modify other's events specification
Data	Event
Stimulus and Preconditions	A manager wants to modify his event specifications
Attack 1 Flow	A manager can insert other's manager credentials
Response and Postconditions	Restore event specs changed
Mitigations	
Non functionals Requirements	

Fig. 12: Esempio di completamento del Misuse Case legato all'Event Horizontal Privilege Escalation

[2] Tutti gli Abuse Case e Misuse Case specificati sono nel file excel allegato nei rispettivi fogli "Abuse Case" e "Misuse Case".

Risk Assessment and Mitigation Table

La fase di analisi prosegue, sempre seguendo la rotta della metodologia "STRIDE" proposta da Microsoft[3], con la compilazione della Risk Assessment and Mitigation Table, che va ad aggiungere a quanto fatto anche una lista di attacchi possibili che ogni singolo asset identificato può subire. Ogni attacco è stato corredato anche di una serie di violazioni che porta con sé, ed una probabilità, intesa come un valore qualitativo in scala di Likert crescente a cinque valori.

ASSET	Spoofing	Tampering	Repudiation	Information Disclosure	Dos	Elevation of Privilege	Attack	Probability
Manager's Wallet				X			Intercept	3
	X	X		X			Unauthorized Access	4
	X	X		X		X	Vertical Privilege Escalation	2
		X			X		Denial of service	4
Event	X	X				X	Privilege Escalation	3
		X			X		Denial of Service	4
	X	X					Unauthorized Access	4
	X	X					Horizontal Privilege Escalation(C	2
		X	X				Accidental Change or Deletion(C	5
	X	X				X	Vertical Privilege Escalation(Clur	2
Ticket	X	X		X		X	Vertical Privilege Escalation	3
		X			X		Denial of Service	4
	X	X		X		X	Vertical Privilege Escalation(Clur	1
			X				Overbuying(Clumsy Buyer)	5
	X	X		X			Horizontal Privilege Escalation(Clumsy Manager)	3
		X	X				Accidental Change or Deletion(Clumsy Manager)	5
	X	X		X		X	Vertical Privilege Escalation(Clur	2
Buyer's Wallet				X			Interception	4
	X	X		X			Unauthorized Access	4
	X	X		X			Horizontal Privilege Escalation	4
		X			X		Denial of Service	5
System	X	X		X		X	Vertical Privilege Escalation	4
		X			X		Denial of Service	4

Fig. 13: Dettaglio della Risk Assesment and Mitigation Table in cui vengono aggiunte informazioni riguardo i possibili attacchi e la loro probabilità

Si può portare come esempio l'attacco di tipo *interception* che l'asset *Buyer's Wallet* può subire: un attacco di questo tipo apporta sicuramente una minaccia del tipo *information disclosure*, vale a dire una violazione della confidenzialità dei dati. Inoltre, si è deciso di associargli una probabilità pari a 4 su 5, in quanto i dati sono molto importanti e in quanto tali è probabile che si cerchi una loro violazione.

[3] https://www.microsoft.com/security/blog/2009/08/27/the-threats-to-our-products/

Mitigation: STRIDE Model

In seguito, l'analisi prosegue, prendendo in considerazione ora la fase di mitigazione.

Per ogni singola minaccia identificata, è stata individuata la proprietà violata e successivamente sono state riportate tutte le tecniche di controllo adottate.
Nella tabella seguente, ad esempio, si può notare come alla minaccia “Spoofing” sia stata associata una serie di tecniche di difesa contro la violazione dell’autenticazione, del genere di password più complesse (gestite in fase implementativa con le espressioni regolari), o una autenticazione a due fattori per l’utente System.
Infatti, l’utente System, avendo permessi superiori rispetto agli altri, è bene che sia protetto con un ulteriore grado di sicurezza, in quanto ha la funzione di cambiare i ruoli degli utenti, vale a dire che può fare in modo che un utente sia una biglietteria, un manager e viceversa sia per biglietteria che per manager.

Threat	Property	Control
Spoofing	Authentication	1. Authentication protocols with complex passwords 2. Strong authentication for System User 3. Wallet authentication 4. Password hashing on DB 5. Blockchain address hashing on DB

Threat	Property	Control
Tampering	Integrity	2. Data validation checks on Blockchain structures and functions 3. Web3 controls on transactions integrity 4. MySQLjs controls on query integrity (SQL injection) 5. Hashes and Encryption 6. Tamper-resistant protocols

Threat	Property	Control
Repudiation	Nonrepudiation	1. Authentication protocols with complex passwords 2. Strong Authentication for System User 3. Secure Logging and Auditing 4. Blockchain transactions signatures 5. Secure time stamps 6. Trusted third parties 7. Nonrepudiaton possible executions

Threat	Property	Control
Information Disclosure	Confidentiality	1. Encryption 2. ACLs 3. Privacy-enhanced protocols 4. Protected secrets 5. Not stored secrets

Threat	Property	Control
DoS	Availability	1. ACLs 2. Throttle quotas and Blockchain timing 3. Authentication Protocols

Threat	Property	Control
Elevation of Privileges	Authorization	1. Data execution prevention 2. Linux kernel's address space layout randomization 3. Running applications with least previlege 4. Use of compilers and try-catch structures that trap buffer overruns 5. Encryption of software components

Fig. 14: Tabelle dello STRIDE model contestualizzato al software in progettazione

Continuando, è stata poi arricchita la Risk Assessment and Mitigation Table[4], portandola nella sua forma ultima, aggiungendo colonne relative a tutta la fase di mitigazione:

- "Control"
- "Cost"
- "Feasibility"
- "Priority"
- "Effort"
- "Impact"
- "Risk"

ASSET	Spoo	Tam	Rep	Info	Dos	Elev	Attack	Proba	Control	Cost	Feasibility			Impact	Risk
Manager's Wallet				X			Intercept	3	1. Use of ACL for users control	3	1. Technically feasible	9	9	4	12
	X	X		X			Unauthorized Access	4	1. Authentication 2. ACL	4 3	1. Technically feasible but needs user approval 2. Technically feasible	14	28	4	16
	X	X		X		X	Vertical Privilege Escalation	2	1. Strong Authentication 2. ACL 3. Data Execution Prevention	4 3 2	1. Technically feasible but needs user approval 2. Technically feasible 3. Technically feasible, already implemented with use of blockchains	6	18	4	8
		X			X		Denial of service	4	1. ACL 2. Throttle Quotas	3 3	1. Technically feasible 2. Technically feasible, requires a waiting list	12	24	3	12
Event	X	X				X	Privilege Escalation	3	1. Authentication 2. ACL 3. Data Execution Prevention	4 3 2	1. Technically feasible but needs user approval 2. Technically feasible 3. Technically feasible, already implemented with use of blockchains	9	27	3	9
		X			X		Denial of Service	4	1. ACL 2. Throttle Quotas	3 3	1. Technically feasible 2. Technically feasible, requires a waiting list	12	24	3	12
	X	X					Unauthorized Access	4	1. Authentication 2. ACL	4 3	1. Technically feasible but needs user approval 2. Technically feasible	14	28	2	8
	X	X					Horizontal Privilege Escalation(C	2	1. Authentication 2. ACL	4 3	1. Technically feasible but needs user approval 2. Technically feasible	7	14	3	6
		X	X				Accidental Change or Deletion(C	5	1. ACL 2. Secure logging and auditing	3 2	1. Technically feasible 2. Technically feasible, requires control mechanisms and staff commitment	12.5	25	3	15
	X	X				X	Vertical Privilege Escalation(Clus	2	1. Strong Authentication 2. ACL 3. Data Execution Prevention	4 3 2	1. Technically feasible but needs user approval 2. Technically feasible 3. Technically feasible, already implemented with use of blockchains	6	18	4	8

Fig. 15: Dettaglio della forma finale della Risk Assessment and Mitigation Table in cui sono aggiunti gli indicatori di rischio

Le prime tre colonne presentano le tecniche di difesa per ogni attacco, il relativo costo ed alcuni commenti sulla fattibilità per ciascuna di esse.
La colonna *Priority* valuta la media dei prodotti tra probabilità e costo, ed *Effort* la somma dei prodotti, secondo le seguenti formule:

$$Priority = \frac{1}{N}\sum_{i=1}^{N} Probability * Cost(i) \qquad Effort = \sum_{i=1}^{N} Probability * Cost(i)$$

Priority ed Effort sono due misure da noi assegnate per definire la priorità prevista per la messa in sicurezza di una eventuale minaccia e lo sforzo, inteso come difficoltà di intervento, per effettuarne la mitigazione.

Impact invece riguarda una valutazione qualitativa dell'impatto di una violazione ad ogni asset identificato, con una scala di Likert a cinque valori, ed il prodotto tra i valori di questa colonna e le probabilità fornisce i corrispondenti nell'ultima colonna, ovvero *Risk*, colorata seguendo i criteri imposti dalla matrice dei rischi presentata di seguito:

[4] Tabella completa nel file excel allegato nel foglio con titolo "RiskAssessmentandMitigationTable".

	Negligible (1)	Minor (2)	Moderate (3)	Significant (4)	Severe (5)
Very Likely (5)	5	10	15	20	25
Likely (4)	4	8	12	16	20
Possible (3)	3	6	9	12	15
Unlikely (2)	2	4	6	8	10
Very Unlikely (1)	1	2	3	4	5

1-->3
4-->6
7-->12
13-->19
20-->25

Fig. 16: Risk Matrix e sua legenda per la codifica dell'indicatore di rischio

Questa matrice presenta nelle righe le probabilità, e nelle colonne l'impatto, entrambi secondo scale di Likert a cinque valori. Dunque, nelle celle troviamo il prodotto tra i relativi valori delle righe e colonne, ed ogni cella viene colorata secondo la legenda giustapposta.

Ogni scelta del colore permette una rapida identificazione della gravità di ogni situazione, infatti il colore verde si utilizza nel caso in cui il prodotto tra i valori sia relativamente basso (un range tra 1 e 3), il che significa che sia la probabilità che l'impatto sono bassi, permettendo così di porre una minore importanza ai rischi associati, a favore chiaramente di quelle celle colorate in rosso, il cui prodotto ha i valori maggiori (range da 20 a 25), che quindi consiglia fortemente di porre loro la massima rilevanza in fase di mitigazione.

Mitigation: Abuse/Misuse Case

La fase di progettazione della mitigazione si conclude con il completamento degli Abuse Case Specification e Misuse Case Specification, con l'inserimento delle due ultime righe rimaste vuote finora con: le metodologie di controllo proposte ed i requisiti non funzionali, ovvero caratteristiche non richieste esplicitamente dal cliente ma che si avrà particolare premura nel garantire ed implementare, essendo state ritenute caratteristiche imprescindibili.
Di seguito riportiamo l'esempio dell'abuse case *Event Privilege Escalation*, ovvero una violazione delle restrizioni associate agli utenti nelle interazioni con gli eventi, come un attaccante interno che si appropria di permessi di utenti di maggior grado, permettendogli così di mettere in azione delle modifiche ad eventi che altrimenti non potrebbe cambiare, come mostrato nella riga *Description*.

Un abuse di questo tipo viene gestito tramite una lista di controllo degli accessi (ACL), tenendo così sotto controllo tutti gli accessi al server con una lista di log che presenta tutte le informazioni chiave sugli utenti, e quindi come requisito non funzionale è stato richiesto che il server su cui ci si appoggia mantenga la ACL, mettendo in pratica quanto proposto.

Abuse Case	
Use case ID:	A-EV-01
Use case name:	Event Privilege Escalation
Actors	Internal/Outsider Attacker, Manager
Description	An internal attacker gets higher credentials to manage the event
Data	Event
Stimulus and Pre.	An attacker gets holds of an unsanified input and conquers the manager's credentials to modify event informations
Attack 1 flow	Not sanitized input could bring to a shell injection with a consequent manager's privilege escalation
Response and Post.	System should restore access credentials and revoke violated certificates
Mitigations	Monitoring system access log with ACL
Non Functionals Requirements	The server should mantain the ACL (access control list)

Fig. 17: Mitigation Table dell'Abuse Case di esempio relativo all'Event Privilege Escalation

A questo punto tutte le attività di progettazione volgono al termine, permettendo così l'avvio della successiva fase di implementazione.

Implementazione

Sistemi Utilizzati: OS e VM

A partire dalla progettazione effettuata nel capitolo precedente, per fare in modo di ottenere una base di partenza che potesse permettere la realizzazione efficace di tutti i requisiti descritti si è optato per una architettura Client-Server nella forma di un sito web consultabile attraverso browser.

Tale applicazione verrà quindi realizzata dal gruppo di progetto con l'obiettivo di essere inserita all'interno di una macchina con OS GNU/Linux che possa fungere da server fittizio su cui depositare il codice eseguibile per l'adempimento dei requisiti.
Si è scelto quindi di optare per una macchina virtuale compatibile con la piattaforma Oracle VM Virtualbox[5]. Tale macchina virtuale, oltre che per estrema comodità di sviluppo, rappresenta un'immagine a costo nullo di una eventuale macchina reale su cui poi depositare l'eseguibile della nostra applicazione.
Come sistema operativo scelto, viste la comodità di sviluppo e la sufficiente portabilità sulle macchine personali dei membri del gruppo di progetto, si è scelto di utilizzare la distribuzione di GNU/Linux Lubuntu 21.04[6], derivata della più famosa sorella maggiore Ubuntu 21.04 con la quale condivide gran parte degli applicativi se non il Desktop Environment meno esoso di risorse basato su Lx/Qt.
La scelta di un OS di questa famiglia rispetta alcuni dei requisiti di sicurezza minimi richiesti relativi all'esecuzione di codice sorgente sotto adeguato livello di permessi. La politica infatti rispecchia il principio dei least privileges usati, un esempio, tra i tanti implementati, di principio seguito e adottato nella risoluzione del progetto, descritto nelle tante librerie open-source relativa alla sicurezza informatica, come l'OWASP. Tale principio garantisce una minima protezione nel caso in cui diversi tipi di attaccanti riescano ad avere accesso alla macchina su cui risiede il codice. Sulla base di queste librerie, si sono poi adottate altre regole e best practices che forniscono nell'insieme un elevato livello di sicurezza all'applicazione.

Infrastruttura Utilizzata: Client-Server e pacchetti NodeJS

Per realizzare l'architettura Client-Server sopracitata, si è scelto di ricorrere al framework ExpressJS[7] eseguibile con il runtime basato su JavaScript NodeJS[8]. ExpressJS mette a disposizione diversi strumenti per la realizzazione di applicazioni Web e consente di collegare a NodeJS uno strato di middleware a cui poi congiungere diversi driver per l'utilizzo di altre strutture software.
A questa struttura di base vengono annessi alcuni moduli compatibili con NodeJS. Tra i più importanti:

- Mysqljs[9]: per ottenere un driver di collegamento con il database MySQL e la gestione delle query.
- Web3JS[10]: API di collegamento che contiene diverse librerie per interfacciarsi ad un Ethereum Node attraverso porte HTTP, IPC o WebSocket.

---

[5] Disponibile presso https://www.virtualbox.org/. La VM allegata al progetto richiede, oltre ad una installazione standard di Oracle VM Virtualbox, l'installazione aggiuntiva del pacchetto Oracle VM Extension Pack. Il codice è stato progettato, creato e descritto per e con la versione 6.1.28 di tale applicazione.
[6] Disponibile presso https://www.ubuntu-it.org/derivate/lubuntu
[7] Disponibile presso https://expressjs.com/it/
[8] Disponibile presso https://nodejs.org/
[9] Disponibile presso https://github.com/mysqljs/mysql
[10] Disponibile presso https://web3js.readthedocs.io/

Oltre ai due moduli riportati, sono stati utilizzati diversi altri pacchetti installati attraverso il package manager *npm*[11]. Funzione interessante di questo package manager è quella fornita dall'*npm Registry*, un registro Open Source che tiene traccia delle vulnerabilità software dei pacchetti disponibili presso i repository collegati. Attraverso la funzione di audit della suite è quindi possibile, nel caso siano disponibili, applicare patch di sicurezza al codice installato per risolvere queste vulnerabilità. Va citato che nel corso dello sviluppo dell'applicazione di progetto si è ricorso di frequente a questa funzione e che alla data di consegna del codice di progetto[12] non erano presenti vulnerabilità tra i pacchetti utilizzati.

I pacchetti per NodeJS minori utilizzati come complemento all'applicativo basato su Express sono:

- bcryptjs[13]: Libreria molto diffusa per l'hashing delle password.
- cookie-parser[14]: Libreria per il parsing dei cookie e il loro utilizzo.
- crypto-js[15]: Libreria che implementa in JavaScript diversi algoritmi crittografici, utilizzati, nel nostro caso, per crittografare in chiave simmetrica.
- dotenv[16]: Modulo che carica diverse variabili di ambiente da un file .env in un processo *process.env* che le serve su richiesta.
- express-session[17]: Modulo che consente la gestione dei dati di sessione.
- hbs[18]: Libreria e motore grafico per la scrittura di template semantici di viste di pagine basati su codice html.
- nodemon[19]: Script di supporto allo sviluppo per ricaricare il codice eseguibile del server successivamente ad una modifica del software. Questo strumento non viene utilizzato nello stato definitivo di progetto.
- winston[20]: Una libreria universale adibita al logging di diverse informazioni e lo sviluppo di liste di accesso.

Il pacchetto hbs mette a disposizione un motore per l'interpretazione di linguaggio Handlebars. Definendo hbs come motore predefinito per Express con cui effettuare il rendering delle viste delle pagine Web si fa sì che l'engine si occupi di fornire al client una pagina statica codificata in base a diversi parametri passati dal controller. Questi vengono codificati con degli helpers che il linguaggio Handlebars mette a disposizione al fine di ottenere il rendering o meno di blocchi specifici del codice. Così facendo, se ad esempio si inserisce un blocco html all'interno di un blocco contraddistinto da un helper hbs di tipo:

{{#if condizione}} …<blocco html>… {{/if}}

se condizione è falsa il codice interno ai "due baffi" non viene riportato nella vista statica, altrimenti se condizione è vera ne viene effettuato il processo di render e si ottiene la pagina richiesta.

Come descritto nei capitoli precedenti, il programma risultante dovrà essere in grado di tenere traccia di alcune informazioni relative agli utenti che lo utilizzano. Per questo, si è deciso di

[11] Disponibile presso https://www.npmjs.com/
[12] 22 novembre 2021.
[13] Disponibile presso https://github.com/dcodeIO/bcrypt.js#readme
[14] Disponibile presso https://github.com/expressjs/cookie-parser#readme
[15] Disponibile presso https://github.com/brix/crypto-js#readme
[16] Disponibile presso https://github.com/motdotla/dotenv#readme
[17] Disponibile presso https://github.com/expressjs/session#readme
[18] Disponibile presso https://handlebarsjs.com/
[19] Disponibile presso https://nodemon.io/
[20] Disponibile presso https://github.com/winstonjs/winston#readme

appoggiarsi anche all'utilizzo di un semplice database MySQL su cui salvare le informazioni base dell'utente (nome, cognome, e-mail di iscrizione, password), il ruolo che l'utente ha all'interno della nostra piattaforma, l'indirizzo del suo wallet in carico alla Blockchain e informazioni relative agli accessi.
Per ottenere le funzionalità appena descritte ci si è affidati all'applicativo XAMPP[21] nella sua versione per GNU/Linux. Grazie a questo software, famoso ambiente di sviluppo PHP, si ha a disposizione un'interfaccia MySQL grafica impostata e funzionante. La scelta di questo applicativo, per quanto non elegante, visto il fatto che solitamente viene utilizzato come supporto ad ambienti diversi dal nostro, deriva da una primissima indagine preliminare sulle possibili implementazioni del codice di progetto in cui si era momentaneamente deciso di effettuare prove con codifica PHP+HTML. Si è scelto quindi di mantenere questa soluzione, nonostante molti dei supporti forniti di XAMPP possano essere assimilati a codice bloatware per la nostra soluzione. Soluzioni più eleganti potevano essere realizzate con il semplice ausilio di applicativi quali MariaDB[22], ma si è comunque scelto di risparmiarsi parte del lavoro già svolto nelle prime fasi piuttosto che optare per un trasferimento dei dati relazionali.

Software e Strumenti Utilizzati: Blockchain e Smart-Contract

L'architettura client-server appena descritta non soddisfa pienamente i requisiti di progetto; per questo si è ricorsi all'utilizzo di una blockchain e di alcune interfacce di collegamento.

Si è scelto di optare per l'utilizzo di uno script di generazione accelerata della rete di nodi virtuali sulla quale si basa l'architettura delle blockchain. Lo script utilizzato per il download automatico delle configurazioni da adottare è disponibile nel package manager npm sotto il nome di GoQuorum-Wizard[23]. Tale script di configurazione consente il setup di una rete GoQuorum in un tempo minimo ed è mantenuto dalla compagnia ConsenSys. La rete GoQuorum generata è una rete basata sull'utilizzo dello strato di protocolli Quorum, il quale è sovrapposto al core-code delle reti Ethereum e ne modifica alcune caratteristiche. Rispetto ad una più classica implementazione del codice su di una blockchain basata sulla piattaforma Ethereum, una blockchain del tipo installato consente, tra le varie diverse caratteristiche funzionali, l'implementazione di algoritmi di consenso basati sulla Proof-of-Authority in sostituzione alla più onerose Proof-of-Work o Proof-of-Stake. Questo rende molto più fruibile lo sviluppo di applicativi del genere e ne azzera i costi.

Dal punto di vista concettuale, inoltre, la Proof-of-Authority consentita dal fatto che le reti GoQuorum sono di tipo *permissioned,* si addice positivamente al caso di studio nel quale si assume che la piattaforma di vendita di biglietti preferisca gestire autonomamente la propria blockchain in modo da poter godere dei vantaggi dell'utilizzo di tale struttura software senza dover curarsi delle spese addizionali e delle fee richieste dall'utilizzo di una rete pubblica. Così facendo potrebbe quindi mantenere il totale controllo sul codice in esecuzione.

Per ottenere la fruibilità del codice in carico alla blockchain appena descritta si è scelto l'utilizzo dell'API Web3 e delle librerie che il pacchetto mette a disposizione. Il nostro codice, quindi, sfrutterà le potenzialità di quest'ultimo per generare delle istanze degli smart-contract caricati sulla rete e interfacciarsi con essi.

Vista l'assenza della necessità progettuale di sfruttare la possibilità di utilizzare transazioni private fornita dal codice Quorum, non sono stati installati ed utilizzati pacchetti aggiuntivi capaci di fornire

[21] Disponibile presso https://www.apachefriends.org/
[22] Disponibile presso https://mariadb.org/
[23] Disponibile presso https://github.com/ConsenSys/quorum-wizard#readme

questa funzionalità sovrapponendosi a Web3. Tale tipologia di transazioni esula dalla semplicità del nostro caso di studio ed è quindi stata ignorata.

Gli smart-contract alla base dell'implementazione e dell'uso della blockchain del progetto sono stati scritti in linguaggio Solidity 0.8.8 e successivi. Sono stati poi compilati e caricati all'interno della rete suddetta con l'uso e la disponibilità del collegamento dell'IDE utilizzato, RemixIDE, con la porta RPC del nodo principale. Fornendo in contratti in modalità as-is, assumendo quindi che questi non vengano modificati, non si è ritenuto necessario effettuare compilazione e deploy a livello del codice del nostro server e non sono quindi state utilizzate librerie aggiuntive per questo scopo.

Implementazione: Smart-Contract e Blockchain

Il processo di implementazione e sviluppo dell'applicazione si è svolto seguendo un principio di costruzione bottom-up: si è scelto di partire dagli elementi di base del backend, quindi codice per la blockchain e codice per la gestione del server, con la minima aggiunta delle librerie grafiche che ne consentissero il testing quando richieste, fino a concludere con la finalizzazione del frontend e delle interfacce grafiche. Per questo, nella trattazione, vengono prima riportate le informazioni relative ai primi processi di sviluppo.

Come già introdotto, la nostra applicazione si avvale dell'utilizzo di una blockchain che funge da spina dorsale della maggior parte delle funzionalità del software. Tali funzionalità, codificate nella gestione di eventi e biglietti relativi, richiedono l'implementazione di alcuni smart-contract che possano fungere da supporto informatico e da strumento per la definizione dei possibili casi d'uso dei vari utenti.

Si è scelto quindi di realizzare la stesura di due smart-contract: quello che spesso verrà definito come *contratto Evento*, codificato in /contracts/Evento.sol e il *contratto Ticket* in /contracts/Ticket.sol.

Il contratto Evento definisce e supporta una struttura che funge da base di dati per gli eventi che la piattaforma deve rendere disponibile. La struttura consente il salvataggio di informazioni quali il nome dell'evento, la data in cui avverrà, informazioni sul proprietario e sulla disponibilità di biglietti. Attorno a questa struttura vengono realizzate diverse funzioni per la gestione delle informazioni, inserimento o cancellazione degli eventi. La maggior parte di queste utilizzerà lo strumento *require()* fornito dal linguaggio Solidity per la definizioni di condizioni in assenza delle quali la transazione che si vuole eseguire non può andare a termine. Ad esempio, l'eventuale modifica di un evento può avvenire se e solo se l'evento che si vuole modificare è già stato inserito nella piattaforma: grazie all'utilizzo dello strumento require(), con diverse e varie condizioni, è quindi possibile far terminare l'esecuzione della transazione prima che si generi un errore di integrità dei dati o che si vìoli una policy di sicurezza. Questo potente strumento è stato utilizzato anche nel caso in cui si voglia garantire l'integrità dei dati e, ad esempio, si richiede che soltanto la figura del Manager effettivo proprietario dell'evento possa modificarne le caratteristiche.

Tra le funzioni così definite, ve ne sono diverse che corrispondono a ciò che in fase di progettazione sono stati definiti come eventi rilevanti, con il sinonimo di avvenimenti a livello di codice, che consentono la realizzazione di un caso d'uso o di una parte di esso. Queste funzioni sfruttano un altro costrutto del linguaggio Solidity che è la funzione *emit* per l'emissione di un *event()* definito precedentemente. Gli event() verranno quindi definiti in corrispondenza delle attività di interazione con la struttura dati degli eventi (concerti, spettacoli…) e genereranno dei log a livello della blockchain che attraverso codice apposito saranno tradotti in log dell'applicazione. Così facendo inizia a definirsi una possibile lista di controllo delle modifiche che vengono fatte agli asset integrati nella blockchain.

Il contratto Evento, inoltre, oltre ad una pura interfaccia per la gestione degli spettacoli, funge anche da interfaccia iniziale per l'acquisto, il sigillo e la validazione dei biglietti. Questo perché nell'interazione con i ticket è necessario effettuare precedentemente controlli sugli stati dell'evento ad esso relativo.

Il contratto Ticket, allo stesso modo del precedente, utilizza le strutture messe a disposizione dal linguaggio Solidity per diversi controlli di sicurezza e integrità dei dati e delle interazioni forniti dall'utente. Anch'esso conterrà quindi le informazioni necessarie alla definizione di una struttura dati che rappresenti il biglietto da acquistare e contenga quindi informazioni relative all'evento di cui fa parte, al suo sigillo fiscale ed alla sua validazione.

Si genera quindi una struttura indentata nella quale, eccetto alcuni casi di interazione diretta con il suddetto, il contratto Evento funge da interfaccia per il contratto Ticket. Affinché l'uno possa quindi caricare ed eseguire le funzioni dell'altro sulle strutture dati dell'altro è necessario che entrambi conoscano l'*address* che la blockchain gli assegna in fase di deployment del contratto. Questa lieve problematica viene risolta attraverso l'adeguato utilizzo delle funzioni di costruzione dei due contratti in modo che sia quindi possibile settare le variabili relative agli address dei contratti corrispettivi.

Come già anticipato, i contratti definiti vengono caricati all'interno della blockchain e resi disponibili dal wallet predefinito fornito dall'inizializzazione della blockchain. Questo wallet verrà definito come *wallet di sistema* e verrà assicurato all'utente admin. Tutte le altre interazioni verranno svolte dagli utenti con l'utilizzo di un loro wallet personale e questo farà in modo di rispettare i requisiti di isolamento e sicurezza dei portafogli degli utenti. Portafogli nei quali gli smart contract consentiranno il salvataggio e deposito dei ticket acquistati e degli eventi inseriti.

Nel caso in cui vi sia una irregolarità nell'esecuzione del codice appena definito che possa condurre su tracce di esecuzione definite errate in fase progettazione, perché possibili di una violazione delle policy di sicurezza, ci si aspetta che la Ethereum Virtual Machine incappi nelle condizioni di errore definite dai require(), o dal linguaggio di programmazione, ed effettui un revert della transazione, bloccandola.
Vedremo più avanti nella trattazione come il codice inserito all'interno di questi strumenti sarà utilizzato per avvisare l'utente di un errato comportamento o di un fault del codice.

Implementazione: Server e applicativo Web

L'applicativo web ha sua definizione di partenza nello script /index.js: qui vi risiede il codice relativo all'importazione dei moduli middleware definiti in uno dei paragrafi precedenti, nonché l'inizializzazione di variabili di ambiente per l'applicativo, l'inizializzazione dei moduli e l'avvio del server con le indicazioni sui percorsi di routing di partenza.

In questo file viene inizializzato anche il modulo di gestione delle sessions: sessioni di visualizzazione dell'interfaccia web da parte di un utente le cui informazioni vengono archiviate server-side. L'accesso al canale di comunicazione per la gestione delle sessioni avviene attraverso l'utilizzo di una chiave criptata archiviata nei cookies di navigazione. Sono gli stessi moduli *ExpressJS* e *express-session* ad occuparsi di encrypting e offuscamento della trasmissione in oggetto; al programmatore viene richiesto l'utilizzo di un segreto che, in accordo con il principio del "Not-saved secrets" il gruppo ha deciso di generare randomicamente per ogni avvio del server. Si fa l'assunto, infatti, che in un ambiente di tipo Production, il sistema venga mentenuto e opportunamente riavviato magari in condizioni di basso carico o all'evenienza.

Definito l'applicativo di partenza e la porta di ascolto, il gruppo ha deciso di avviare il server Express sulla porta 8888. Limitati dagli strumenti didattici utilizzati e dall'obiettivo dimostrativo di questo progetto si è scelto di non prevedere l'utilizzo di protocolli di tipo SSL o TLS, ma in caso di effettivo deploy dell'applicazione in un ambiente Production, si ipotizza che sia possibile implementare una trasmissione di tipo HTTPS e non HTTP standard grazie all'utilizzo di soluzioni che prevedano i protocolli sopracitati. In questo modo, per quanto il resto del codice sia comunque posto in condizioni di sicurezza per violazioni di questo tipo, si rendono più difficili le decodifiche delle trasmissioni che possono essere intercettate da un eventuale attaccante.

Per il resto della codifica il gruppo ha scelto di utilizzare diverse direttive di buona programmazione del paradigma MVC, in particolare, in assenza di un vero e proprio modello di OOP da codificare, si sono mantenute soprattutto le strutture relative al campo View e al campo Controller del paradigma.

Definite quindi delle viste *dummy* di test, è stato possibile realizzare i controller dell'applicativo. In particolare, sono stati definiti diversi script per effettuare il controllo di diverse funzioni:

- Lo script auth.js viene utilizzato per il controllo del codice che consente l'autenticazione al sito web da parte delle diverse tipologie di utente.
- Lo script blockchain.js consente il controllo e l'interfaccia con la blockchain GoQuorum definita precedentemente.
- Lo script cripter.js mette a disposizione diverse funzioni per effettuare l'encrypting o l'hashing di stringhe.
- logger.js è lo script utilizzato per la gestione e generazione dei log dell'applicazione su diversi livelli informativi.
- Infine, session.js, uno degli script e dei file più corposi del progetto, è quello che ci consente la gestione della risposta in viste codificate da offrire all'utente in base ai permessi che possiede ed alle richieste che quest'ultimo fa.

Auth.js

Lo script definito in /controllers/auth.js mette a disposizione le funzioni relative alla registrazione, all'autenticazione e alla modifica degli utenti sul portale di progetto.

L'inizializzazione del plugin mysqljs avviene fornendo indirizzo e dati di accesso al Database che sono opportunamente archiviati nel file *.env*, in modo che il modulo dotenv si occupi di renderli disponibili solo con l'interfaccia al processo relativo, garantendo un maggiore strato di sicurezza riguardo i segreti conservati sulla macchina di progetto.

La funzione di registrazione effettua il parsing delle informazioni inserite in una form nella quale, come meccanismo di sicurezza, viene inserita la password di registrazione due volte. Quindi, verificato che il pattern della password sia corretto e cioè che la password superi gli 8 caratteri e ne contenga almeno uno maiuscolo, uno numerico ed uno simbolico, procede ad una query sul database per l'opportuno salvataggio dei dati. Allo stesso modo, la funzione di login, effettuato il parsing della richiesta proveniente dalla form apposita, effettua una query annidata sul database per ottenere i dati rilevanti relative alle autorizzazioni di navigazione e salvarli sulla sessione dell'utente. Per tutte le evenienze per le quali sia necessaria una query al database, quindi per tutti gli Aspect relativi alle query con il database, ci si è avvalsi di una protezione per l'SQL-Injection: la query è stata scritta utilizzando un preciso formato che fa sì che il modulo mysqljs si occupi del parsing della stessa, riportando un errore nel caso in cui venga effettuato un tentativo di manomissione.

```
if (email && password) {
  db.query(
    "SELECT * FROM users WHERE email = ?",
    [email],
    async function (error, results, fields) {
      if (error) {
        log.error(error);
      } else if (results.length > 0) {
        if (results[0].locked && results[0].locked_at > time - 600 * 1000) {
          //Locked vale ancora per 10min in millisecondi
          req.session.destroy();
          res.render("index", {
            error: "Utente temporaneamente bloccato.",
          });
        } else {
          let comparison = await bcrypt.compare(
            password,
            results[0].password
          );
          if (comparison) {
            var events = await blockchainController.getEventi();

            function compare(a, b) {
              if (a.ndispbiglietti > b.ndispbiglietti) {
                return -1;
```

Fig. 18: Esempio di codice relativo alla funzione di login in cui nel secondo blocco è ben visibile la query SQL e il formato di scrittura utilizzato

Per proteggersi da attacchi di tipo Brute-Force per la disclosure delle informazioni di accesso degli utenti all'interno dello script è previsto un controllo di bloccaggio dell'account che si verifica nel caso in cui si provi ad inserire una password errata per un'e-mail di accesso per più di cinque volte. A questo punto, giunti al bloccaggio dell'account per 10 min, anche se venisse inserita la password corretta si rimarrebbe in blocco; l'unica soluzione effettiva per garantire lo sblocco dell'account è quindi l'attesa temporale.

Lo script inoltre mette a disposizione le funzioni che consentono all'utente di sistema di effettuare l'accesso alla piattaforma per prevedere la modifica delle autorizzazioni degli altri utenti. In modo da rendere più sicuro l'accesso, l'utente di sistema deve inserire in aggiunta all'email di accesso e la sua password, anche un codice a sei cifre aggiuntivo. Questa multi-factor authentication, oltre che rappresentare una ulteriore protezione di sicurezza, ci permette di effettuare una dimostrazione tecnica delle possibilità della soluzione scelta.

In sede di progettazione dell'implementazione, il gruppo ha studiato l'evenienza di utilizzare servizi esterni di autenticazione. Questo tipo di servizi, che utilizzano protocolli di protezione all'avanguardia, avrebbero reso possibile la *delocalizzazione* del database di gestione degli utenti e avrebbero consentito una maggiore usabilità dell'applicativo, consentendo magari l'accesso degli utenti via il proprio account Facebook, Google o Apple. Il gruppo di progetto ne consiglia l'implementazione in ambiente Production, ma ancora una volta, limitati dalla semplicità della soluzione, si è preferita una codifica più semplice e meno costosa.

Blockchain.js

Lo script si occupa di gestire una istanza degli smart-contract generata a partire dal codice ABI degli stessi e, con questa, interfacciarsi alla blockchain. Le istanze dei contratti vengono quindi associate al loro address sulla blockchain: address che vengono opportunamente archiviati nel file /configs/.env in modo da permetterne un salvataggio più sicuro.

Come già accennato precedentemente, il codice dell'applicativo web si interfaccia alla rete GoQuorum con l'utilizzo della porta WebSocket[24] del nodo principale. Rispetto alla standard RPC, la WS consente il supporto alle *subscription* e un canale di sicurezza più sicuro, essendo WS un protocollo più recente e mantenuto rispetto a RPC. Anche l'indirizzo della porta WebSocket del nodo principale è archiviato nel file *.env*.

Per ogni funzione di interesse definita negli smart-contract, o se necessario per più di una di queste, è stata definita una funzione dello script che si occupa preparare l'esecuzione della transazione o di effettuarne il collect del risultato. In particolare, oltre ad effettuare il parsing della comunicazione che proviene dai form html, queste funzioni si occupano di ottenere l'address dell'utente dalla sua sessione, di sbloccare l'account dell'utente per far sì che questo consenta le transazioni e infine di lanciare la transazione codificata nello smart-contract di riferimento.

Similarmente a come veniva fatto nello script di autenticazione per la gestione dell'Aspect di collegamento con il database, ogni funzione nello script che consente di settare dei parametri nella struttura della blockchain è contenuta in una *try-catch* che oltre a consentire un corretto handling delle transazioni, fa sì che in caso di errore la piattaforma risponda restituendo la pagina home. Corredato alla pagina home, in risposta si ha anche un messaggio di errore che riporta una stringa di codice che lo definisce. Questa stringa è esattamente la stringa inserita nel codice di require() degli smart-contract. Per ottenere questa stringa si sfruttano i revert della EVM definiti precedentemente e il preventivo settaggio a *true* della variabile di ambiente di Web3 *handleRevert*.

Ogni funzione di scrittura sulla blockchain che viene lanciata dallo script contiene alcune istruzioni che vengono lanciate in maniera asincrona in quanto è necessario attendere che il nodo principale scriva il blocco contenente la transazione sulla blockchain e che questo blocco venga approvato. Lo stesso ovviamente non avviene per le funzioni di lettura delle variabili, in quanto non si ha la necessità di scrivere blocchi. Si segnala quindi che il tempo massimo di attesa che il blocco con una o più transazioni richieste, visto l'utilizzo del protocollo IBFT, è esattamente il tempo di generazione dei blocchi definito dal protocollo, non modificato dal gruppo di progetto, empiricamente misurato come circa due secondi.

Visto il modesto tempo di attesa di generazione del blocco, quando possibile, il gruppo ha scelto di preferire che più transazioni contigue venissero eseguite parallelamente e quindi approvate all'interno di un solo blocco; ne è un esempio l'implementazione della funzione per l'acquisto di biglietti che supporta la generazione di un blocco contenente più transazioni eseguendone il codice di alcune sincronamente (e quindi parallelamente).

Lo script mette a disposizione anche del codice per la gestione delle subscription agli eventi *emitted* dagli smart-contract in modo da renderne possibile il logging. Vengono quindi definiti diversi listeners, uno per ogni event nei contratti, che al lancio del codice di *emit* relativo dei contratti, ne effettuano il parsing e ne salvano l'accadimento all'interno dei log On-Chain. In questo modo e grazie ad una opportuna definizione degli eventi nei contratti è possibile ottenere delle liste di accesso ed utilizzo degli smart-contract e quindi di entrare in possesso di uno dei meccanismi di

[24] Definito e standardizzato dall'IETF: https://datatracker.ietf.org/doc/html/rfc6455

sicurezza più importanti definiti in fase di progettazione. Attraverso l'analisi dei log generati è dunque possibile individuare molti tipi di violazioni di sicurezza tra quelle previste negli Attack-Trees.

Cripter.js e logger.js

I due script fungono da semplice interfaccia di gestione dei moduli per la cifratura di CryptoJS e di winstonJS per il logging.

In particolare, le funzioni di cifratura e decifratura vengono utilizzate come ridondanza di controllo di sicurezza nella gestione degli address dei wallet degli utenti. Il codice di questi indirizzi è necessario per effettuare le transazioni sulla blockchain e, per mantenere una discreta separazione dei moduli di progetto, si è scelto di mantenere tale informazione nella sessione dell'utente, piuttosto che effettuare diverse query al DB all'interno dello script blockchain.js.
Per effettuare l'encrypting di queste informazioni è stato usato l'algoritmo Rabbit per il quale viene utilizzato un segreto archiviato nel file nascosto /configs/.env schermato dal modulo *dotenv*.

Il logging, sia Off che On-Chain, viene definito nello script logger.js grazie all'utilizzo del già introdotto modulo winstonJS. Lo script mette a disposizione la possibilità di generare diversi file di log multilivello per il quale si è scelto di mantenere l'impostazione piramidale di default: ad ogni tipologia di log (errore, sistema, transazioni, …) viene associato un livello. Ad ogni livello corrisponde la possibilità di definire un output su file in cui vengono scritti i log di livello uguale o inferiore al livello del canale scelto. Ad esempio, se per quanto riguarda i log Off-Chain mi trovo nel file che definisce le interazioni definite come livello action, che definiscono le azioni (di scrittura) di un utente su database o blockchain, troverò nel file anche i log di errore e i log di sistema che gli sono superiori in priorità (quindi inferiori in livello). Non saranno presenti i log di livello info, quindi. Nel file relativo ai log info saranno presenti invece tutti i log Off-Chain.

Nonostante il codice di progetto non se ne occupi per semplicità, il sistema mette a disposizione dei log totali in formato *.json* in modo che, nel caso di violazione o studi successivi, sia possibile effettuarne il parsing agevolmente.

Session.js

Il controller si occupa di gestire il rendering delle pagine generato a partire dalle richieste degli utenti che visitano la piattaforma e ai permessi che questi possiedono. È quindi il vero e proprio controller web del nostro server ed è lo script che più comunica con le viste. Lo script si occupa quindi di interfacciarsi al bisogno con gli altri controller e di ottenere le informazioni relative alla gestione della pagina.

Per ogni vista dell'applicativo sarà presente una funzione che ne codificherà la corretta gestione. È quindi questo script ad effettuare la preparazione delle informazioni da raccogliere prima di rispondere con la pagina web richiesta ed è quindi in questa porzione di codice che si trova la maggior parte delle istruzioni che vengono utilizzate per la prevenzione del lateral movement e della privilege escalation. I sistemi di prevenzione per queste vulnerabilità si avvalgono di semplici strutture *if-else* che verifichino gli effettivi permessi dell'utente, nonché la sua pagina di provenienza. Le strutture appena descritte forniscono quindi delle variabili chiave al motore di generazione delle pagine che si occuperà in fase di rendering di fornire o meno all'utente alcuni blocchi html al fine di garantire sempre un corretto information flow. Nel caso invece per il quale il visitatore abbia contattato esplicitamente una pagina per il quale non ha l'accesso o i permessi necessari, lo script si occupa di rispondere con la pagina home su cui viene visualizzato un messaggio che indica che la funzionalità non è permessa.

Oltre a questi, il codice dello script si occupa di effettuare controlli più banali relativi a problematiche generali di implementazione per i quali si rimanda alla visione del codice.

Viste

Infine, gran parte del resto dell'applicativo si traduce nell'interfaccia grafica realizzata attraverso l'uso del motore HBS per la generazione di viste statiche a partire da template scritti in html. Come già brevemente anticipato, con l'utilizzo di alcuni helper specifici del linguaggio hbs è possibile incapsulare blocchi di codice html in strutture che si deciderà di mostrare o meno all'utente sulla base delle informazioni passate in fase di rendering della pagina stessa.

L'utilizzo di questa soluzione ci ha permesso di codificare la stessa pagina per diverse tipologie di utente a partire dal livello dei privilegi che questo porta con sé.

Note generiche

Il corpo dell'applicazione presenta diverse problematiche che si è scelto di risolvere con diversi approcci. Tra gli approcci più importanti da citare, ve ne sono un paio che il gruppo di progetto crede essere sostanziali a garantire la sicurezza dell'applicativo e a dimostrare le potenzialità che le scelte fatte hanno reso disponibili.

Il primo approccio corrisponde a quello già brevemente introdotto dell'individuazione di alcuni Aspect chiave, intesi come gli aspetti dell'Aspect Oriented Programming, da proteggere o gestire in maniera efficiente. Sono state citate problematiche e soluzioni di un paio di casi in cui l'approccio è stato utilizzato, ma nell'interezza del codice queste sono molteplici. Si è scelto quindi di coprire queste vulnerabilità manualmente, senza servirsi di interpreti, poiché non sono state trovate risorse con cui effettuare questo tipo di preprocessing e perché si è voluto avere il pieno controllo sulla codifica dell'applicazione, nonostante si consiglia l'utilizzo di software in grado di gestire in maniera più efficiente l'AOP se si dovesse discutere l'implementazione del server Web in ambito production.

Il secondo approccio da citare per la gestione di alcune minacce è l'implementazione in ridondanza e quando possibile in diversità di controlli sul codice o di funzioni di gestione. Ad esempio, nell'implementazione di password sicure, si è scelto di inserire il controllo della presenza di caratteri alfanumerici maiuscoli e di punti sia a livello di codice hbs (html) sia a livello di controller javascript. In questo modo, se un utente riuscisse a superare le difese del codice html, dovrebbe scontrarsi con la difficoltà di superare lo stesso controllo in un altro linguaggio. Altro esempio sta nel fatto che per limitare la quantità di biglietti acquistabili e rispondere così all'Availability dei biglietti, sono stati implementati controlli che impediscono l'acquisto di più di cinque biglietti sia a livello di smart-contract definiti sulla blockchain, sia a livello di implementazione del codice hbs.

Non si citano inoltre moltissimi altri aspetti di "buona programmazione" che sono stati utilizzati per non appesantire troppo la trattazione. Si rimanda quindi all'ispezione del codice corredato a questa relazione.

Testing e collaudi

Per garantire una buona qualità del software e che questo risponda ai diversi requisiti di sicurezza prevista, sono stati effettuati diversi test di funzionamento e prove del codice.

Nell'impossibilità di definire nel dettaglio come e quali test sono stati effettuati ne verrà descritta la principale modalità con cui questi sono stati portati a termine.

Le modalità in cui sono stati effettuati i test derivano fortemente dalle condizioni nelle quali il codice è stato scritto. I test svolti sono ovviamente successivi ad un primissimo controllo del funzionamento della funzione appena implementata da parte del programmatore che l'ha scritta.

Ad esempio, per quanto riguarda la scrittura dell'interfaccia grafica, solitamente si è scelto di lavorare in un gruppo di due o tre persone, in modo da avere la possibilità di concentrarsi su diversi aspetti di visualizzazione; erano quindi i membri residui del gruppo, magari singolarmente, a testare il funzionamento delle funzioni di visualizzazione e navigazione implementate dagli altri, concentrandosi soprattutto sull'information flow e l'usabilità del sistema.

Allo stesso modo, dell'implementazione delle funzioni del backend che di solito era svolta singolarmente ne veniva testato il corretto funzionamento dal gruppo che poi ne scriveva l'interfaccia grafica o da chi implementava le funzioni che vi si collegavano. Erano altri quindi ad occuparsi della revisione del codice e a proporre nuovi sistemi di controllo o l'implementazione di policy di sicurezza aggiuntive.

Giunti al completamento della prima versione del software si sono effettuati tre successivi collaudi totali in cui il gruppo si è riunito al completo e ogni membro mostrava completamente le funzionalità ottenute fino alla replicazione di ogni caso d'uso del software mostrando per punti sia l'esecuzione corretta sia il risultato di eventuali errori, umani o di software, che potevano occorrere all'esecuzione. Terminato il collaudo si provvedeva quindi a correggere le vulnerabilità insorte e ad implementare le funzioni mancanti, da collaudare in una riunione successiva.

Si è giunti infine ad una versione del software funzionante che secondo il gruppo di progetto risponde ottimamente ai requisiti funzionali e di sicurezza definiti in fase di progettazione.
A corredo del software si è deciso, come fase ultima, di scrivere una breve guida al software allegata al progetto intero.