



Università
di Catania

DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA TRIENNALE IN INFORMATICA

Paolo Maria Scarlata

Analisi e Sviluppo di Competizioni Capture The Flag come Strumento Didattico

RELAZIONE PROGETTO FINALE

Relatore:
Chiar.mo Dott. Sergio Esposito

Anno Accademico 2024 – 2025

Indice

Elenco delle figure	VIII
1 Introduzione	5
2 Background e stato dell'arte	9
2.1 Vulnerabilità web e classificazione	9
2.2 Capture the flag: evoluzione e metodologie	11
2.2.1 Storia e tipologie delle competizioni CTF	11
2.2.2 Design di challenge efficaci	13
2.3 Stato dell'arte e gap identificati	15
3 Challenge realizzate	20
3.1 Infrastruttura comune e architettura condivisa	20
3.1.1 JSON Web Token	20
3.1.2 Gestione utenti e database hardcoded	23
3.1.3 Stack tecnologico	24
3.1.4 Interfaccia utente e design pattern condivisi	25

3.1.5	Containerizzazione e deployment	25
3.2	Metodologia di sviluppo	27
3.2.1	Struttura del progetto e organizzazione	29
3.3	Challenge 1: IDOR con WAF bypass - Processo di sviluppo . . .	31
3.3.1	Concept design e definizione degli obiettivi educativi . .	31
3.3.2	Implementazione del sistema di autenticazione	33
3.3.3	Sviluppo del Web Application Firewall	34
3.3.4	Struttura dati e posizionamento della flag	35
3.3.5	Implementazione degli endpoint API	37
3.3.6	Workflow di exploit IDOR	38
3.3.7	Mitigazioni e considerazioni educative	45
3.4	Challenge 2: JWK Header Parameter Injection	48
3.4.1	Concept design e definizione degli obiettivi educativi . .	48
3.4.2	Workflow dell'exploit	53
3.4.3	Mitigazioni e considerazioni educative	58
3.5	Challenge 3: Information Disclosure e Weak Cryptography . . .	61
3.5.1	Concept design, superficie d'attacco e vulnerabilità di backup	61
3.5.2	Contesto tecnico: l'hashing delle password	63
3.5.3	Workflow dell'exploit	67
3.5.4	Possibili mitigazioni	72

4	Integrazione con la piattaforma CTF	74
4.1	Obiettivi dell'integrazione	74
5	Conclusioni	78

Ringraziamenti

In primo luogo, i miei genitori. Grazie a loro ho avuto le possibilità che mi hanno permesso di intraprendere questo viaggio e di portarlo a termine. È grazie al loro sostegno, alla loro pazienza e ai sacrifici che hanno fatto per me, se posso festeggiare questo risultato.

Un ringraziamento speciale va a mia sorella Gloria, che è sempre stata una presenza fondamentale per il mio equilibrio emotivo. Durante il mio percorso, ha saputo darmi conforto, ascolto e affetto incondizionato.

Ringrazio anche mio fratello Dario, che nella mia vita ha sempre rappresentato un punto di riferimento, ma soprattutto, un rivale. È stato proprio questo senso di sfida, questa voglia di “superarlo”, a darmi spesso la spinta per fare di più e per crescere, difatti devo a lui la mia vena agonistica.

Un grazie enorme va ai miei amici, alla mia comitiva, che in tutti questi anni sono stati una seconda famiglia, e non è tanto per dire ma lo penso davvero, una famiglia numerosa e problematica, ma tenuta in piedi da legami inscindibili. Mi hanno sempre appoggiato, sia quando avevo bisogno di un passaggio, sia quando servivano parole di conforto o semplicemente una distrazione. Sono stati sempre presenti, e mi hanno sempre dato tutto ciò di cui avevo bisogno. Molti esami sono riuscito a sostenerli anche grazie al loro aiuto costante, alla disponibilità, ai passaggi (Grazie Luigi) e alla loro generosità, senza mai farsi problemi a darmi una mano.

Vorrei ringraziare anche Maria, negli ultimi tre anni ha sopportato un numero inimmaginabile di miei lamenti: sullo studio, sull’università, sullo stress e su

qualsiasi altra preoccupazione (di famiglia ci piace lamentarci). È sempre stata lei ad ascoltarmi e a rispondermi con un sorriso, ostinandosi a ripetermi che tutto sarebbe andato bene. Le sono immensamente grato per esserci stata.

E infine, per ultimo vorrei ringraziare colui con il quale ho condiviso più di tutti, Marco. Abbiamo affrontato insieme questo percorso universitario e a tratti infernale, passo dopo passo, esame dopo esame. Sono certo che se tu non avessi scelto questa strada, oggi io non sarei qui a festeggiare. Sei stato il mio compagno di vita, il mio “simbionte” in questi ultimi quattro anni. Il nostro rapporto potrebbe essere esplicitato dall’analogia di due scalatori, uniti dalla medesima corda, consapevoli che, nel momento in cui uno dovesse scivolare, non precipiterebbe nel vuoto perché l’altro lo trattiene e lo sorregge con il legame che li unisce. Abbiamo condiviso questo percorso, affrontando fianco a fianco, sostenendoci, consigliandoci, aiutandoci e, talvolta anche rosicando l’uno dell’altro, difatti, proprio come quando giochiamo insieme ai videogiochi, che essi siano fps, scacchi, strategici, di combattimento o di corse, ti batto sempre. E a dimostrazione di ciò, anche oggi è andata così. Come ogni volta ti aspetto qui, al traguardo.

Grazie a tutti voi, non solo per essere qui oggi, ma anche per far parte della mia vita.

Elenco delle figure

3.1	Struttura del token JWT. Fonte: <i>Components of JWTs Explained</i> [33].	21
3.2	Struttura dei file delle challenge CTF	29
3.3	Schermata di CTFd per l'accesso alla challenge	39
3.4	Intercettazione delle richieste HTTP tramite Burp Suite	40
3.5	Richiesta GET per <code>/api/users/2/data</code>	41
3.6	Tentativo di accesso a <code>/api/users/3/data</code> intercettato con Burp Suite, con risposta 403 dal WAF che indica un pattern sospetto.	42
3.7	Richiesta GET per <code>/api/users/%33/data</code>	43
3.8	Configurazione Sniper attack	44
3.9	Esito Sniper attack	45
3.10	Negazione accesso al pannello di controllo	54
3.11	Analisi pacchetto GET <code>/admin</code>	55
3.12	Tentativo di Token tampering fallito	56
3.13	Generazione coppia di chiavi RSA	57
3.14	Firma del token con chiave RS256	58

3.15 Tentativo di token tampering con esito positivo	59
3.16 Struttura della terza challenge	61
3.17 Accesso alla terza challenge	67
3.18 Esito ottenuto con gobuster	68
3.19 Download di server.js.bak	69
3.20 Analisi server.js.bak	70
3.21 Esito ottenuto con John The Ripper	71
4.1 Challenge presenti su CTFd	74
4.2 Statistiche CTFd	76

Glossario

Autenticazione	Verifica dell'identità di un utente, qui implementata tramite JSON Web Token (JWT).
Autorizzazione	Controllo dei permessi di accesso alle risorse; nella challenge IDOR è volutamente assente.
Backup (.bak)	Copia di un file sorgente lasciata sul server (<code>server.js.bak</code>); se accessibile pubblicamente può rivelare codice e dati sensibili.
Burp Suite	Strumento per test di sicurezza web, utile per analizzare e manipolare richieste HTTP.
Bypass	Tecnica per aggirare controlli di sicurezza (p. es. WAF) sfruttando debolezze logiche.
Challenge (CTF)	Esercizio pratico di sicurezza informatica in cui si cerca di trovare una <code>FLAG{}</code> .
Containerizzazione	Impacchettamento di un'applicazione e delle sue dipendenze in un ambiente isolato (Docker).
Content Discovery	Enumerazione di directory e file nascosti per individuare risorse esposte (scanner, wordlist, ecc.).
CTFd	Piattaforma open-source per gestire competizioni Capture The Flag.
Docker	Strumento per creare e gestire container software.
Docker Compose	Tool per l'orchestrazione di più container Docker tramite file YAML.
Endpoint	Percorso API usato per accedere a funzionalità di backend.

Express.js	Framework web per Node.js impiegato nel backend delle challenge.
Flag	Stringa di testo (formato FLAG{}) nascosta all'interno della challenge e obiettivo dell'attaccante.
Gobuster	Strumento a riga di comando per content discovery di URL, directory e file.
Hard-coded	Dato scritto direttamente nel codice, non modificabile in fase di esecuzione.
HS256	Algoritmo di firma simmetrica HMAC-SHA256 per JWT; richiede la stessa chiave segreta per firma e verifica.
IDOR	<i>Insecure Direct Object Reference</i> : vulnerabilità dovuta all'assenza di controlli di autorizzazione sugli identificatori.
Information Disclosure	Esposizione involontaria di informazioni sensibili a causa di configurazioni errate.
John the Ripper	Tool di password-cracking (dictionary, brute-force, rainbow table) per invertire hash.
JWK	<i>JSON Web Key</i> : rappresentazione JSON di una chiave crittografica; può essere incorporata nell'header di un JWT.
JWT	<i>JSON Web Token</i> : token compatto (firmato con HS256, RS256, ecc.) per l'autenticazione e la trasmissione di claim.
kid	<i>Key ID</i> : parametro dell'header JWT/JWK che identifica in modo univoco la chiave utilizzata.
MD5	<i>Message Digest 5</i> : algoritmo di hash a 128 bit deprecato, vulnerabile a collisioni e inversione rapida.
Middleware	Componente che intercetta le richieste HTTP per aggiungere logica (p. es. autenticazione o WAF).

NIST	<i>National Institute of Standards and Technology</i> : ente USA che pubblica linee guida crittografiche (SP 800-XXX).
Node.js	Runtime JavaScript lato server, usato per sviluppare le challenge.
OWASP	<i>Open Web Application Security Project</i> : community che redige linee guida e Top Ten sulle vulnerabilità web.
Path Traversal	Tecnica che sfrutta sequenze <code>../</code> per accedere a file fuori dalla directory prevista.
PKCS#8	Standard per la codifica di chiavi private, qui usato per la chiave RSA del server.
Rainbow Table	Tabelle pre-calcolate hash → password utili a invertire hash non salati più velocemente.
Rate Limiting	Meccanismo per limitare il numero di richieste in un intervallo di tempo.
RSA	Sistema di crittografia asimmetrica basato su coppie di chiavi pubblica/privata; supporta firme come RS256.
RS256	Algoritmo di firma asimmetrica RSA-SHA256 per JWT; firma con chiave privata e verifica con chiave pubblica.
Salt	Valore casuale concatenato alla password prima dell'hash per mitigare attacchi di pre-computazione.
Signature Bypass	Tecnica che elude la verifica della firma digitale di un token o messaggio (p. es. algorithm confusion, JWK injection), consentendo l'uso di dati non autenticati.
SPKI	<i>Subject Public Key Info</i> : formato standard per la codifica di chiavi pubbliche in PEM.
Token Tampering	Alterazione intenzionale di un token (p. es. JWT) modificandone header o payload per ottenere privilegi non autorizzati o aggirare controlli di integrità.

URL Encoding	Codifica dei caratteri in URL (p. es. $3 \rightarrow \%33$), talvolta usata per aggirare controlli.
User-Agent	Intestazione HTTP che identifica il software client (browser, Burp, ecc.).
UUID	Identificatore univoco universale, utile per evitare enumerazioni prevedibili sugli ID.
Vulnerabilità	Debolezza sfruttabile in un sistema.
Weak Cryptography	Uso di algoritmi o parametri deboli/obsoleti (es. MD5) che non garantiscono più adeguata sicurezza.
WAF	<i>Web Application Firewall</i> : filtro HTTP per bloccare attacchi comuni.

Capitolo 1

Introduzione

Il panorama della cybersecurity contemporaneo presenta sfide in continua evoluzione, caratterizzate da un divario crescente tra la velocità di emergenza di nuove vulnerabilità e la capacità di sviluppare contromisure adeguate [1]. I sistemi informatici moderni, nonostante decenni di progressi nella ricerca sulla sicurezza, continuano a manifestare vulnerabilità che vengono sistematicamente sfruttate da attaccanti sempre più sofisticati [2]. Questa situazione evidenzia una problematica strutturale: la formazione di professionisti della cybersecurity richiede approcci che vadano oltre i metodi didattici tradizionali [3].

I metodi convenzionali di insegnamento della sicurezza informatica presentano limitazioni significative nel preparare i professionisti alle sfide reali del settore. Le lezioni teoriche, pur fornendo basi concettuali solide, spesso non riescono a trasmettere la comprensione pratica necessaria per identificare e sfruttare vulnerabilità in contesti realistici. I laboratori tradizionali, vincolati da considerazioni di sicurezza e limitazioni tecniche, offrono esperienze controllate che non sempre riflettono la complessità degli scenari di attacco contemporanei [4].

Le competizioni Capture The Flag rappresentano un approccio promettente per colmare questo divario formativo, offrendo ambienti controllati dove è possibile sperimentare con vulnerabilità reali in sicurezza. Tuttavia, la creazione di challenge efficaci presenta sfide impegnative che richiedono competenze multidisciplinari, dalla programmazione alla sicurezza informatica, dalla gestione di

sistemi alla progettazione pedagogica.

Le challenge vengono spesso create per eventi specifici e successivamente abbandonate, disperdendo il lavoro investito nella loro progettazione e implementazione. Questa pratica rappresenta un'inefficienza che impedisce il miglioramento continuo della qualità formativa e costringe organizzatori futuri a ripetere processi di sviluppo già sperimentati.

L'espansione di piattaforme e framework per la gestione di competizioni CTF ha democratizzato l'accesso a questi strumenti formativi [5], ma ha anche evidenziato una carenza importante: la mancanza di standardizzazione nei processi di creazione e distribuzione delle challenge. La conseguenza è una duplicazione di sforzi, dove organizzatori diversi affrontano ripetutamente gli stessi problemi tecnici invece di concentrarsi sul miglioramento della qualità pedagogica del contenuto. Questo approccio frammentato limita l'efficacia complessiva dell'ecosistema CTF e la possibilità di costruire un patrimonio di conoscenze condivise.

La containerizzazione, attraverso tecnologie come Docker, offre un meccanismo per incapsulare ambienti complessi in unità di distribuzione standardizzate e facilmente riproducibili grazie alla portabilità [6]. È necessario bilanciare i requisiti di isolamento e sicurezza con le esigenze di accessibilità e prestazioni, garantendo che la containerizzazione non introduca complessità che potrebbero compromettere gli obiettivi pedagogici delle challenge.

L'approccio containerizzato offre l'opportunità di migliorare questa situazione, ma richiede una documentazione sistematica che catturi non solo gli aspetti tecnici dell'implementazione, ma anche le decisioni progettuali e le lezioni apprese durante il processo di sviluppo. Solo attraverso questa sistematizzazione è possibile costruire un corpo di conoscenza che permetta di evitare la ripetizione di errori comuni e favorisca l'accumulo di best practices [7].

L'obiettivo generale di questo lavoro è la creazione di un framework documentato per la progettazione, implementazione e distribuzione di challenge CTF containerizzate. Questo framework deve fornire indicazioni sufficientemente dettagliate per permettere la replicazione del processo, mantenendo al contempo la flessibilità necessaria per adattarsi a contesti e requisiti diversi.

Gli obiettivi specifici includono l'implementazione di un insieme rappresentativo di challenge che coprano diverse tipologie di vulnerabilità e scenari di attacco [8]. Ogni challenge deve essere progettata per funzionare correttamente dal punto di vista tecnico e per essere didatticamente efficace, offrendo un percorso di apprendimento che guidi i partecipanti dalla scoperta iniziale della vulnerabilità fino alla comprensione completa dei meccanismi di sfruttamento.

Un obiettivo aggiuntivo è la validazione dell'approccio attraverso l'integrazione con piattaforme CTF esistenti. Questo processo di integrazione permetterà di identificare eventuali limitazioni dell'approccio proposto e di raccogliere indicazioni per miglioramenti futuri. È importante che le challenge non rimangano esempi teorici, ma dimostrino la loro applicabilità pratica in contesti reali di competizione o formazione.

La metodologia adottata in questo lavoro è di natura sperimentale e iterativa, riflettendo la natura esplorativa del problema affrontato.

L'approccio riconosce che ogni challenge rappresenta un caso di studio con specificità tecniche e pedagogiche proprie. Tuttavia, attraverso l'analisi sistematica di multiple implementazioni, è possibile identificare pattern comuni e sviluppare best practices che possano essere applicate in contesti diversi.

I contributi attesi da questo lavoro si articolano su diverse dimensioni. Dal punto di vista tecnico, il lavoro fornirà una raccolta di challenge CTF funzionali e pronte per l'uso, complete dei rispettivi Dockerfile e della documentazione necessaria per il deployment e la manutenzione.

Dal punto di vista metodologico, il contributo principale sarà la sistematizzazione del processo di dockerizzazione di challenge CTF, con particolare attenzione alla documentazione delle decisioni progettuali e delle lezioni apprese durante l'implementazione.

Dal punto di vista didattico, ogni challenge implementata rappresenterà un caso di studio che illustra non solo come sfruttare una specifica vulnerabilità, ma anche come progettare un'esperienza di apprendimento efficace. Questo aspetto riveste particolare importanza considerando che il successo di una challenge CTF dipende tanto dalla correttezza tecnica quanto dalla qualità dell'esperienza didattica offerta ai partecipanti.

Il presente lavoro è organizzato in 3 capitoli principali, ciascuno dei quali affronta un aspetto specifico del problema. Il capitolo di Background fornisce il contesto teorico necessario per la comprensione del lavoro, introducendo i concetti fondamentali relativi alle vulnerabilità informatiche, alle competizioni CTF, e alle tecnologie di containerizzazione.

Il capitolo Challenge Realizzate costituisce il nucleo tecnico del lavoro, documentando in dettaglio il processo di progettazione e implementazione di ogni challenge. Questo capitolo include l'analisi delle tecnologie utilizzate, la descrizione delle vulnerabilità implementate, la documentazione degli exploit sviluppati e l'identificazione delle possibili mitigazioni.

Il capitolo sull'Integrazione con la piattaforma CTF descrive il processo di testing e validazione delle challenge in ambiente operativo, documentando le procedure di deployment e le considerazioni relative alla gestione delle challenge in contesti di competizione reale.

Capitolo 2

Background e stato dell'arte

2.1 Vulnerabilità web e classificazione

La sicurezza delle applicazioni web è una disciplina in continua evoluzione, segnata da tecniche di attacco sempre più sofisticate e da contromisure in costante aggiornamento. In questo contesto, il termine “vulnerabilità” indica una debolezza in un software o nella sua configurazione che può essere sfruttata per compromettere la riservatezza, l'integrità o la disponibilità dei dati o dei servizi. È importante distinguere fra debolezza, vulnerabilità ed esposizione: la debolezza è la causa tecnica che può dare origine a una o più vulnerabilità; la vulnerabilità è il difetto effettivamente sfruttabile; l'esposizione riguarda configurazioni o condizioni che ampliano la superficie d'attacco. Questa distinzione è formalizzata nella tassonomia del Common Weakness Enumeration (CWE), mantenuta da MITRE [9].

Fra i riferimenti più autorevoli per classificare le vulnerabilità web vi è l'OWASP Top Ten [10], la cui ultima edizione stabile risale al 2021. Oltre a confermare rischi noti, come il Cross-Site Scripting (XSS), ha introdotto nuove categorie, fra cui Insecure Design e Server-Side Request Forgery (SSRF), e ha ridefinito concetti come la Sensitive Data Exposure, oggi trattata come Cryptographic Failures. L'obiettivo del progetto non è solo stilare un elenco delle principali minacce, ma anche guidare sviluppatori e professionisti verso buone pratiche di

progettazione e revisione del codice.

Sul piano delle cause tecniche, il CWE Top 25 del 2024 individua le debolezze software più pericolose sulla base della loro frequenza e dell'impatto potenziale, analizzando oltre 31.770 vulnerabilità censite nel National Vulnerability Database (NVD), con dati aggregati da MITRE e dalla Cybersecurity and Infrastructure Security Agency (CISA) [9]. In cima alla classifica figura la Out-of-Bounds Write (CWE-787), responsabile dell'8,55% del rischio complessivo, che può causare corruzione della memoria ed esecuzione arbitraria di codice. Seguono il Cross-Site Scripting (CWE-79), con il 6,98% del rischio totale, che consente l'iniezione di script malevoli nel browser delle vittime, e la Out-of-Bounds Read (CWE-125), meno frequente ma insidiosa perché può rivelare informazioni riservate o facilitare exploit più complessi.

La metodologia di calcolo del CWE Top 25 integra il numero di vulnerabilità note, i punteggi di severità derivati dal Common Vulnerability Scoring System (CVSS) e la rilevanza delle minacce nel contesto reale. Il CVSS, nella versione 4.0 rilasciata nel 2023, introduce importanti innovazioni, fra cui la separazione fra metriche di base, di minaccia e ambientali, per consentire una valutazione del rischio più aderente agli specifici scenari operativi [11]. Questa metrica consente di assegnare un punteggio numerico alla gravità delle vulnerabilità, supportando la definizione di priorità negli interventi di mitigazione.

Completa il quadro il contributo del National Institute of Standards and Technology (NIST), il cui documento SP 800-115 offre linee guida dettagliate per la conduzione di test di sicurezza e attività di penetration testing [12].

L'integrazione di framework come OWASP Top Ten, CWE Top 25 e CVSS consente di correlare le debolezze tecniche alle minacce pratiche, permettendo di definire soglie di rischio accettabile basate su metriche oggettive. Questo approccio multilivello costituisce un elemento essenziale nella pianificazione delle strategie di sicurezza e nella gestione del rischio lungo l'intero ciclo di vita del software (SDLC).

2.2 Capture the flag: evoluzione e metodologie

2.2.1 Storia e tipologie delle competizioni CTF

Le competizioni Capture The Flag rappresentano un'evoluzione naturale delle tradizionali esercitazioni militari di addestramento, adattate al dominio digitale per rispondere alle crescenti minacce informatiche. La prima manifestazione moderna di queste competizioni può essere ricondotta alla DEF CON del 1996 [13], dove la comunità hacker statunitense iniziò a formalizzare competizioni basate sulla risoluzione di problemi di sicurezza informatica in tempo reale. Questa transizione dal concetto fisico di "cattura la bandiera" a quello digitale ha rappresentato un punto di svolta nella didattica della cybersecurity, introducendo elementi di gamification che si sono rivelati particolarmente efficaci per l'apprendimento di competenze tecniche complesse.

L'evoluzione delle competizioni CTF ha raggiunto una legittimazione accademica significativa con il DARPA Cyber Grand Challenge del 2016[14], che ha dimostrato come queste metodologie competitive potessero essere applicate non solo per l'intrattenimento della comunità hacker, ma anche per la ricerca avanzata e lo sviluppo di sistemi autonomi di difesa informatica. Questo evento ha segnato il riconoscimento formale delle CTF come strumento didattico validato scientificamente, aprendo la strada all'adozione sistematica di queste metodologie negli ambienti accademici e professionali.

La transizione delle competizioni CTF da eventi ludici a strumenti educativi formali si è iniziata principalmente nel periodo 2000-2010, quando le istituzioni universitarie hanno iniziato a riconoscere il potenziale didattico di questi ambienti competitivi. Le università hanno progressivamente integrato le metodologie CTF nei curricula di cybersecurity, sfruttando la capacità di queste competizioni di coinvolgere attivamente gli studenti in scenari pratici che riproducono fedelmente le sfide del mondo professionale. Parallelamente, il settore privato ha iniziato a adottare format CTF per la formazione continua del

personale tecnico, riconoscendo l'efficacia di questi approcci nella preparazione di team di sicurezza informatica capaci di affrontare minacce evolute.

Le tipologie moderne di competizioni CTF si distinguono principalmente in due categorie fondamentali che riflettono filosofie didattiche e obiettivi formativi differenti. Il formato Jeopardy-style si caratterizza per una struttura organizzata in categorie tematiche, dove ogni challenge rappresenta un problema isolato con obiettivi specifici e una flag associata. Questa metodologia favorisce la scalabilità dell'evento, permettendo la partecipazione simultanea di centinaia di concorrenti senza limitazioni logistiche significative, e facilita l'assessment individuale delle competenze attraverso sistemi di scoring automatizzati. La natura modulare delle challenge Jeopardy-style consente inoltre una flessibilità didattica notevole, permettendo ai docenti di selezionare specifiche tipologie di vulnerabilità in base agli obiettivi curricolari del corso.

Il formato Attack/Defense rappresenta un approccio alternativo che privilegia il realismo degli scenari attraverso la simulazione di reti aziendali complesse, dove team concorrenti devono simultaneamente difendere la propria infrastruttura e attaccare quella degli avversari. Questa metodologia offre un livello di autenticità superiore rispetto agli scenari Jeopardy-style, riproducendo fedelmente le dinamiche operative dei Security Operations Center reali e promuovendo competenze di teamwork essenziali per l'ambiente professionale. Tuttavia, la complessità logistica degli eventi Attack/Defense e la difficoltà nell'assessment granulare delle performance individuali limitano significativamente l'applicabilità di questo formato in contesti didattici standard.

Esistono anche implementazioni ibride moderne che tentano di combinare i vantaggi di entrambi gli approcci, introducendo elementi di persistenza e interazione tra challenge mantenendo la gestibilità del formato Jeopardy-style. Queste soluzioni rappresentano un'evoluzione promettente per l'applicazione delle metodologie CTF in ambienti educativi, permettendo di bilanciare efficacemente realismo operativo e praticabilità didattica.

Il presente lavoro adotta il formato Jeopardy-style per massimizzare l'accessibilità didattica e la reproducibilità dell'ambiente di apprendimento, mantenendo elementi di realismo attraverso la progettazione di scenari che simulano quelli aziendali, riflettendo le problematiche di sicurezza del settore più comuni. Questa scelta metodologica permette di concentrare l'attenzione educativa sui meccanismi specifici delle vulnerabilità implementate, evitando la complessità gestionale associata a formati più complessi che potrebbero distrarre dagli obiettivi formativi primari.

2.2.2 Design di challenge efficaci

La progettazione di challenge CTF educativamente efficaci richiede l'applicazione sistematica di principi didattici consolidati che favoriscano l'apprendimento attivo e la retention delle competenze acquisite, molte teorie di apprendimento famose si prestano alla perfezione al caso di apprendimento offerto dalle CTF. Il framework del constructivist learning, sviluppato da Jean Piaget[15] e successivamente raffinato da Jerome Bruner[16], fornisce la base teorica per comprendere come gli studenti costruiscano attivamente la propria comprensione delle vulnerabilità informatiche attraverso l'esperienza diretta di exploitation. Questo approccio didattico si rivela particolarmente appropriato per la cybersecurity, dove la comprensione profonda dei meccanismi di attacco richiede necessariamente un'esperienza pratica che permetta agli studenti di sperimentare direttamente le conseguenze delle configurazioni insicure e delle logiche di business inadeguate.

La Flow Theory, formulata da Mihaly Csikszentmihalyi[17], offre un framework psicologico essenziale per bilanciare la difficoltà delle challenge con il livello di competenza degli studenti. Secondo questa teoria, l'apprendimento ottimale si verifica quando esiste un equilibrio dinamico tra la complessità del compito e le capacità dell'individuo, creando uno stato di coinvolgimento profondo caratterizzato da concentrazione intensa e forte motivazione. L'applicazione di

questi principi al design di challenge CTF implica la necessità di implementare meccanismi di difficoltà progressiva che permettano agli studenti di sviluppare competenze in maniera graduale, evitando sia la frustrazione associata a challenge eccessivamente complesse sia la noia derivante da esercizi troppo elementari.

Il concetto di scaffolding educativo, introdotto da Lev Vygotsky nella sua teoria della zona di sviluppo prossimale[18], fornisce un framework operativo per la progettazione di sistemi di supporto che guidino gli studenti attraverso il processo di discovery senza compromettere l'esperienza di apprendimento autonomo. L'implementazione efficace di scaffolding nelle challenge CTF richiede un bilanciamento delicato tra guida e indipendenza, utilizzando meccanismi come suggerimenti progressivi, messaggi di errore specifici e informativi, feedback immediato per mantenere gli studenti all'interno della loro zona di sviluppo ottimale.

I principi di game design applicati alle competizioni CTF derivano dalla ricerca del coinvolgimento e della motivazione nei contesti ludici educativi. La definizione di obiettivi chiari e misurabili, tipicamente rappresentati dal formato standardizzato delle flag, fornisce agli studenti un senso di direzione e risultati che sostengono la motivazione e l'umore alto durante il processo di problem-solving. Il feedback immediato, implementato attraverso sistemi di validazione automatica delle flag, soddisfa il bisogno psicologico di riconoscimento del progresso e permette agli studenti di calibrare le proprie strategie di apprendimento in tempo reale.

Una sequenza ben progettata dovrebbe introdurre concetti fondamentali attraverso challenge relativamente semplici, consolidare queste competenze attraverso variazioni incrementali, e infine sfidare gli studenti con scenari multi-stage che richiedono l'integrazione di multiple tecniche apprese precedentemente. Questa progressione riflette i principi della Bloom's Taxonomy[19], guidando gli studenti attraverso livelli crescenti delle competenze mnemoniche e cognitive fino ai livelli superiori di analisi, valutazione e creazione.

L'importanza della progettazione narrativa nelle challenge CTF è spesso sottovalutata ma rappresenta un fattore determinante per il coinvolgimento degli studenti e il trasferimento delle competenze al contesto professionale. Challenge contestualizzate all'interno di scenari aziendali realistici offrono agli studenti un ambiente cognitivo collaudato che facilita la comprensione delle implicazioni delle vulnerabilità tecniche, preparandoli meglio per l'applicazione delle competenze acquisite in ambienti professionali reali.

Il bilanciamento tra realismo tecnico e accessibilità didattica costituisce una delle sfide più complesse nella progettazione di challenge educative. Scenari eccessivamente semplificati rischiano di fornire una comprensione superficiale delle vulnerabilità, mentre implementazioni troppo complesse possono sovraccaricare cognitivamente gli studenti e compromettere l'apprendimento dei concetti fondamentali. La ricerca di questo equilibrio richiede una comprensione approfondita sia del background tecnico degli studenti sia degli obiettivi formativi specifici del corso.

L'implementazione delle challenge presentate in questo lavoro cerca di applicare alcuni di questi principi teorici, utilizzando scenari aziendali come BankSecure Corp. (azienda fittizia creata ad hoc), per fornire contesto narrativo e implementando meccanismi come suggerimenti strategici per orientare il processo di scoperta. L'approccio adottato tenta di bilanciare realismo e accessibilità attraverso la containerizzazione di ambienti vulnerabili e controllati, pur riconoscendo che l'efficacia di queste scelte progettuali richiederà validazione attraverso l'utilizzo in contesti didattici reali.

2.3 Stato dell'arte e gap identificati

La crescente popolarità delle competizioni Capture The Flag (CTF) ha favorito negli ultimi anni lo sviluppo di numerosi strumenti, piattaforme e linee di ricerca dedicate alla formazione pratica nel campo della cybersecurity. A livello internazionale, diverse piattaforme open-source sono emerse per supportare la

creazione e la gestione delle challenge CTF. Tra queste:

- **CTFd** è una piattaforma scritta in Python/Flask pensata principalmente per eventi di tipo Jeopardy-style. Consente di creare e gestire challenge tramite interfaccia web, integra scoreboard, gestione dei team, suggerimenti (hint) e allegati. Supporta inoltre plugin per autenticazione esterna (ad esempio OAuth o LDAP), temi personalizzati e mette a disposizione API REST per l'integrazione con sistemi esterni. È apprezzata per la facilità di deploy, anche tramite Docker, e per la possibilità di estendere le sue funzionalità. Tuttavia, non è pensata nativamente per scenari Attack/Defense [20].
- **FBCTF**, sviluppata da Facebook, è progettata sia per eventi Jeopardy-style sia per modalità Attack/Defense. Utilizza PHP e HackLang con database MySQL e introduce una mappa interattiva sulla quale sono distribuite le challenge, creando un approccio più dinamico e gamificato. Nonostante il design innovativo, la piattaforma ha richiesto ambienti piuttosto complessi (es. HHVM) e non viene più mantenuta attivamente, pur restando un importante riferimento storico per l'evoluzione delle piattaforme CTF [21].
- **RootTheBox** è una piattaforma Python/Django orientata al Jeopardy-style ma arricchita da elementi di “game economy”. Oltre alle challenge tradizionali, integra meccanismi che consentono ai partecipanti di acquistare hint o strumenti utilizzando crediti virtuali guadagnati durante il gioco. Offre funzionalità avanzate di logging e di monitoraggio delle attività, risultando particolarmente adatta a contesti didattici più gamificati. Tuttavia, l'interfaccia e le numerose opzioni possono risultare complesse per gli organizzatori meno esperti [22].
- **Mellivora** è un progetto in PHP caratterizzato da leggerezza e semplicità, pensato anch'esso per eventi Jeopardy-style. Permette la gestione degli utenti, la pubblicazione di challenge e la visualizzazione di una scoreboard

essenziale. Non offre però supporto nativo per modalità Attack/Defense né meccanismi avanzati di game design. È spesso scelta per eventi di dimensioni piccole o medie, dove semplicità e stabilità rappresentano requisiti prioritari [23].

Oltre alle piattaforme gestionali, esistono anche repository e ambienti di allenamento che mettono a disposizione challenge preconfigurate. Esempi noti sono **VulnHub** [24], che distribuisce immagini VM vulnerabili scaricabili e **Hack The Box** [25] e **TryHackMe** [26], piattaforme commerciali molto diffuse che offrono laboratori virtualizzati accessibili via VPN, con scenari realistici orientati sia alla formazione individuale sia alla preparazione di competizioni. Tuttavia, molte di queste piattaforme, pur essendo ottime per la pratica individuale, non offrono strumenti nativi per la containerizzazione personalizzata o per l'integrazione diretta di challenge proprie all'interno di ambienti didattici privati.

Nonostante la varietà e il numero crescente di piattaforme e strumenti dedicati alle competizioni Capture The Flag (CTF), lo stato dell'arte presenta ancora diversi limiti e criticità che rendono complesse sia la progettazione sia l'integrazione didattica delle challenge.

Un primo limite riguarda la mancanza di standardizzazione nei processi di sviluppo e distribuzione delle challenge. Le piattaforme esistenti utilizzano formati eterogenei per la definizione delle challenge, i metodi di scoring e l'organizzazione delle informazioni. Questo comporta notevoli difficoltà nel riutilizzo o nella migrazione delle challenge da una piattaforma all'altra, generando una significativa duplicazione degli sforzi da parte di sviluppatori e docenti che intendono condividere o integrare contenuti esistenti [27, 28].

Sul piano tecnico, la creazione di ambienti vulnerabili tramite containerizzazione rappresenta certamente un progresso rispetto alle macchine virtuali tradizionali, grazie alla maggiore leggerezza e velocità di deploy [29].

Un ulteriore limite è costituito dalla carenza di materiale didattico riutilizzabile e documentazione approfondita. Molti progetti open-source o repository di challenge offrono codice funzionante, ma raramente accompagnato da spiegazioni dettagliate sui principi di sicurezza coinvolti, sulle vulnerabilità implementate o sulle scelte architettureali. Ciò rende più impegnativo per docenti e formatori adattare tali risorse a contesti educativi specifici o estrarne valore didattico.

Inoltre si riscontra una diffusa difficoltà nell'integrazione di challenge personalizzate all'interno delle piattaforme esistenti, sia gestionali sia di training. Sebbene piattaforme come CTFd o RootTheBox siano estensibili, la creazione di challenge nuove richiede spesso competenze avanzate non solo sul piano tecnico, ma anche nella conoscenza delle specifiche implementative della piattaforma. Analogamente, ambienti di allenamento come Hack The Box o TryHackMe, pur eccellenti per l'apprendimento individuale, risultano spesso ambienti chiusi e non offrono strumenti per integrare facilmente challenge sviluppate in proprio [25, 26].

Sulla base dei limiti individuati, questo lavoro prova a dare un contributo, pur consapevole di non poter risolvere tutte le complessità del settore. L'obiettivo principale è descrivere in modo chiaro e pratico come realizzare challenge CTF utilizzando Docker, fornendo esempi concreti e spiegazioni comprensibili anche a chi non ha grande esperienza nella gestione di ambienti containerizzati. Si punta inoltre a rendere più semplice il deployment delle challenge, grazie a configurazioni già pronte o facilmente adattabili, per permetterne un utilizzo veloce anche a docenti o professionisti. Un altro intento è quello di favorire il riuso delle challenge in contesti diversi, tramite una documentazione che spieghi non solo l'aspetto tecnico, ma anche gli obiettivi didattici e i passaggi fondamentali per adattare ad altre piattaforme o corsi.

Inoltre si vuole proporre alcuni suggerimenti didattici per progettare challenge che siano utili all'apprendimento, senza la pretesa di definire regole rigide, ma nella speranza di offrire un punto di partenza per chi voglia integrare queste attività nella propria didattica. Il lavoro mira a essere un aiuto pratico per chi

voglia avvicinarsi alla creazione di challenge CTF containerizzate, con l'intento di ridurre la complessità e facilitare la condivisione di contenuti didattici.

Capitolo 3

Challenge realizzate

3.1 Infrastruttura comune e architettura condivisa

L'implementazione delle challenge CTF create in questo progetto segue un approccio architetturale unificato che garantisce coerenza nell'esperienza utente e semplicità di gestione tecnica. Tutte e tre le challenge condividono componenti fondamentali come il sistema di autenticazione, la gestione degli utenti e l'infrastruttura di containerizzazione, differenziandosi esclusivamente nelle vulnerabilità specifiche implementate. Questa scelta progettuale permette di concentrare l'attenzione educativa sui concetti di sicurezza target di ogni challenge, eliminando distrazioni legate a differenze implementative non rilevanti per gli obiettivi formativi. Un approccio simile è adottato, ad esempio, dalla Web Security Academy di PortSwigger, dove le esercitazioni mantengono un layout e un'applicazione comune, modificando solo la vulnerabilità specifica da analizzare [30].

3.1.1 JSON Web Token

Il sistema di autenticazione implementato mediante JSON Web Token, comunemente abbreviato in JWT, costituisce una soluzione ormai consolidata e standardizzata per la gestione sicura della comunicazione delle identità e delle informazioni tra due entità in ambito web. Definito nella RFC 7519,

il JWT rappresenta uno strumento compatto e indipendente, progettato per trasportare in modo sicuro dichiarazioni o asserzioni (claims) tra un emittente e un destinatario, il cui contenuto può includere dati relativi all'identità dell'utente, ai privilegi di accesso o ad altri aspetti necessari a supportare le logiche applicative [31]. L'uso del JWT si colloca perfettamente all'interno dell'architettura dei sistemi moderni orientati ai microservizi e alle applicazioni a pagina singola (SPA), in quanto consente di svincolare la gestione dello stato di sessione dal server, demandando al token stesso la responsabilità di contenere le informazioni di autenticazione e autorizzazione [32].



Figura 3.1: Struttura del token JWT. Fonte: *Components of JWTs Explained*[33].

Dal punto di vista tecnico, come anticipato dalla precedente Figura 3.1, il JWT è costituito da tre segmenti codificati in Base64URL e separati da punti, i quali racchiudono rispettivamente l'header, il payload e la firma digitale. L'header descrive l'algoritmo utilizzato per la firma, che nel caso specifico del sistema qui descritto è l'HS256, ovvero HMAC basato su SHA-256, un algoritmo simmetrico che impiega la medesima chiave segreta sia in fase di generazione sia in fase di verifica della firma [32]. L'impiego di HS256 implica la necessità di proteggere la chiave segreta con un livello di robustezza adeguato, motivo per cui nel sistema in esame viene adottata una chiave esadecimale casuale di 128 caratteri, corrispondente a 512 bit, così da mitigare il rischio di attacchi brute force o di tentativi di guessing della chiave stessa, in linea con le raccomandazioni di

sicurezza evidenziate dal National Institute of Standards and Technology [34].

Il payload del JWT incorpora claims, ossia coppie chiave-valore che possono includere informazioni standard, quali l'identificativo dell'utente (claim "sub"), il nome utente, il ruolo associato e gli eventuali riferimenti temporali come la data di emissione (iat) o la scadenza (exp), nonché eventuali claims specifici dell'applicazione. Tali informazioni consentono al server di effettuare controlli di autorizzazione basati su ruoli o privilegi granulari senza la necessità di interrogare il database a ogni richiesta, rendendo così il processo più efficiente e scalabile [32]. Tuttavia, nonostante il payload sia codificato, è essenziale sottolineare che esso non è cifrato, salvo esplicita applicazione di tecniche come JWE (JSON Web Encryption), e può dunque essere visualizzato da chiunque intercetti il token, ragione per la quale non dovrebbe mai contenere dati sensibili come password o informazioni strettamente personali [31].

La firma digitale, calcolata sul concatenamento dell'header e del payload, sta alla base della sicurezza del framework poiché garantisce l'integrità del token e permette al server di accertare che il contenuto non sia stato alterato. Nel contesto dell'applicazione descritta, la verifica del token avviene attraverso la funzione `jwt.verify`, la quale decodifica il token, ricostruisce la firma e la confronta con quella ricevuta, rigettando il token qualora vi siano discrepanze o in caso di scadenza. Tale meccanismo costituisce una difesa efficace contro attacchi noti come il token tampering, dove un attaccante tenta di modificare il payload per innalzare i propri privilegi, o contro vulnerabilità legate all'accettazione di token non firmati o firmati con algoritmi insicuri, come l'algoritmo "none", la cui pericolosità è stata documentata in diversi studi di sicurezza [32].

Nell'architettura complessiva del sistema, il JWT funge dunque da veicolo per la trasmissione di informazioni tra il client e il server, consentendo a quest'ultimo di riconoscere l'identità dell'utente e di applicare le corrette regole di autorizzazione in base ai privilegi dichiarati nel payload. Questo approccio si rivela particolarmente adatto in scenari in cui il server deve gestire numerosi utenti autenticati senza mantenere uno stato persistente lato server, il che

favorisce la scalabilità e la distribuzione orizzontale delle applicazioni moderne [32]. In definitiva, l'adozione del JWT nel sistema descritto risponde a precise esigenze di sicurezza, performance e aderenza agli standard internazionali, rappresentando una soluzione robusta e conforme alle migliori pratiche delineate sia dalla comunità accademica sia dagli enti regolatori in materia di sicurezza informatica [34, 31].

3.1.2 Gestione utenti e database hardcoded

Per garantire un ambiente controllato e completamente riproducibile, tutte le challenge utilizzano un sistema di gestione utenti completamente hardcoded che elimina la necessità di registrazione dinamica e database esterni. Questa scelta progettuale assicura che ogni esecuzione delle challenge presenti condizioni identiche, facilitando la valutazione standardizzata degli studenti e garantendo la riproducibilità degli exploit in contesti didattici diversificati.

Il sistema implementa una struttura dati che include tre utenti con identificativi non sequenziali per simulare condizioni realistiche di database di produzione. La scelta di utilizzare identificativi non consecutivi riflette scenari reali dove i sistemi di gestione degli utenti non seguono pattern prevedibili, richiedendo agli studenti di sviluppare tecniche di enumerazione e reconnaissance per identificare gli obiettivi delle challenge.

Ai partecipanti vengono fornite le credenziali dell'utente Paolo con username `paolo` e password `password1` (tranne nella terza challenge), corrispondente all'identificativo 2, che rappresenta il punto di partenza standard per tutte le challenge. Questo approccio elimina la necessità di processi di registrazione che potrebbero introdurre complessità non necessarie nell'ambiente di apprendimento.

3.1.3 Stack tecnologico

La selezione di uno stack tecnologico unico per tutte le challenge garantisce coerenza nell'esperienza di sviluppo e semplicità nelle operazioni di deployment, riducendo la complessità operativa e permettendo un focus completo sugli aspetti di sicurezza e apprendimento. La scelta delle tecnologie è stata influenzata dalla loro ampia diffusione nell'industria del software, come evidenziato nel report del 2024 di Stack Overflow, secondo cui JavaScript è utilizzato dal 62% dei professionisti del settore [35], dalla relativa facilità d'uso sia per chi sviluppa le challenge sia per chi le affronta, e dalla possibilità di realizzare scenari didattici realistici nell'ambito della sicurezza informatica.

Il backend di tutte le challenge utilizza Node.js come runtime JavaScript, scelto per la sua ampia diffusione in applicazioni web moderne [35]. Express.js fornisce il framework web minimalista [36] che permette un controllo granulare sull'implementazione delle vulnerabilità senza introdurre astrazioni che potrebbero nascondere meccanismi di sicurezza importanti per l'apprendimento.

Il frontend è implementato utilizzando tecnologie web standard come HTML5, CSS3 e JavaScript vanilla senza dipendenze esterne, garantendo massima compatibilità e semplicità di comprensione. Questa scelta elimina la complessità associata a framework frontend moderni, permettendo agli studenti di concentrarsi sui concetti di sicurezza piuttosto che su specifiche tecnologie di interfaccia utente.

La gestione delle operazioni crittografiche utilizza il modulo `crypto` nativo di Node.js per operazioni come hashing delle password e generazione di token, come descritto nella documentazione ufficiale [37]. , mentre la libreria `jsonwebtoken` fornisce un'implementazione robusta e ampiamente utilizzata per la gestione dei JSON Web Token, come descritto nella documentazione ufficiale [38].

3.1.4 Interfaccia utente e design pattern condivisi

Tutte le challenge condividono un layout coerente e uno stile visivo uniforme, progettati per simulare interfacce aziendali realistiche, aumentando l'immersione nello scenario e la credibilità dell'ambiente di testing. L'interfaccia utente implementa un tema corporate professionale che richiama l'identità visuale di istituzioni finanziarie, utilizzando una palette di colori rosso-nero e elementi di design moderni per creare un'esperienza utente coinvolgente e realistica.

Il sistema di interfaccia include componenti comuni, come il form di login, con gestione degli errori e feedback visivo, una dashboard per la visualizzazione dei dati personali post-autenticazione, e interfacce amministrative quando applicabili alle specifiche challenge. Il design implementa principi di responsive design per garantire compatibilità su dispositivi desktop e mobile, riflettendo le aspettative moderne degli utenti riguardo all'accessibilità delle applicazioni web.

La scelta di utilizzare CSS3 avanzato con gradients lineari, box shadows e hover effects crea un'esperienza utente sofisticata che migliora l'engagement degli studenti e la percezione di realismo dell'ambiente di testing. La tipografia gerarchica e l'*information architecture* sono utilizzate per guidare l'utente attraverso le funzionalità disponibili in modo intuitivo, riducendo la curva di apprendimento associata alla navigazione dell'interfaccia, in accordo con i principi di design dell'esperienza utente [39].

3.1.5 Containerizzazione e deployment

Ogni challenge è completamente containerizzata utilizzando Docker per garantire isolamento, portabilità e facilità di deployment in ambienti didattici diversificati. La strategia di containerizzazione applica pratiche di sicurezza raccomandate, come l'utilizzo di immagini base leggere, la creazione di utenti non privilegiati all'interno del container e la riduzione del numero di layer, in

accordo con le linee guida di Docker e OWASP [40, 41]. In particolare, viene utilizzata Alpine Linux come immagine base, poiché la sua dimensione ridotta e la presenza minima di pacchetti preinstallati contribuiscono a ridurre la superficie di attacco rispetto a immagini più grandi come Debian o Ubuntu [42].

La configurazione Docker implementa principi di sicurezza come l'esecuzione di processi con utenti non-root e la limitazione delle capacità del container per prevenire potenziali escalation di privilegi. Ogni challenge include un Dockerfile che automatizza il processo di build e configurazione, garantendo riproducibilità dell'ambiente indipendentemente dalla piattaforma di deployment.

Nel seguente Listing (Listing 3.1) è riportato il Dockerfile per tutte le challenge. Si utilizza l'immagine `node:18-alpine`, l'istruzione `WORKDIR /usr/src/app` definisce una cartella di lavoro dedicata, facilitando la leggibilità e la coerenza dei percorsi all'interno del container [43]. Il file copia il manifest delle dipendenze (`package*.json`), installa i moduli necessari tramite `npm install`, e infine trasferisce il resto del codice; l'istruzione `EXPOSE` dichiara la porta su cui gira il server, mentre il comando `CMD ["node", "server.js"]` ne avvia l'esecuzione. Per aumentare la sicurezza, il container viene eseguito con l'utente non-root `'appuser'`. Questa pratica, suggerita dalla documentazione Docker impedisce all'applicazione di operare con privilegi di root [40].

```
1 FROM node:18-alpine
2 WORKDIR /usr/src/app
3 COPY package*.json ./
4 RUN npm install
5 RUN addgroup -S appgroup && adduser -S appuser -G appgroup
6 COPY . .
7 USER appuser
8 EXPOSE <porta>
9 CMD ["node", "server.js"]
```

Listing 3.1: Struttura docker file

L'orchestrazione dell'intera infrastruttura CTF utilizza Docker Compose per

gestire simultaneamente tutte le challenge, permettendo deployment con un singolo comando e semplificando significativamente la gestione in ambienti didattici. Il file di composizione include configurazioni per network isolation, volume management e resource limiting per garantire un ambiente stabile e controllato per l'apprendimento.

3.2 Metodologia di sviluppo

L'implementazione delle challenge CTF per la formazione in cybersecurity richiede un approccio metodologico strutturato che bilanci efficacemente il realismo tecnico con l'accessibilità educativa. La progettazione di ambienti di apprendimento vulnerabili presenta sfide uniche: è necessario bilanciare il realismo delle vulnerabilità con la sicurezza dell'infrastruttura host, mantenere la giusta complessità per supportare l'apprendimento senza scoraggiare, aggiornare o versionare correttamente i software vulnerabili, e garantire il rispetto della normativa sui dati sensibili quando si simula la gestione di informazioni riservate [27, 44, 45].

La metodologia adottata per lo sviluppo delle challenge segue un framework didattico basato sui principi del learning by doing[46], che privilegia l'apprendimento attraverso l'esperienza pratica diretta rispetto alla sola acquisizione teorica. Questo approccio si rivela particolarmente efficace nel dominio della cybersecurity, dove la comprensione profonda delle vulnerabilità e delle tecniche di exploit richiede necessariamente un'esperienza hands-on che permetta agli studenti di sperimentare direttamente i meccanismi di attacco e difesa.

La progressione della difficoltà tra le diverse challenge è stata progettata seguendo una curva di apprendimento graduale che introduce concetti di complessità crescente, permettendo agli studenti di costruire competenze solide prima di affrontare vulnerabilità multi-stage più sofisticate. La prima challenge (IDOR con WAF bypass) introduce i concetti fondamentali di autorizzazione inadeguata attraverso vulnerabilità IDOR combinate con tecniche di bypass elementari,

fornendo una base concettuale che viene poi estesa nelle challenge successive con l'introduzione di vulnerabilità crittografiche e tecniche di exploitation più avanzate.

Ogni challenge è progettata per essere autocontenuta dal punto di vista concettuale, ma al contempo costruisce competenze che facilitano la comprensione delle vulnerabilità successive. Questa struttura permette flessibilità nell'utilizzo didattico, consentendo ai docenti di selezionare challenge specifiche in base agli obiettivi formativi del corso e al livello di preparazione degli studenti.

3.2.1 Struttura del progetto e organizzazione

```

CTF-Challenges/
├── 01-idor-waf/
│   ├── server.js
│   ├── package.json
│   ├── Dockerfile
│   ├── public/
│   │   ├── index.html
│   │   ├── script.js
│   │   └── style.css
│   └──
├── 02-jwt-injection/
│   ├── server.js
│   ├── package.json
│   ├── Dockerfile
│   ├── public/
│   │   ├── index.html
│   │   ├── script.js
│   │   └── style.css
│   └──
├── 03-path-traversal/
│   ├── server.js
│   ├── package.json
│   ├── Dockerfile
│   ├── public/
│   │   ├── index.html
│   │   ├── script.js
│   │   ├── style.css
│   │   └── server.js.bak
│   └──
├── docker-compose.yml
├── deploy.sh
└── README.md
    
```

Figura 3.2: Struttura dei file delle challenge CTF

L'organizzazione del progetto, rappresentata nella Figura 3.2 segue un'architettura modulare che facilita la manutenzione, l'estensibilità e la comprensione della struttura da parte degli studenti. Ogni challenge è progettata come

un'applicazione completamente autocontenuta che include tutte le dipendenze, gli asset e le configurazioni necessarie per il funzionamento indipendente.

La struttura modulare presenta vantaggi significativi per la gestione didattica, permettendo ai docenti di utilizzare selettivamente challenge specifiche in base agli obiettivi del corso. Come mostrato nella precedente Figura 3.2, ogni directory di ciascuna challenge contiene un server Node.js completo con le proprie dipendenze definite nel file `package.json`, garantendo isolamento delle varie challenge e prevenendo conflitti di dipendenze che potrebbero compromettere la stabilità dell'ambiente.

Il file `docker-compose.yml` nella directory principale consente di orchestrare il deployment simultaneo di tutte le challenge in modo che un singolo comando (`docker compose up`) avvii automaticamente tutti i container necessari secondo le configurazioni previste [47]. Se si desidera far partire una sola challenge, è possibile utilizzare un file aggiuntivo (es. `docker-compose.single.yml`) che include solo il servizio specifico. Lo script `deploy.sh` automatizza completamente il processo di deployment, permettendo configurazione dell'intera infrastruttura CTF con un singolo comando.

La directory `public` di ogni challenge contiene tutti gli asset frontend necessari, inclusi file HTML per l'interfaccia utente, script JavaScript per la logica client-side e fogli di stile CSS per la presentazione. Questa organizzazione facilita la comprensione della struttura da parte degli studenti e permette modifiche rapide per personalizzazioni didattiche specifiche.

3.3 Challenge 1: IDOR con WAF bypass - Processo di sviluppo

3.3.1 Concept design e definizione degli obiettivi educativi

La prima challenge del percorso formativo è stata progettata per introdurre i partecipanti ai concetti fondamentali dell'Insecure Direct Object Reference (IDOR)[48] combinati con le tecniche di bypass dei Web Application Firewall. Gli obiettivi educativi primari includono la comprensione dei meccanismi di autorizzazione inadeguati nelle applicazioni web e l'apprendimento di metodologie di evasione per aggirare i controlli di sicurezza superficiali.

Lo scenario aziendale scelto simula BankSecure Corp., un sistema bancario per l'accesso ai dati personali e finanziari dei clienti. Questa scelta riflette situazioni reali, come dimostrato dal caso Capital One, in cui una "configurazione errata del WAF" ha permesso un attacco SSRF ¹ che ha bypassato le protezioni e ha portato all'esfiltrazione di credenziali e dati sensibili [50, 51]. In ambienti bancari dove sistemi legacy sono protetti da WAF configurati in modo incompleto o impreciso, vulnerabilità di autorizzazione possono persistere proprio a causa di tali lacune nella sicurezza del filtro web.

Il contesto bancario fornisce una giustificazione credibile per la presenza di dati altamente sensibili come carte di credito, codici fiscali e IBAN, rendendo critica la necessità di controlli di accesso rigorosi.

Architettura del sistema e gestione Utenti Il sistema non prevede registrazione dinamica degli utenti - tutte le credenziali e i profili sono hardcoded nel database applicativo per garantire un ambiente controllato e riproducibile. Ai partecipanti vengono fornite le credenziali di un profilo utente standard

¹Vulnerabilità che permette a un attaccante di far sì che il server effettui richieste HTTP verso risorse non previste (es. servizi interni, localhost), bypassando controlli di accesso e potenzialmente estraendo dati sensibili[49]

predefinito (`paolo/password1` con `ID=2`) che rappresenta il punto di partenza per l'esplorazione della vulnerabilità.

L'architettura della challenge prevede esattamente tre utenti con ID non sequenziali per simulare un ambiente bancario realistico:

- **ID=3**: Utente standard (non accessibile direttamente nella challenge)
- **ID=2**: Paolo - Account fornito ai partecipanti, privilegi standard come `ID=3`
- **ID=37**: Admin - Obiettivo finale contenente la flag, privilegi amministrativi

Questa configurazione deliberatamente non sequenziale simula uno scenario con enumerazione progressiva, voluta per dare un senso di soddisfazione allo studente per essere riuscito ad accedere ai dati di un altro utente creando anche una propedeuticità alla risoluzione della challenge. Vi è anche lo scenario con enumerazione non progressiva, richiedendo ai partecipanti tecniche di enumerazione attiva per scoprire l'ID admin contenente la flag.

Nell'endpoint `/api/users/{userId}/data` è stata implementata una vulnerabilità IDOR classica, dove il parametro `userId` consente l'accesso diretto ai dati di qualsiasi utente senza verifiche di autorizzazione appropriate. Il sistema implementa un'autenticazione JWT corretta che utilizza `jwt.verify` per la validazione dei token, ma manca il controllo di autorizzazione che dovrebbe verificare se l'utente autenticato ha il diritto di accedere ai dati richiesti.

Questa separazione didattica tra autenticazione (implementata correttamente) e autorizzazione (deliberatamente omessa) consente di focalizzare l'apprendimento su un singolo concetto di sicurezza senza distrazioni.

3.3.2 Implementazione del sistema di autenticazione

Il sistema di autenticazione utilizza JSON Web Token con algoritmo HS256 e una chiave segreta robusta di 128 caratteri esadecimali per garantire un meccanismo sicuro. L'implementazione utilizza `jwt.verify` invece di `jwt.decode`, come mostrato nel Listing 3.2, per assicurare la validazione crittografica del token, questo garantisce che la firma sia corretta, proteggendo l'applicazione dagli attacchi volti a manomettere il token al fine di alterarne il payload, come evidenziato nelle best practice di OWASP e Auth0 [52, 53]. In particolare, `'jwt.verify'` esegue sia la verifica della firma sia il controllo della validità temporale, impedendo l'accettazione di token non autenticati.

```
1  const SECRET_KEY = '*****';
2  function authenticate(req, res, next) {
3      const token = req.headers['authorization'];
4      if (token) {
5          try {
6              const decoded = jwt.verify(token, SECRET_KEY);
7              req.user = decoded;
8              next();
9          } catch (error) {
10             return res.status(401).json({ message: 'Token non
valido' });
11         }
12     } else {
13         res.status(401).json({ message: 'Token mancante' });
14     }
15 }
```

Listing 3.2: Implementazione del middleware di autenticazione

L'endpoint di login genera token JWT validi per tutti gli utenti con credenziali corrette, implementando una business logic che include la verifica di password e la generazione di token con payload appropriati contenenti ID utente, username e ruolo. Il token viene utilizzato correttamente per l'autenticazione ma deliberatamente non per l'autorizzazione negli endpoint IDOR, creando la condizione

necessaria per la vulnerabilità educativa.

3.3.3 Sviluppo del Web Application Firewall

Il WAF è implementato come funzione globale nel backend Express: registrata con `app.use(wafMiddleware)`, questa funzione intercetta tutte le richieste HTTP prima che raggiungano qualsiasi altro componente dell'applicazione. In questo modo agisce come un filtro a livello di applicazione, ispezionando ogni richiesta, potendo bloccarla (via status 403) o lasciarla passare al resto del server chiamando `next()` [54], come mostrato nel Listing 3.3.

```
1 function wafMiddleware(req, res, next) {
2   console.log('[WAF CHECK] ${req.method} ${req.originalUrl}')
3   ;
4   const url = req.originalUrl.toLowerCase();
5   const ua = (req.headers['user-agent'] || '').toLowerCase();
6
7   const blockedPatterns = [
8     /\api\/users\/37\/data/,
9     /\api\/users\/3\/data/
10  ];
11
12  if (ua.includes('burp') || ua.includes('sqlmap')) {
13    return res.status(403).json({ message: "WAF: Tool di
14    scanning rilevato" });
15  }
16  for (const pattern of blockedPatterns) {
17    if (pattern.test(url)) {
18      return res.status(403).json({ message: "WAF:
19      Accesso negato - Pattern sospetto" });
20    }
21  }
22  next();
23 }
```

Listing 3.3: Middleware WAF in Express

Come riportato dal Listing 3.3 sopra, ogni richiesta, sia GET che POST, a qualunque endpoint, viene prima controllata da questa funzione. Se corrisponde a un pattern bloccato o a User-Agent in blacklist, viene immediatamente respinta. Solo passando questi controlli la richiesta prosegue.

La vulnerabilità intenzionale è implementata attraverso la seguente modalità: mancando il decoding URL prima del pattern match, un attaccante può bypassare il WAF usando URL-encoded ("%33%37" per "37"), esempio classico di evasion WAF noto per molte implementazioni [55, 56].

3.3.4 Struttura dati e posizionamento della flag

La struttura dati degli utenti riflette un database bancario con informazioni finanziarie sensibili che giustificano rigorosamente la presenza di controlli di accesso. Ogni utente contiene credenziali, dati personali e informazioni bancarie organizzate gerarchicamente per simulare sistemi di customer relationship management del settore finanziario.

```
1 let users = [  
2 {  
3   id: 37,  
4   username: 'Sergio',  
5   password: '*****',  
6   role: 'admin',  
7   personalData: {  
8     creditCard: "4539-7894-5698-1234",  
9     codiceFiscale: "RSSMRA80A01H501U",  
10    iban: "IT60X0542811101000000123456",  
11    flag: "FLAG{e108c267a8...}"  
12  },  
13 },  
14 {  
15   id: 2,  
16   username: 'paolo',  
17   password: 'password1',
```

```
18 | role: 'standard',
19 | personalData: {
20 |   creditCard: "4532-1234-5678-9012",
21 |   codiceFiscale: "VRDLGU85B15H501V",
22 |   iban: "IT60X0542811101000000789012"
23 | }
24 | },
25 | {
26 |   id: 3,
27 |   username: 'Giampaolo',
28 |   password: '*****',
29 |   role: 'standard',
30 |   personalData: {
31 |     creditCard: "4532-7891-2345-6789",
32 |     codiceFiscale: "BRNGNN82C14H501W",
33 |     iban: "IT60X0542811101000000456789",
34 |     hint: "se solo questo fosse l'account giusto!"
35 |   }
36 | }
37 | ];
```

Listing 3.4: Struttura dati degli utenti del sistema bancario

Come visto nel precedente Listing 3.4, la flag è strategicamente posizionata nel campo `personalData.flag` dell'utente con ID 37 (Sergio), integrata naturalmente tra informazioni finanziarie realistiche. Questa collocazione evita posizionamenti artificiosi e mantiene la credibilità dello scenario bancario. La flag utilizza formato standard con prefisso `FLAG{}` con un hash di 128 caratteri che garantisce unicità e difficoltà di predizione.

L'account target (Sergio) possiede privilegi amministrativi completi, rappresentando un caso di studio interessante per i partecipanti che devono superare le protezioni WAF per accedere a un account con elevati privilegi di sistema. L'utente Giampaolo include un campo `hint` con il messaggio "se solo questo fosse l'account giusto!" che fornisce feedback diretto ai partecipanti, guidandoli a comprendere che hanno raggiunto un utente protetto ma non quello contenente

la flag ricercata. Questo meccanismo di feedback è essenziale nelle challenge CTF per evitare frustrazione e guidare il processo di apprendimento [57].

3.3.5 Implementazione degli endpoint API

```
1 // Endpoint vulnerabile IDOR
2 app.get('/api/users/:userId/data', authenticate, (req, res) =>
3   {
4     const requestedId = parseInt(req.params.userId);
5     const user = users.find(u => u.id === requestedId);
6     if (user) {
7       res.json({ personalData: user.personalData });
8     } else {
9       res.status(404).json({ message: 'Utente non trovato' });
10    }
11  });
12 // API amministrativa per lista utenti
13 app.get('/api/admin/users', authenticate, (req, res) => {
14   if (req.user.role === 'admin') {
15     const userList = users.map(u => ({
16       id: u.id,
17       username: u.username,
18       password: u.password,
19       role: u.role
20     }));
21     res.json({ users: userList });
22   } else {
23     res.status(403).json({ message: 'Accesso negato' });
24   }
25 });
```

Listing 3.5: Implementazione degli endpoint API principali

Il Listing 3.5. mostra che la struttura è progettata per essere intuitiva e semplice soprattutto per eventuali sviluppi ed evoluzioni future.

La vulnerabilità IDOR nell'endpoint `/api/users/:userId/data` si verifica a causa dell'assenza di controlli di autorizzazione adeguati: l'applicazione non verifica se l'utente autenticato ha effettivamente il diritto di accedere ai dati richiesti tramite il parametro `userId`. In tal modo, un utente può manipolare l'identificativo nella richiesta e visualizzare dati personali di altri utenti, come descritto nelle linee guida OWASP per prevenire l'accesso diretto insicuro agli oggetti [58].

3.3.6 Workflow di exploit IDOR

Il processo si articola in sei fasi principali: accesso alla challenge tramite CTFd, autenticazione, analisi delle richieste HTTP con Burp Suite [59], tentativo di enumerazione iniziale, bypass del WAF e enumerazione avanzata con estrazione della flag. Questo flusso riflette un attacco realistico contro un sistema bancario mal configurato, come lo scenario di BankSecure Corp., e promuove l'apprendimento pratico di tecniche di enumerazione e evasion.

Ovviamente la risoluzione della challenge può essere svolta con la metodologia prevista o meno e soprattutto con i tools che si preferisce.

Accesso alla challenge tramite CTFd

I partecipanti iniziano accedendo alla piattaforma CTFd come mostrato nella Figura 3.3, dove la challenge è descritta come un portale interno di BankSecure Corp. con l'obiettivo di recuperare una flag nascosta.

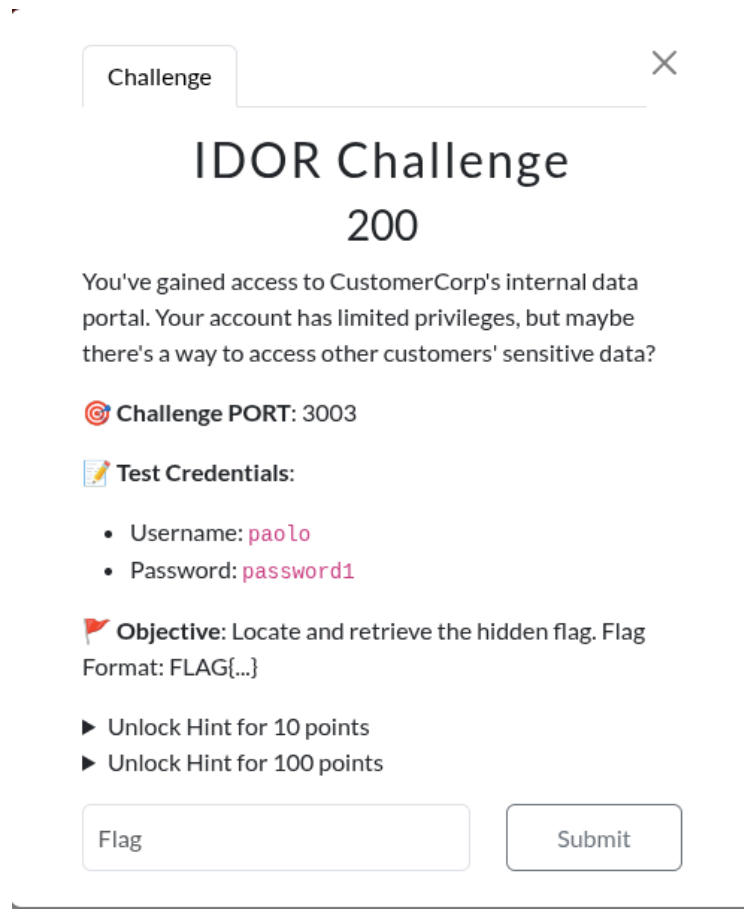


Figura 3.3: Schermata di CTFd per l'accesso alla challenge

Autenticazione

Gli studenti effettuano il login utilizzando le credenziali fornite: username `paolo` e password `password1`, corrispondenti all'utente con ID 2. L'autenticazione avviene tramite l'endpoint `/login`, che restituisce un token JSON Web Token (JWT) firmato con algoritmo HS256, contenente il payload `{id: 2, username: "paolo", role: "standard"}`, nella Figura 3.4 è possibile osservare il login e l'accesso ai dati dell'utente con ID 2, la risposta include il token JWT, da includere come Bearer `<token>` nell'intestazione Authorization delle richieste successive.

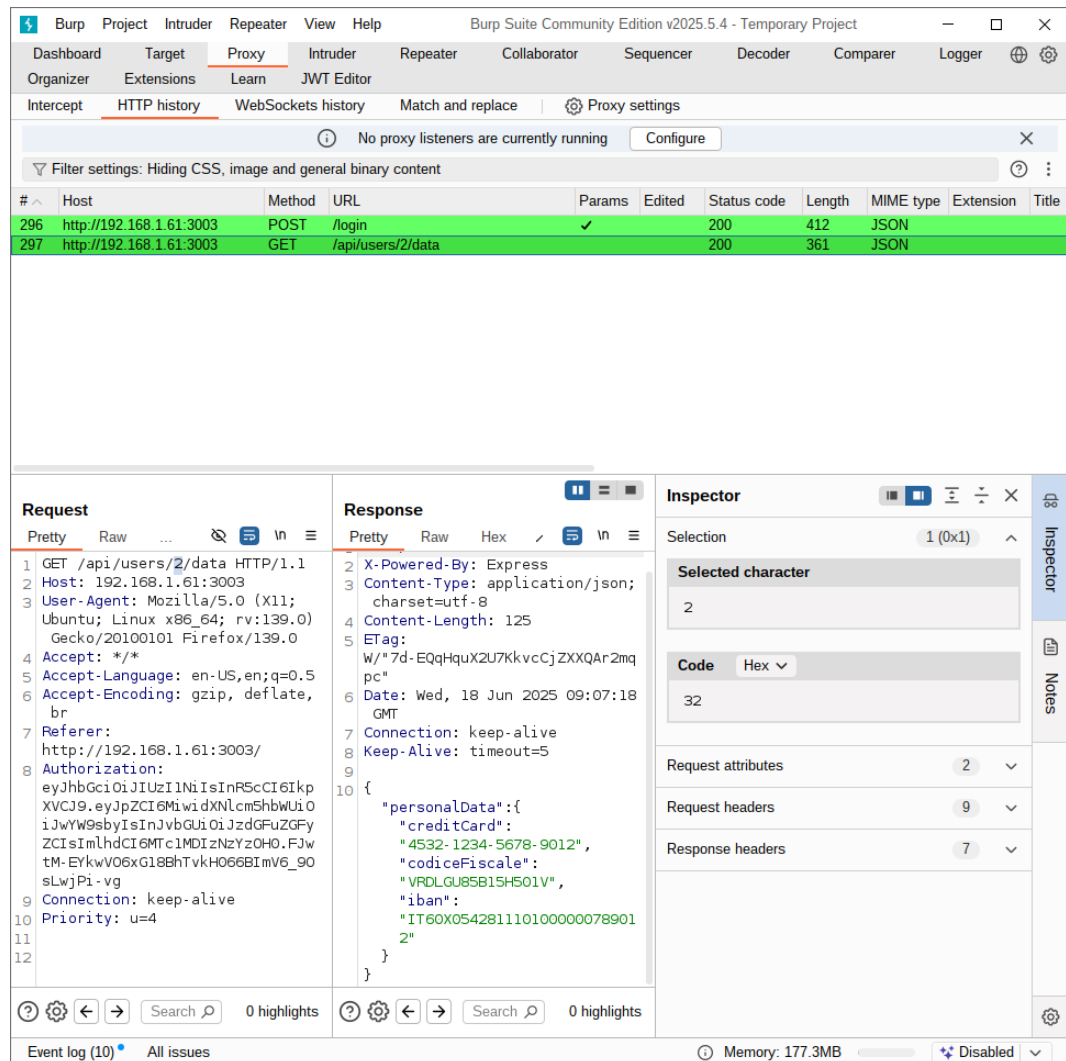


Figura 3.5: Richiesta GET per /api/users/2/data

Tentativo di enumerazione iniziale

Per testare l'IDOR, gli studenti modificano il parametro `userId` da 2 a 3, inviando una richiesta tramite Burp Suite, la richiesta viene bloccata dal WAF con una risposta 403 e il messaggio "WAF: Accesso negato - Pattern sospetto", indicando che l'ID 3 è protetto, visibile nella Figura 3.6. Tentativi simili con altri ID (es. 4, 5, ecc.) restituiscono risposte 404 ("Utente non trovato") o, in alcuni casi, 403, suggerendo che il WAF protegge specifici ID sensibili.

Riconoscendo il blocco del WAF, gli studenti ipotizzano che il filtro utilizzi

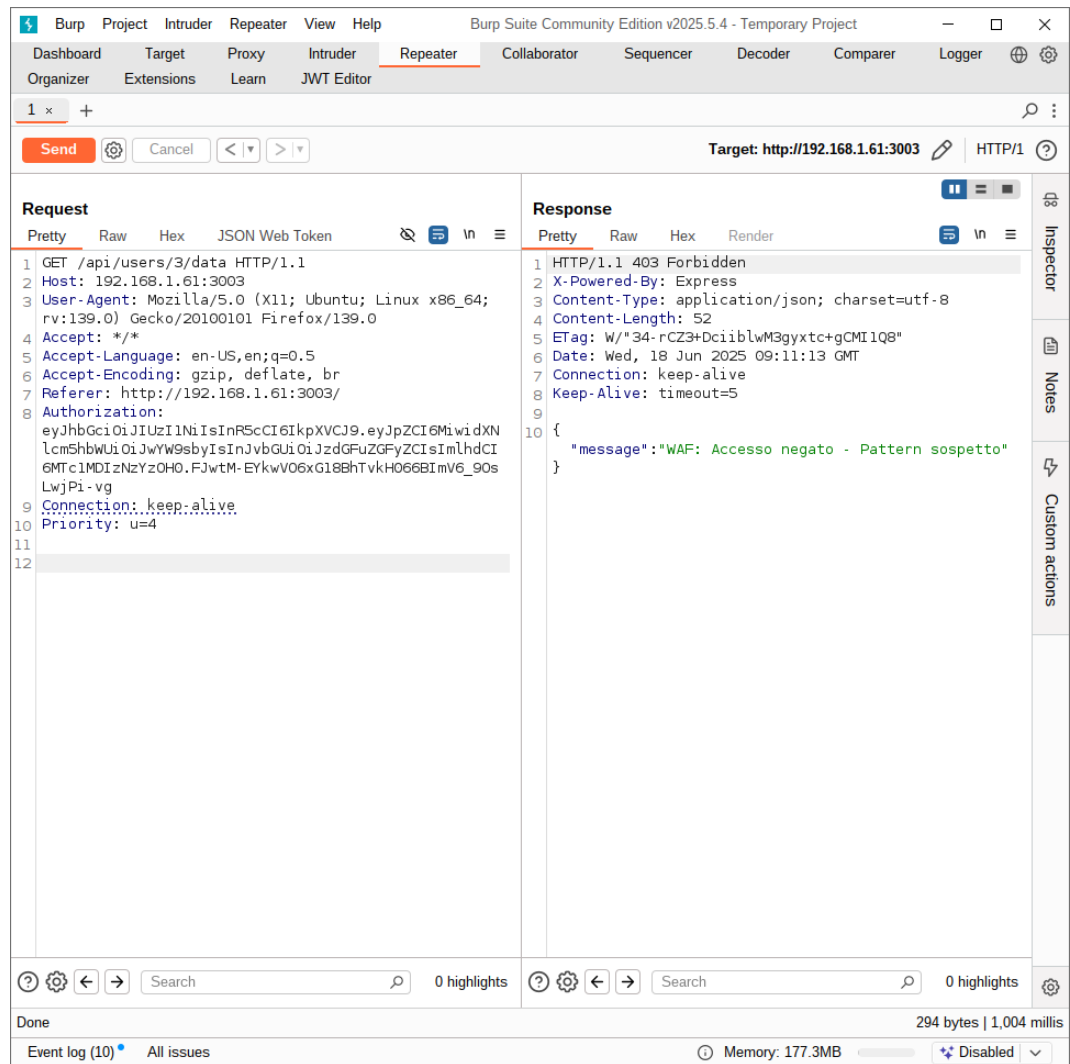


Figura 3.6: Tentativo di accesso a `/api/users/3/data` intercettato con Burp Suite, con risposta 403 dal WAF che indica un pattern sospetto.

pattern matching sull'URL senza gestire correttamente l'URL encoding. Provano a codificare l'ID 3, sostituendo il carattere '3' con `%33`, formando l'URL `/api/users/%33/data`. Come mostrato in Figura 3.7, la richiesta bypassa il WAF e restituisce i dati dell'utente Giampaolo (ID 3), incluso il campo `hint`: "se solo questo fosse l'account giusto!", confermando che l'encoding aggira le regole del WAF ma che ID 3 non contiene la flag, suggerendo che l'ID corretto è altrove.

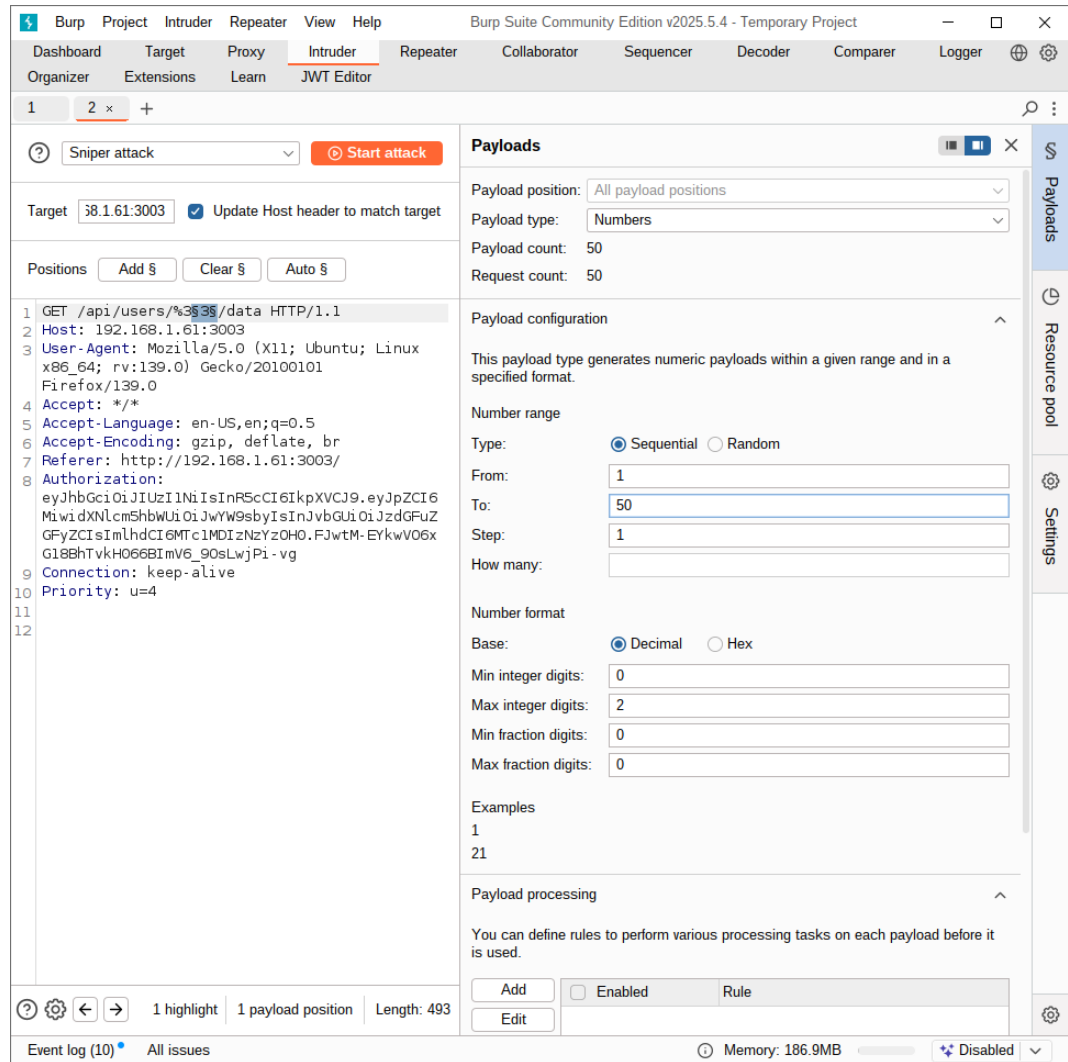


Figura 3.8: Configurazione Sniper attack

Le risposte vengono analizzate in base al codice di stato HTTP:

- **404:** Indica un ID inesistente (es. ID 4–36, 38–50).
- **200:** Indica un ID valido, come ID 2 (non protetto) o ID codificati come %33 (ID 3) e altri ID protetti.

Tra le risposte, quella per `/api/users/%337/data` restituisce una risposta 200, di lunghezza superiore agli altri id, suggerendo la presenza della flag, in Figura 3.9 si possono osservare i risultati dell'attacco Sniper nell'Intruder di Burp Suite, mostrando l'enumerazione degli ID da 0 a 50 e il recupero della flag per l'ID 37.

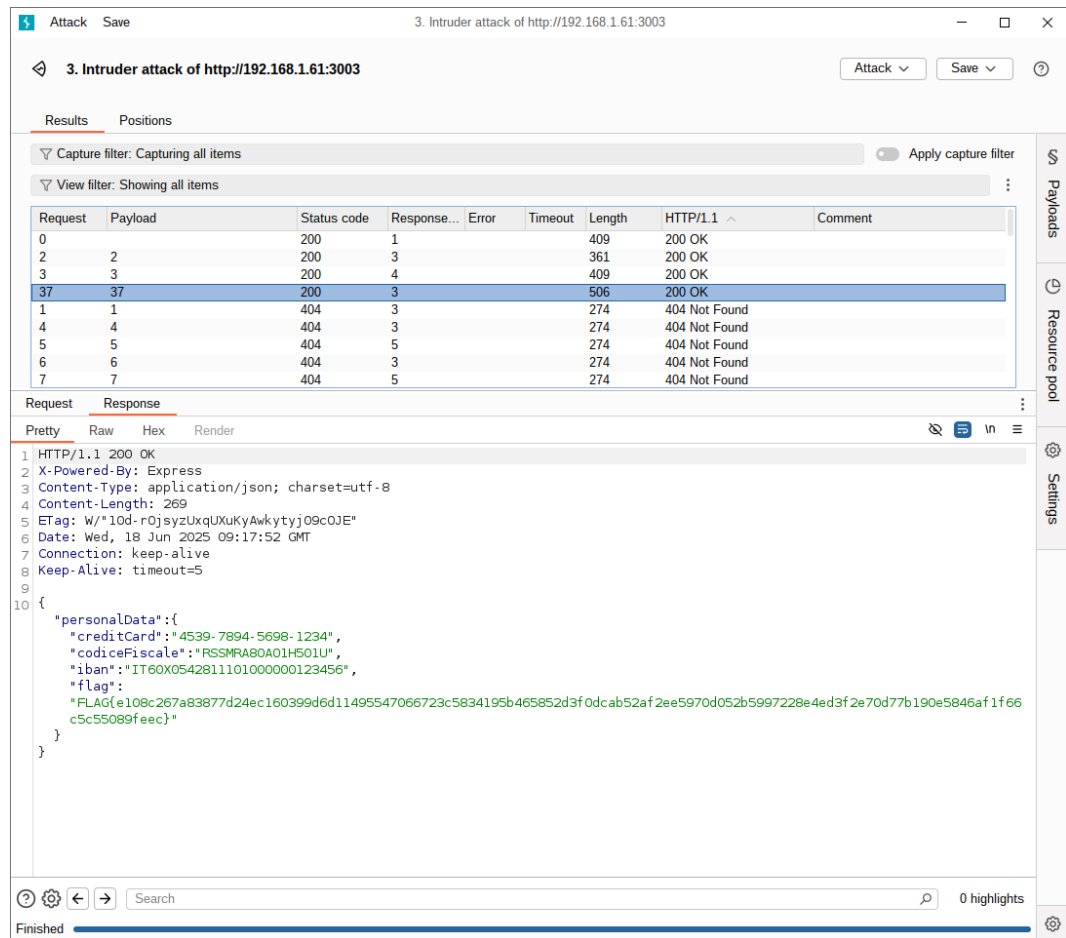


Figura 3.9: Esito Sniper attack

3.3.7 Mitigazioni e considerazioni educative

La Challenge 1 insegna agli studenti come sfruttare una vulnerabilità IDOR e bypassare un WAF, ma sottolinea anche l'importanza di implementare controlli di sicurezza adeguati. Le seguenti mitigazioni avrebbero prevenuto l'exploit:

- **Controlli di Autorizzazione:** L'endpoint `/api/users/:userId/data` dovrebbe verificare che `req.user.id` corrisponda a `userId` o che l'utente sia admin, impedendo l'accesso non autorizzato, come mostrato nel seguente Listing 3.6:

```

1   app.get('/api/users/:userId/data', authenticate, (req,
2     res) => {
3       const requestedId = parseInt(req.params.userId);
  
```

```
3         if (req.user.id !== requestedId && req.user.role
4         !== 'admin') {
5             return res.status(403).json({ message: 'Accesso
6             non autorizzato' });
7         }
8         const user = users.find(u => u.id === requestedId);
9         if (user) {
10            res.json({ personalData: user.personalData });
11        } else {
12            res.status(404).json({ message: 'Utente non
13            trovato' });
14        }
15    });
```

Listing 3.6: Endpoint IDOR sicuro

- **Normalizzazione URL nel WAF:** Decodificare gli URL prima del pattern matching con `decodeURIComponent` impedirebbe il bypass tramite codifiche come `%33%37`. Una configurazione robusta del WAF dovrebbe gestire tutte le varianti codificate degli input, in particolare, l'approccio basato su blacklist è quasi sempre aggirabile da un attaccante: bastano leggere modifiche alla forma della richiesta per eludere le regex o le parole chiave vietate. Numerosi casi studio lo confermano, evidenziando che questa metodologia è intrinsecamente insicura e pertanto sconsigliata rispetto a modelli di tipo whitelist [60, 61].
- **Validazione Input:** Limitare `userId` a valori numerici con una regex, o meglio, validare il formato atteso ridurrebbe il rischio di manipolazioni non previste.
- **Rate Limiting:** Implementare un meccanismo di rate limiting per limitare il numero di richieste per utente o indirizzo IP (es. 100 richieste al minuto per endpoint) ostacolerebbe attacchi di enumerazione massiva, come l'attacco Sniper con Burp Intruder. Soluzioni come Redis o middleware

come `express-rate-limit` [62] possono essere utilizzate per questo scopo, come nel seguente Listing 3.7:

```
1  const rateLimit = require('express-rate-limit');
2  app.use('/api/users/', rateLimit({
3    windowMs: 60 * 1000, // 1 minuto
4    max: 100, // massimo 100 richieste
5    message: 'Troppe richieste, riprova più tardi'
6  }));
7
```

Listing 3.7: Esempio di rate limiting con express

- **Uso di UUID:** Sostituire gli ID numerici sequenziali con identificatori universali (UUID), come `550e8400-e29b-41d4-a716-446655440000`, renderebbe l'enumerazione impraticabile, poiché gli UUID sono casuali e non prevedibili [63], questa mitigazione però non basta perchè gli UUID non devono essere usati come dei segreti, come riportato nel loro RFC[64]
- **WAF di Terze Parti Collaudati:** Adottare WAF di terze parti ben collaudati, come Cloudflare, AWS WAF o ModSecurity, configurati correttamente per proteggere da attacchi come IDOR e bypass tramite URL encoding. Una configurazione efficace include regole aggiornate per rilevare pattern sospetti, normalizzazione degli input e protezione contro tecniche di evasion. Ad esempio, un WAF dovrebbe essere configurato per decodificare gli URL e applicare filtri su tutti i parametri, evitando bypass come `%33%37`.
- **Offuscamento degli Errori:** Restituire messaggi di errore generici (es. "Richiesta non valida") invece di dettagli specifici (es. "Utente non trovato") impedirebbe agli attaccanti di dedurre l'esistenza di determinati ID, riducendo il rischio di enumerazione.
- **Protezione contro Fingerprinting del WAF:** Configurare il WAF per non rivelare informazioni sulla sua presenza o versione (es. rimuovendo

header come **X-Powered-By** o personalizzando i messaggi di errore) renderebbe più difficile per un attaccante identificare e sfruttare punti deboli specifici.

Dal punto di vista educativo, la challenge introduce la separazione tra autenticazione e autorizzazione, le limitazioni dei WAF basati su pattern e l'enumerazione in ambienti con ID non sequenziali. L'uso dell'URL encoding rafforza l'approccio pratico, mentre l'hint su ID 3 guida gli studenti senza compromettere la sfida. Le mitigazioni proposte, inclusi rate limiting, UUID e WAF di terze parti, riflettono best practice reali, preparando gli studenti a progettare sistemi sicuri e a comprendere le contromisure contro attacchi comuni.

3.4 Challenge 2: JWK Header Parameter Injection

3.4.1 Concept design e definizione degli obiettivi educativi

La seconda challenge introduce la vulnerabilità *JWK Header-Parameter Injection*² nei JSON Web Token, focalizzandosi sui meccanismi crittografici asimmetrici e sulla manipolazione di metadati crittografici incorporati. L'obiettivo formativo consiste nell'apprendimento di tecniche di exploitation che sfruttano la fiducia impropria verso chiavi pubbliche incluse nell'header del token, rappresentando un'evoluzione significativa rispetto alla challenge precedente basata su algoritmi HMAC.

Come la precedente challenge, il sistema implementa tre profili utente con segmentazione rigorosa dei privilegi, l'account Sergio (ID = 3) mantiene un ruolo amministrativo continuando a rappresentare l'obiettivo finale dell'exploitation. Paolo (ID = 2) rappresenta il punto di partenza per i partecipanti, configurato con privilegi utente standard e credenziali `paolo/password1`

²La vulnerabilità è descritta da PortSwigger come "JWT authentication bypass via jwk header" [65].

La vulnerabilità risiede in un middleware di autenticazione che utilizza JSON Web Token per verificare l'identità degli utenti. Il problema critico consiste nell'accettazione di JSON Web Keys incorporate direttamente nell'header del token. Quando il campo `jwk` è presente, il sistema ignora la propria chiave pubblica legittima e utilizza invece la chiave fornita nell'header per verificare la firma.

È importante notare che la RFC 7515 ammette esplicitamente il parametro di header `jwk` [66]. L'implementazione sicura richiede però che l'applicazione stabilisca criteri rigorosi di affidabilità (*whitelist* o pairing con `kid`) prima di accettare la chiave. Nel nostro caso didattico tali controlli sono assenti, violando il principio di separazione tra chiavi di sistema e chiavi potenzialmente controllate da entità esterne.

Un attaccante può quindi generare una propria coppia di chiavi RSA indipendente, creare un token JWT arbitrario con il payload desiderato e firmarlo con la propria chiave privata. Il passaggio cruciale consiste nell'includere la chiave pubblica nel campo `jwk` dell'header. Il middleware utilizzerà tale chiave per la verifica, convalidando con successo un token in realtà fraudolento.

L'exploitation consente una escalation di privilegi diretta modificando il payload e cambiando il campo `role` da `user` a `admin`, ottenendo accesso completo all'endpoint `/admin`.

Il sistema di autenticazione è basato su crittografia asimmetrica RSA (algoritmo RS256), sostituendo l'approccio HMAC della challenge precedente. Di seguito si mostra la generazione dinamica di una coppia di chiavi RSA da 2048 bit al bootstrap del server (Listing 3.8). La lunghezza di 2048 bit è ancora considerata sicura, ma il NIST, con SP 800-131A Rev. 2, ne prevede la dismissione per nuove implementazioni dopo il 31 dicembre 2030 [67].

```
1  const { publicKey, privateKey } = crypto.generateKeyPairSync('
    rsa', {
2    modulusLength: 2048,
3    publicKeyEncoding: {
```

```
4     type: 'spki',
5     format: 'pem'
6 },
7 privateKeyEncoding: {
8     type: 'pkcs8',
9     format: 'pem'
10 }
11 });
```

Listing 3.8: Generazione dinamica della coppia di chiavi RSA con parametri di sicurezza standard

La codifica SPKI per la chiave pubblica e PKCS#8 per la chiave privata garantisce interoperabilità con l'estensione JWT Editor di Burp Suite, strumento utilizzato per l'exploitation affrontata in seguito.

L'endpoint `/login` verifica credenziali `username/password` e genera JWT firmati con RS256 come mostrato nel Listing (3.9). Il payload utilizza claim standard secondo RFC 7519.

```
1 const payload = {
2   sub: user.username, // subject claim (standard)
3   role: user.role,
4   iat: Math.floor(Date.now() / 1000),
5   exp: Math.floor(Date.now() / 1000) + 3600
6 };
7
8 const token = jwt.sign(payload, privateKey, {
9   algorithm: 'RS256',
10  header: {
11    typ: 'JWT',
12    alg: 'RS256'
13  }
14 });
```

Listing 3.9: Costruzione del payload JWT con claim standardizzate RFC

7519

Il middleware vulnerabile è illustrato nel Listing 3.10. La libreria `jsonwebtoken` controlla che il valore `alg` nell'header corrisponda alla lista passata in opzione; l'attaccante mantiene quindi `RS256` per evitare *algorithm confusion*. La logica insicura, che preferisce la chiave incorporata nel token, quando presente, consente il bypass.

```
1 const header = JSON.parse(Buffer.from(token.split('.')[0], '
    base64').toString());
2 const payload = JSON.parse(Buffer.from(token.split('.')[1], '
    base64').toString());
3
4 // Vulnerabilità critica: uso di JWK embedded
5 if (header.jwk) {
6   if (header.jwk.kty === 'RSA') {
7     const embeddedPublicKey = jwkToPem(header.jwk); //
        conversione JWK -> PEM
8     decoded = jwt.verify(token, embeddedPublicKey, { algorithms
        : ['RS256'] });
9     jwkExploited = true;
10  }
11 } else {
12   decoded = jwt.verify(token, publicKey, { algorithms: ['RS256',
        ] });
13 }
```

Listing 3.10: Implementazione della vulnerabilità JWK Injection nel middleware di autenticazione

Approfondimento tecnico: Confronto tra HS256 e RS256 nei JSON Web Token La sicurezza dei JSON Web Token (JWT) si fonda, tra gli altri aspetti, sulla capacità di garantire integrità e autenticità dei dati trasmessi. A tale scopo vengono comunemente impiegati algoritmi di firma crittografica, tra i quali **HS256** (HMAC-SHA256) e **RS256** (RSA Signature with SHA-256) costituiscono due delle opzioni più diffuse [66, 68].

L'algoritmo HS256 si basa sul costruito HMAC (Hash-based Message Authentication Code), che appartiene alla categoria delle firme *simmetriche*. In questo schema, la medesima chiave segreta K viene impiegata sia per generare la firma sia per verificarla. Il calcolo dell'HMAC avviene secondo la seguente costruzione matematica:

$$\text{HMAC}_H(M) = H((K \oplus \text{opad}) \parallel H((K \oplus \text{ipad}) \parallel M))$$

dove H rappresenta una funzione di hash crittografico, tipicamente SHA-256, mentre M indica il messaggio da autenticare — la concatenazione codificata in Base64Url dell'header e del payload — e **opad**/**ipad** sono costanti specifiche impiegate per garantire la separazione tra il processo interno ed esterno della funzione HMAC. L'uso delle costanti $\text{ipad} = 0x36$ e $\text{opad} = 0x5c$ è definito dallo standard FIPS 198-1 [69]. L'algoritmo HS256 è caratterizzato da un'elevata efficienza computazionale e da bassa latenza; tuttavia, la gestione della chiave segreta risulta critica, poiché la sua compromissione comporta la perdita della capacità di garantire sia l'integrità sia l'autenticità dei token. Inoltre, l'uso di una chiave condivisa limita la scalabilità in architetture distribuite, in quanto tutti i sistemi incaricati della verifica devono disporre della stessa chiave [70].

L'algoritmo RS256 si basa invece sulla *firma digitale asimmetrica* realizzata mediante il sistema crittografico RSA. In questo contesto viene utilizzata una coppia di chiavi, di cui una, detta **privata**, è impiegata esclusivamente dal server per la generazione della firma, mentre l'altra, detta **pubblica**, è distribuita ai soggetti che necessitano di effettuare la verifica [68, 70]. Il processo di firma digitale mediante RSA si articola nel calcolo dell'hash SHA-256 del messaggio M , seguito dalla cifratura di tale hash attraverso la chiave privata RSA, ottenendo così la firma digitale. La verifica viene effettuata dal destinatario, il quale utilizza la chiave pubblica RSA per decifrare la firma e confrontare il valore decifrato con l'hash calcolato sul messaggio ricevuto [66].

All'interno di un JWT firmato con RS256, l'header specifica "alg": "RS256".

In tale scenario, la chiave privata rimane strettamente custodita sul server, riducendo il rischio di compromissione, mentre la chiave pubblica può essere liberamente distribuita per consentire la verifica del token da parte di soggetti terzi. RS256 garantisce dunque una maggiore sicurezza in contesti enterprise e distribuiti, grazie alla netta separazione tra la fase di firma e quella di verifica [70]. Tuttavia, l'algoritmo presenta un costo computazionale superiore rispetto a HS256, dovuto alla complessità intrinseca delle operazioni RSA [68].

È fondamentale evidenziare che **RSA non utilizza HMAC** per la generazione della firma digitale: si tratta di due approcci crittografici concettualmente e tecnicamente distinti, pur perseguendo entrambi l'obiettivo di garantire integrità e autenticità [66, 68].

La scelta tra HS256 e RS256 deve pertanto basarsi sulle esigenze di sicurezza, sull'architettura del sistema e sulle politiche di gestione delle chiavi, tenendo conto dei differenti compromessi in termini di prestazioni e scalabilità.

3.4.2 Workflow dell'exploit

Accesso alla challenge e autenticazione

L'attaccante accede alla piattaforma BankSecure Corp con le stesse modalità viste nella challenge precedente ma con la porta 3007 come mostrato in Figura 3.10. Viene eseguito il login con le credenziali dell'utente standard `paolo/password1`, attraverso una richiesta POST verso l'endpoint `/login`. In risposta, il server restituisce un token JWT firmato utilizzando l'algoritmo RS256.

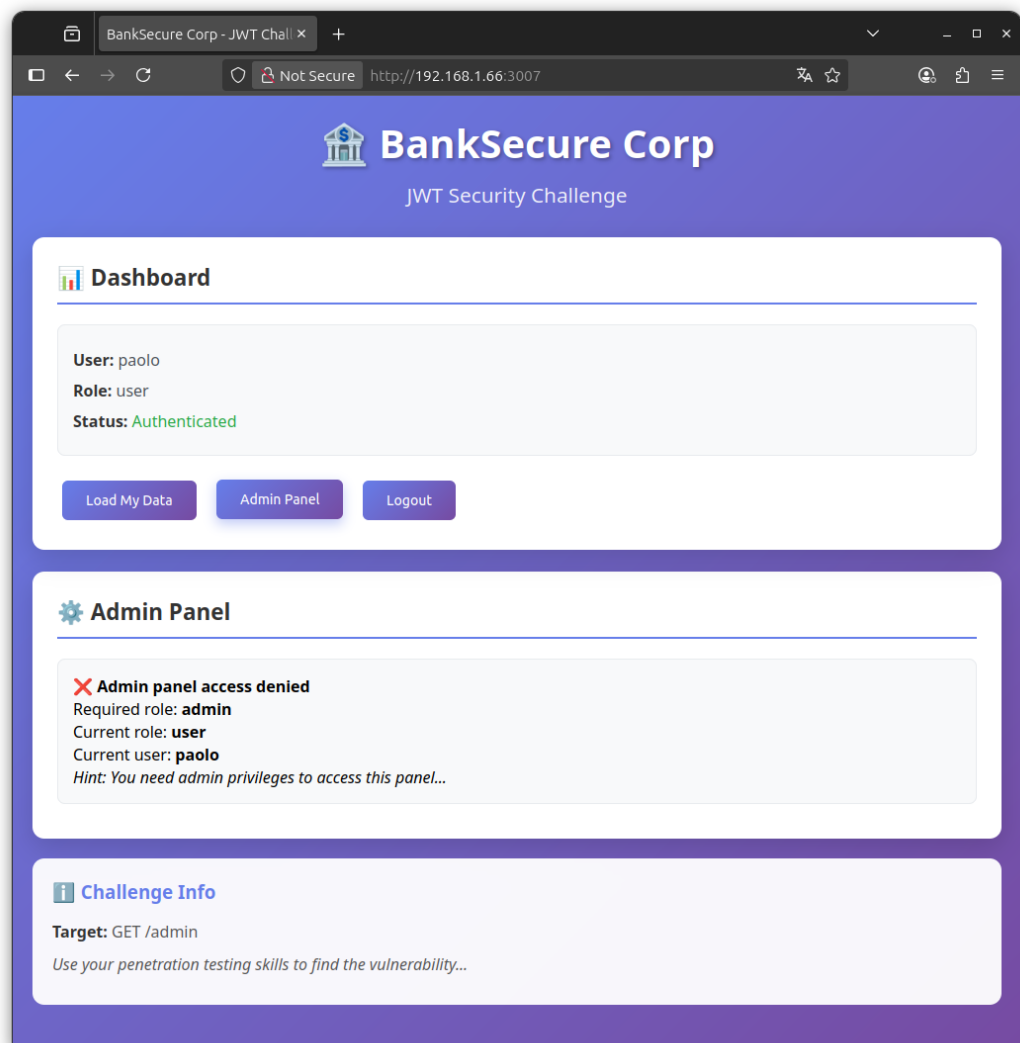


Figura 3.10: Negazione accesso al pannello di controllo

Verifica del ruolo utente

Si tenta di accedere all'endpoint `/admin` utilizzando il token JWT originale nell'header `Authorization`. Il server restituisce un errore `403 Forbidden`, come riportato dalla Figura 3.11, si ottiene anche la conferma che l'utente standard non possiede privilegi amministrativi e che li necessita per l'accesso al pannello di controllo.

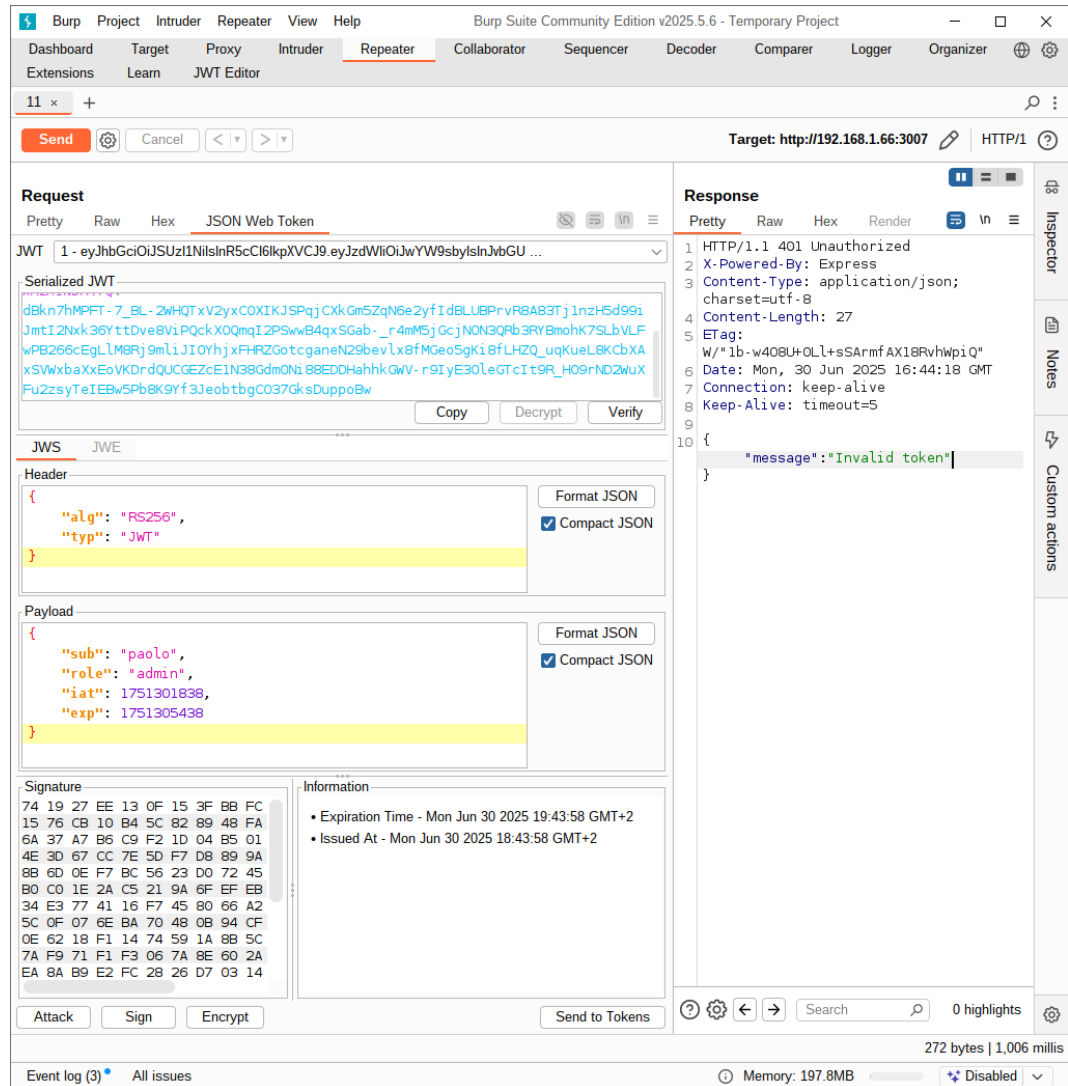


Figura 3.12: Tentativo di Token tampering fallito

Generazione di una nuova coppia di chiavi RSA

Tramite l'estensione JWT Editor, viene generata una nuova coppia di chiavi RSA (2048 bit), osservabile nella Figura 3.13. La chiave pubblica servirà ad essere inserita nel token sotto forma di JSON Web Key (JWK), mentre la chiave privata sarà utilizzata per firmare il token manipolato.

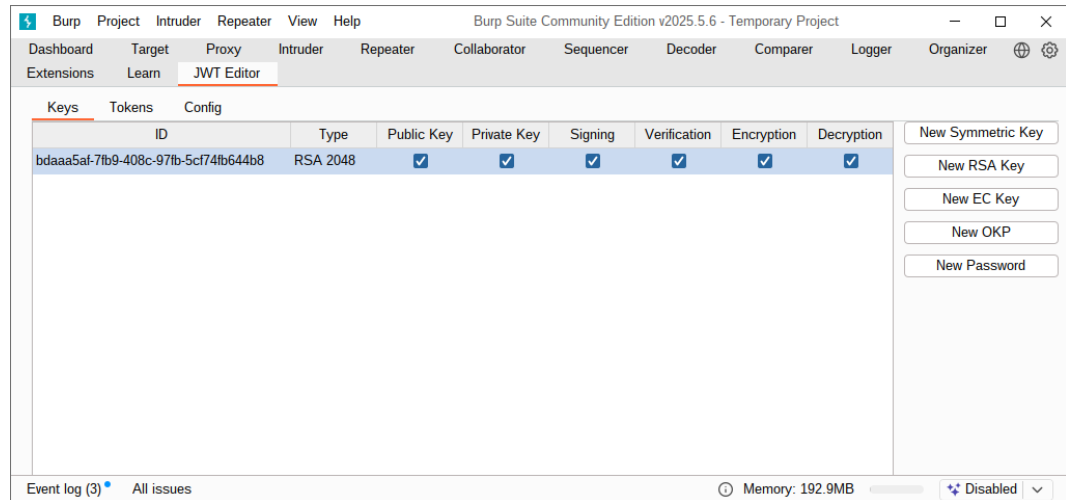


Figura 3.13: Generazione coppia di chiavi RSA

Embedding della chiave pubblica nel JWT

L'attaccante esegue la funzionalità **Embed JWK Attack** nel JWT Editor. Durante questa operazione:

- la chiave pubblica generata viene convertita in formato JWK e inserita nell'header del token sotto il campo `jwk`;
- Burp Suite rigenera automaticamente la firma digitale del token utilizzando la chiave privata corrispondente (il passaggio è mostrato nella Figura 3.14).

Il token JWT manipolato e correttamente firmato viene inserito nell'header **Authorization** della richiesta GET verso l'endpoint `/admin`.

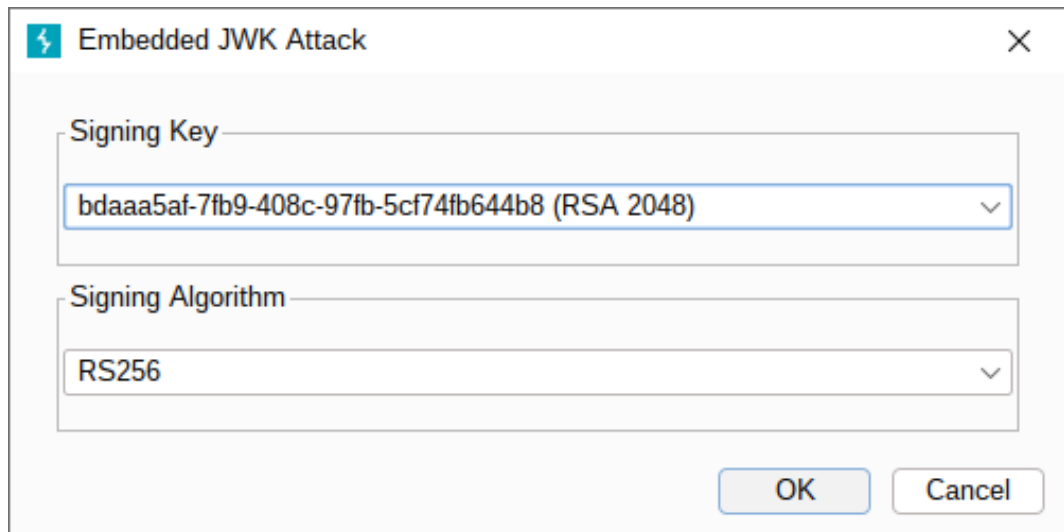


Figura 3.14: Firma del token con chiave RS256

Accesso al pannello amministrativo e recupero della flag

Il middleware vulnerabile, rilevando la presenza del campo `jwk` nell'header, utilizza la chiave pubblica embedded per verificare la firma digitale. Poiché il token è stato firmato con la corrispondente chiave privata, la verifica ha esito positivo e l'attaccante ottiene accesso al pannello amministrativo. Il server restituisce una risposta 200 OK contenente il messaggio di benvenuto e la flag, come mostrato dalla Figura 3.15:

Questo risultato conferma il successo dell'exploit, dimostrando come la fiducia impropria verso chiavi pubbliche inserite nel token consenta l'escalation di privilegi e l'accesso a funzionalità riservate agli amministratori.

3.4.3 Mitigazioni e considerazioni educative

La vulnerabilità dimostrata nella seconda challenge è strettamente legata alla gestione impropria del parametro `jwk` nell'header dei JSON Web Token (JWT). Per prevenirla, la letteratura e le linee guida di settore raccomandano le seguenti contromisure:

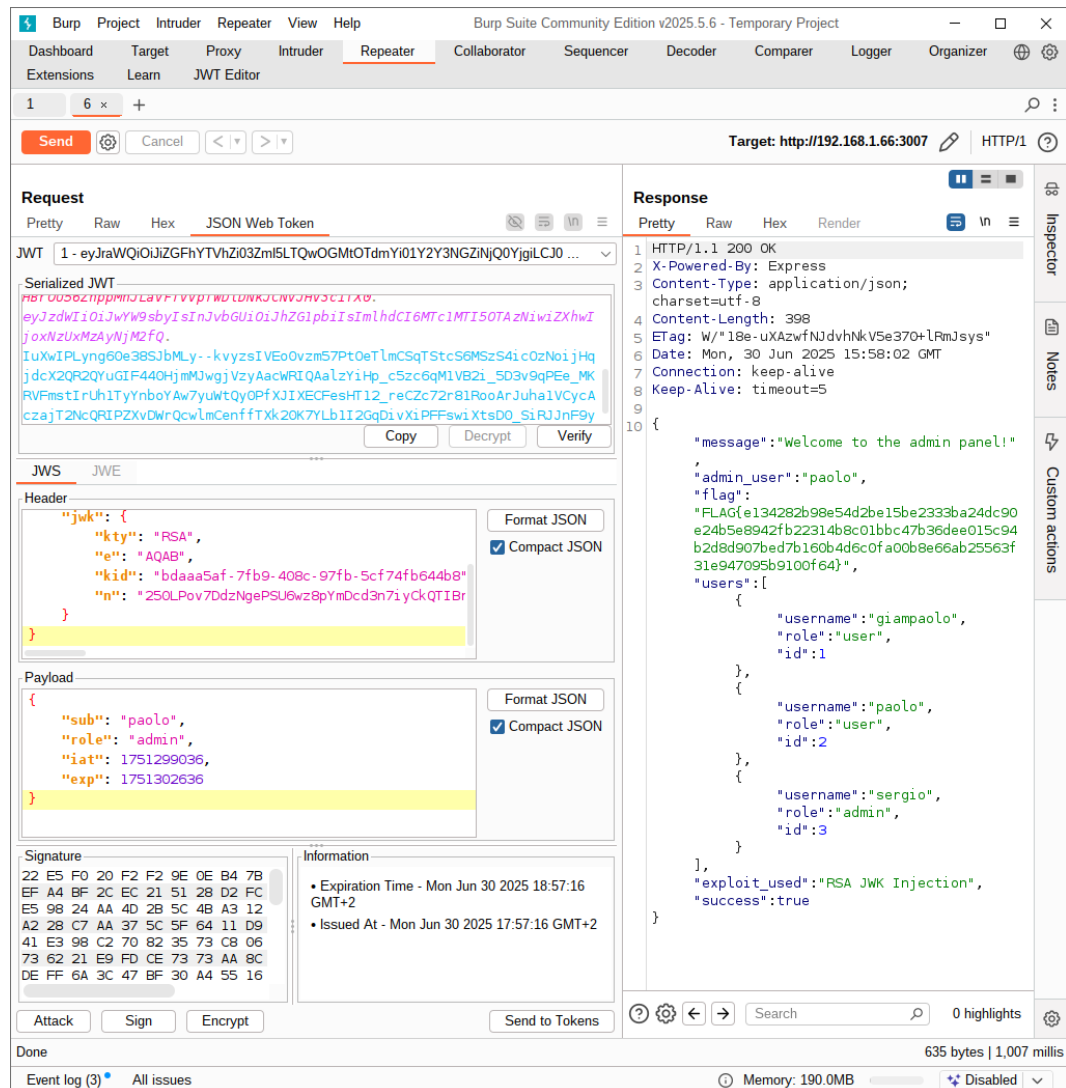


Figura 3.15: Tentativo di token tampering con esito positivo

- **Non accettare chiavi pubbliche fornite dal client** o parametri JWK embedded se non strettamente necessari. L'applicazione dovrebbe utilizzare solo chiavi generate o gestite in modo sicuro sul server oppure provenienti da key store affidabili e sotto il controllo dell'organizzazione [71, 72].
- **Implementare una whitelist degli algoritmi e degli identificatori di chiave (kid)** ammessi. Qualsiasi valore inatteso deve determinare il rifiuto del token, riducendo il rischio di exploit tramite algoritmi deboli o chiavi arbitrarie [71, 73].

Dal punto di vista didattico, questa challenge rappresenta un caso concreto in cui l'uso di tecnologie crittografiche avanzate, se mal configurato o troppo permissivo, può diventare a sua volta un vettore di attacco. È fondamentale che gli studenti comprendano non solo il funzionamento dei JWT, ma anche le implicazioni di sicurezza legate alla gestione dinamica delle chiavi.

3.5 Challenge 3: Information Disclosure e Weak Cryptography

3.5.1 Concept design, superficie d'attacco e vulnerabilità di backup

La terza challenge è progettata per dimostrare come due errori di sicurezza, la pubblicazione involontaria di file di backup e l'uso di algoritmi crittografici deboli, possano convergere in un singolo punto di compromissione con conseguenze potenzialmente gravi.

```
CTF-Challenges/  
├── 03-path-traversal/  
│   ├── server.js  
│   ├── package.json  
│   ├── Dockerfile  
│   └── public/  
│       ├── index.html  
│       ├── script.js  
│       ├── style.css  
│       └── server.js.bak  
├── docker-compose.yml  
├── deploy.sh  
└── README.md
```

Figura 3.16: Struttura della terza challenge

Come visibile in 3.16, all'interno della directory `/public`, viene lasciata per errore una copia di `server.js` con estensione `.bak`. Si tratta di uno scenario verosimile, difatti OWASP segnala la pubblicazione accidentale di file di backup fra le vulnerabilità di configurazione più diffuse [74].

A differenza delle challenge precedenti, questa volta l'utente non dispone di alcuna credenziale iniziale, e la superficie d'attacco percepita sembra limitata alle sole risorse statiche del server. Tuttavia, strumenti di content discovery,

come scanner per directory e file nascosti o enumeratori di percorsi web, possono essere utilizzati per individuare il file di backup e accedere a informazioni che dovrebbero rimanere confinate nel back-end. Ciò mostra come la sicurezza “a cipolla”, basata su strati protettivi progressivi, possa crollare completamente se lo strato più esterno, il web server, espone risorse non destinate al pubblico.

Analizzando il file `.bak`, l'attaccante scopre che i nomi utente e le password sono memorizzati sotto forma di hash generati con l'algoritmo Message Digest 5 (MD5). Nonostante siano passati oltre vent'anni dalle prime evidenze di debolezza, MD5 è ancora presente in alcuni contesti non educativi. Già a partire dal 2004, degli studi hanno evidenziato come MD5 sia vulnerabile alla creazione di collisioni, cioè alla possibilità di generare due input diversi che producano lo stesso hash [75]. Negli anni successivi, queste tecniche sono state ulteriormente perfezionate, rendendo sempre più facile sfruttare tali debolezze [76]. Anche la Internet Engineering Task Force (IETF), nel 2011, ha dichiarato MD5 inadeguato per ulteriori utilizzi in ambito di sicurezza [77]. Inoltre, il Computer Emergency Response Team (CERT) ha segnalato come queste vulnerabilità possano essere sfruttate per creare firme digitali fraudolente [78].

In ambito password hashing, MD5 senza salt, è estremamente vulnerabile agli attacchi, come dimostrato dal lavoro di Oechslin [79]. Questo significa che gli hash presenti nel file di backup possono essere facilmente crackati, rivelando password in chiaro. In questo scenario, dunque, il file lasciato sul server non espone solo il codice sorgente dell'applicazione, ma fornisce anche all'attaccante un vero e proprio dizionario di credenziali, compromettendo completamente il sistema.

Questa challenge spinge lo studente a sperimentare in prima persona quanto possa essere fragile un'architettura che, pur implementando tecniche moderne come JWT e controlli di ruolo, resta vulnerabile a causa di artefatti lasciati nel file system o di scelte crittografiche obsolete. L'obiettivo didattico è far comprendere che la sicurezza non riguarda soltanto la scrittura del codice, ma coinvolge l'intero ciclo di vita dello sviluppo, incluso il deployment. Anche un

singolo file dimenticato nella cartella pubblica può, da solo, vanificare tutte le protezioni applicative, a conferma del principio secondo cui "la resistenza di una catena è data dalla resistenza del suo anello più debole".

3.5.2 Contesto tecnico: l'hashing delle password

È importante comprendere il meccanismo dell'hashing per apprezzare appieno le implicazioni della vulnerabilità introdotta dall'uso di MD5 senza salt. L'hashing è una funzione crittografica unidirezionale che trasforma una password o qualsiasi altro dato di lunghezza arbitraria in un valore di lunghezza fissa, detto *digest*. Questa operazione è concepita per essere facile da calcolare in avanti, ma quasi impossibile da crackare, garantendo che l'hash non consenta di recuperare la password originale in tempi praticabili. Quando una password viene hashata e salvata, in caso di compromissione del database l'attaccante non ha accesso alla password in chiaro, ma solo al relativo digest [80, 81].

Per rendere l'hashing più sicuro si introduce una stringa casuale chiamata *salt*, unica per ciascun utente, che viene concatenata alla password prima di calcolare l'hash. Il salt impedisce che utenti diversi con la stessa password generino lo stesso hash, e rende inutilizzabili attacchi con rainbow table pre-calcolate perché l'attaccante dovrebbe rigenerarle per ogni possibile salt [82]. Il salt è memorizzato insieme all'hash: non deve rimanere segreto, ma serve a garantire che ogni combinazione password+salt sia unica [82].

Ulteriormente, per aumentare notevolmente il costo computazionale di un attacco, si usano funzioni di hashing adattative come PBKDF2, bcrypt, Argon2 e scrypt, che applicano iterazioni multiple o uso intensivo di memoria, rallentando drasticamente ogni tentativo di brute-force. Questi algoritmi sono raccomandati da standard come OWASP e NIST proprio per rendere inefficaci attacchi con hardware moderno, come GPU e FPGA [80, 81].

Al contrario, funzioni hash rapide e lineari, come MD5 o SHA-1, non prevedono

salt integrato né costi computazionali elevati, e perciò sono facilmente attaccabili con strumenti moderni o tabelle pre-calcolate. In quel caso, in pochi secondi o minuti è possibile crackare l'hash e ricavare la password in chiaro [83].

Un meccanismo sicuro di salvataggio degli hash delle password richiede l'uso di un salt unico per ogni utente, l'impiego di funzioni hash lente o ad alto costo computazionale e una gestione attenta affinché hash e salt siano salvati insieme e disponibili per la verifica [80, 81]. È opportuno precisare che il salt e la funzione di hashing sono due elementi distinti e possono essere implementati separatamente. La funzione hash è un algoritmo crittografico che, presa in ingresso una sequenza di byte, produce un digest a lunghezza fissa; difatti, se due utenti hanno la stessa password, il risultato dell'hash puro sarà identico [84]. Per questo motivo, in fase applicativa, si utilizza il salt: una stringa casuale generata per ciascun utente e concatenata alla password prima di calcolare l'hash. Così anche password uguali producono digest differenti [80, 82].

Il salt non appartiene alla definizione matematica dell'algoritmo hash (ad es. MD5, SHA-256, bcrypt), ma è una misura di protezione esterna al puro algoritmo [84]. Questo significa che sistemi diversi possono decidere come e dove generare il salt, come salvarlo (ad esempio nello stesso campo dell'hash o in un campo separato del database) e come usarlo in fase di verifica della password [80].

In fase di registrazione o di cambio password, il server genera un salt casuale e lo associa all'account dell'utente, generalmente identificato dall'username. Questo salt viene salvato nel database insieme all'hash calcolato. Durante il login, quando l'utente inserisce la propria password, il server recupera il salt corrispondente all'username e lo utilizza per ricalcolare l'hash, confrontandolo poi con quello salvato. In questo modo, il server sa sempre quale salt usare per ciascun utente e garantisce che anche due utenti con la stessa password abbiano hash differenti. Tutto il processo avviene lato server: il client non deve conoscere né inviare il salt [80, 81].

Oltre al salt, esiste anche la tecnica del pepper, che consiste nell'aggiungere alla password (o alla password + salt) un valore segreto fisso, detto appunto pepper. A differenza del salt, il pepper non viene salvato nel database insieme all'hash, ma è custodito in un ambiente sicuro (ad esempio in un hardware security module o in un file di configurazione con accesso ristretto). Questo implica che, anche se un attaccante sottrae il database degli hash e conosce i salt, non è comunque in grado di verificare facilmente le password senza possedere il pepper segreto, aggiungendo un ulteriore livello di protezione [80, 81].

Questo approccio rende la compromissione delle password nel database estremamente costosa e impraticabile su vasta scala. Nel caso della challenge, invece, il backup lasciato libero contiene hash MD5 non salati, rendendo immediatamente possibile la loro decodifica con strumenti come **John the Ripper** e compromettendo l'intera sicurezza del sistema [83].

È opportuno discutere anche le proprietà che caratterizzano le funzioni di hashing *crittografiche*, poiché non tutte le funzioni hash comunemente utilizzate sono adatte a scopi di sicurezza informatica. Una funzione hash generica può essere progettata semplicemente per efficienza o per calcolare somme di controllo (checksum), senza garantire adeguate proprietà di sicurezza [84, 85].

Affinché una funzione hash sia considerata *crittografica*, deve possedere alcune proprietà fondamentali [84, 86]:

- **Preimage resistance (resistenza al preimage):** dato un output hash h , deve risultare computazionalmente impraticabile trovare un input m tale che $hash(m) = h$. In altre parole, non deve essere possibile risalire al messaggio originale a partire dal digest.
- **Second preimage resistance (resistenza alla seconda immagine):** dato un input m_1 , deve essere impraticabile trovare un altro input m_2 ($m_1 \neq m_2$) tale che $hash(m_1) = hash(m_2)$. Questa proprietà è essenziale

per evitare che dati differenti producano lo stesso hash.

- **Collision resistance (resistenza alle collisioni)**: deve essere impraticabile trovare due input distinti m_1 e m_2 tali che $hash(m_1) = hash(m_2)$. Le collisioni, se facilmente trovabili, possono compromettere firme digitali, autenticazione e integrità dei dati.

Oltre a queste proprietà fondamentali, nelle applicazioni moderne si richiede che la funzione hash sia veloce da calcolare in avanti ma al tempo stesso non invertibile, e che il minimo cambiamento nell'input (anche un singolo bit) produca una variazione radicale e imprevedibile dell'output, fenomeno noto come *avalanche effect* [85, 86].

Funzioni come MD5 o SHA-1, sebbene un tempo considerate crittografiche, non sono più ritenute sicure poiché vulnerabili ad attacchi che permettono di trovare collisioni in tempi computazionalmente accettabili [87, 88]. Per questo, gli standard attuali raccomandano l'uso di funzioni hash più robuste come SHA-256, SHA-3 o algoritmi di hashing specificamente progettati per la sicurezza delle password, come bcrypt, scrypt o Argon2 [84, 80], quando si parla di *hashing crittografico*, non ci si riferisce semplicemente alla produzione di digest, ma all'uso di algoritmi che garantiscano queste proprietà di resistenza agli attacchi, condizione imprescindibile per la sicurezza delle applicazioni moderne.

3.5.3 Workflow dell'exploit

Accesso iniziale al portale.

Nella terza challenge non vengono fornite credenziali di accesso. Il partecipante si trova inizialmente davanti a una pagina web apparentemente statica e priva di indizi utili. Non essendoci moduli di login né informazioni visibili, l'unico approccio possibile è avviare una fase di raccolta informazioni sul server, alla ricerca di eventuali risorse nascoste o residue.

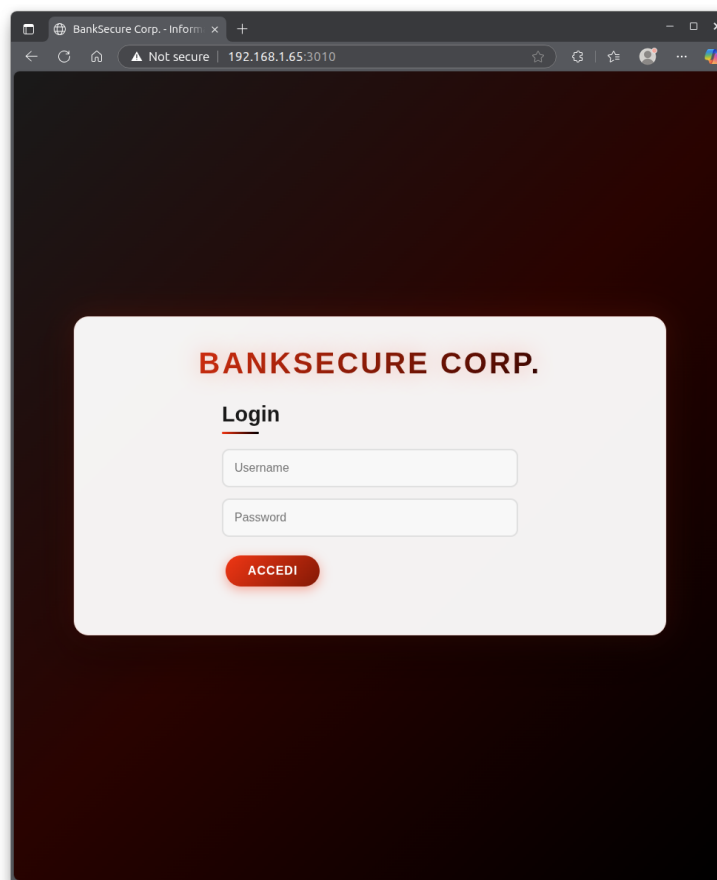


Figura 3.17: Accesso alla terza challenge

Scanning delle directory.

La prima attività operativa consiste nell'eseguire uno scanning delle directory e delle risorse esposte sul server. Strumenti come **Gobuster** si rivelano particolarmente utili per identificare file nascosti, cartelle non documentate o residui di versioni precedenti dell'applicazione, potenzialmente rimasti accessibili online.

```
(kali㉿kali)-[~]
└─$ gobuster dir \
  -u http://192.168.1.66:3010/ \
  -w /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt \
  -x css,js,js.bak,bak,old,backup,map,html,svg,png,jpg,gif,json,txt \
  -t 60 -b 403,401,404 -o gobuster_static.txt

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://192.168.1.66:3010/
[+] Method: GET
[+] Threads: 60
[+] Wordlist: /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
[+] Negative Status codes: 401,404,403
[+] User Agent: gobuster/3.6
[+] Extensions: js.bak,old,html,svg,jpg,gif,css,js,bak,backup,ma
p,png,json,txt
[+] Timeout: 10s

Starting gobuster in directory enumeration mode

/index.html (Status: 200) [Size: 1103]
/server.js.bak (Status: 200) [Size: 3746]
/style.css (Status: 200) [Size: 5322]
/script.js (Status: 200) [Size: 4684]
Progress: 1314960 / 1314975 (100.00%)

Finished
```

Figura 3.18: Esito ottenuto con gobuster

Questo rappresenta un classico caso di *information disclosure*, dove file dimenticati sul server contengono dati critici che dovrebbero rimanere riservati.

Prelievo file esposti

Come visibile nella Figura 3.18 dall'analisi di Gobuster sono stati trovati i file `index.html`, `server.js.bak`, `style.css`, `script.js`, risulta imperativo andare a prelevare il file `server.js.bak`, azione mostrata in 3.19 per vederne il contenuto.

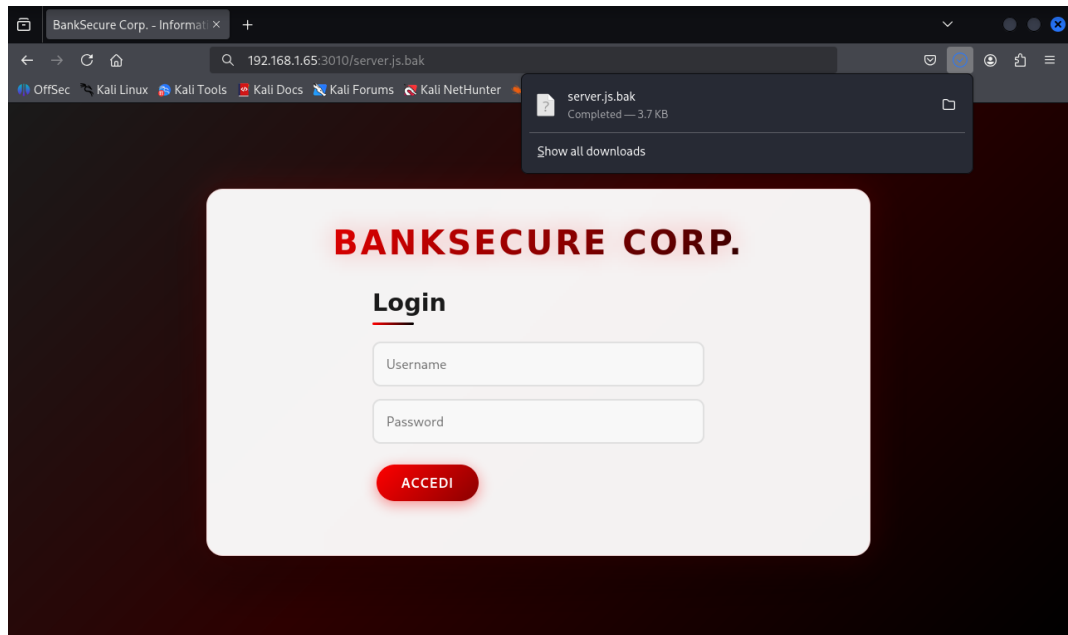


Figura 3.19: Download di `server.js.bak`

Analisi del file di backup

Una volta scaricato il file, possiamo osservare che è una vecchia versione del file `server.js` contenente gli hash delle password in MD5, ma è mancante la **secret key** per il token JWT (3.20).

```
GNU nano 8.4 server.js.bak
const express = require('express');
const jwt = require('jsonwebtoken');
const crypto = require('crypto');

const app = express();
const PORT = 3010;
const SECRET_KEY = ' ';
const HOST = '0.0.0.0';

app.use(express.json());
app.use(express.static('public'));

// Funzione per creare hash MD5
function hashPassword(password) {
  return crypto.createHash('md5').update(password).digest('hex');
}

let users = [
  {
    id: 1,
    username: 'giampaolo',
    password: 'ef1a5131a0f62733d8b599b93658e829',
    role: 'standard',
    personalData: {
      creditCard: "4539-7894-5698-1234",
      codiceFiscale: "RSSMRA80A01H501U",
      iban: "IT60X0542811101000000123456"
    }
  },
  {
    id: 2,
    username: 'paolo',
    password: 'f1bdf5ed1d7ad7ede4e3809bd35644b0',
    role: 'standard',
    personalData: {
      creditCard: "4532-1234-5678-9012",
      codiceFiscale: "VRDLGU85B15H501V",
      iban: "IT60X0542811101000000789012"
    }
  }
];
```

[Read 134 lines]

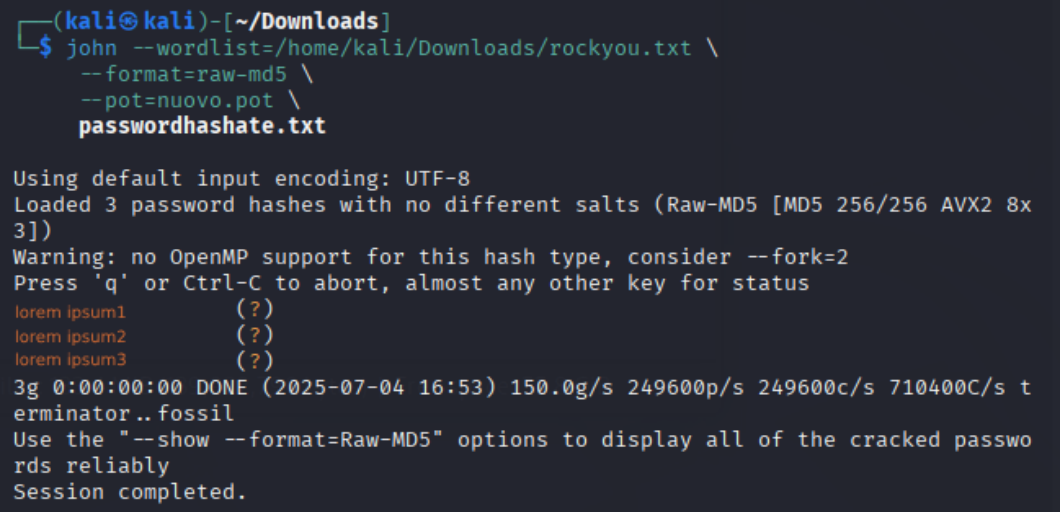
Help **Write Out** **Where Is** **Cut** **Execute**
Exit **Read File** **Replace** **Paste** **Justify**

Figura 3.20: Analisi server.js.bak

Ottenuti gli hash, il partecipante prosegue tentando di ricavare le password in chiaro.

Hash cracking

Per scopi didattici, la challenge impiega l'algoritmo MD5, noto per la sua debolezza e facilmente vulnerabile tramite strumenti come **John the Ripper**. È opportuno evidenziare che esistono numerosi strumenti capaci di eseguire le stesse operazioni di attacco sugli hash. Oltre a **John the Ripper**, software come **Hashcat**, **Cain and Abel**, **Ophcrack** e **Hydra** consentono di effettuare attacchi di tipo brute-force, dictionary e rainbow table contro hash MD5 e altri algoritmi crittografici [89, 90, 91, 92]. Tali strumenti supportano anche l'uso di GPU moderne, accelerando notevolmente la velocità degli attacchi rispetto ai metodi tradizionali. Quelli citati sono solamente alcuni dei tool esistenti che usano questo approccio, ne esistono svariati, che seguendo lo stesso principio, vengono adoperati in scenari differenti. L'esistenza di ampie wordlist o rainbow table pubbliche rende questa operazione rapida ed efficace, l'esito è mostrato in Figura 3.21.



```
(kali㉿kali)-[~/Downloads]
$ john --wordlist=/home/kali/Downloads/rockyou.txt \
  --format=raw-md5 \
  --pot=nuovo.pot \
  passwordhashate.txt

Using default input encoding: UTF-8
Loaded 3 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x
3])
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
lorem ipsum1      (?)
lorem ipsum2      (?)
lorem ipsum3      (?)
3g 0:00:00:00 DONE (2025-07-04 16:53) 150.0g/s 249600p/s 249600c/s 710400C/s t
erminator.. fossil
Use the "--show --format=Raw-MD5" options to display all of the cracked passwo
rds reliably
Session completed.
```

Figura 3.21: Esito ottenuto con John The Ripper

Recuperate le password, il partecipante può autenticarsi nel portale web accedendo ai tre diversi account utente presenti nel sistema. Ogni account contiene informazioni specifiche e consente di ottenere flag differenti, una per ogni account.

Una volta trovate le password in chiaro, basta accedere al portale con le

rispettive credenziali, usando gli username presenti nel file `server.js.bak` per trovare la rispettiva flag per ogni account.

L'obiettivo educativo di questa challenge è duplice: da un lato, mostrare come la cattiva gestione dei file residui sul server possa comportare fughe di informazioni sensibili; dall'altro, evidenziare i rischi legati all'uso di algoritmi crittografici deboli come MD5, la cui vulnerabilità è ben documentata sia dalle linee guida OWASP sia dal National Institute of Standards and Technology (NIST) [93, 94]. Attraverso questo workflow, gli studenti sperimentano tecniche pratiche di *information gathering* e di attacco contro sistemi insufficientemente protetti dal punto di vista crittografico.

3.5.4 Possibili mitigazioni

La mitigazione di tali vulnerabilità richiede interventi su più livelli, che spaziano dall'organizzazione del ciclo di sviluppo alla scelta degli strumenti crittografici. Per quanto riguarda la gestione dei file di backup e dei file temporanei, è fondamentale adottare pratiche DevSecOps che prevedano un controllo rigoroso sui contenuti effettivamente distribuiti in produzione. OWASP raccomanda di configurare il web server in modo da impedire l'accesso a file con estensioni tipiche di backup, come `.bak`, `.old` o `.swp`, attraverso regole esplicite nel file di configurazione o tramite strumenti di sicurezza come i Web Application Firewall (WAF), capaci di bloccare richieste anomale verso percorsi non previsti [95]. Inoltre, l'automazione del deployment tramite pipeline CI/CD consente di integrare scansioni statiche del file system, al fine di identificare eventuali file residui o artefatti di sviluppo prima che questi vengano caricati sul server di produzione [96].

Dal punto di vista crittografico, la prima misura necessaria consiste nell'abbandonare completamente l'uso di algoritmi hash non più considerati sicuri come MD5 o SHA-1, che secondo NIST e OWASP non devono essere utilizzati né per il salvataggio delle password né per la generazione di firme digitali [94,

80]. Al loro posto, è opportuno impiegare algoritmi di hashing adattativi come bcrypt, Argon2 o scrypt, i quali introducono costi computazionali elevati, tali da rendere impraticabile un attacco di forza bruta o tramite hardware parallelo come GPU o FPGA. Questi algoritmi consentono di parametrizzare il livello di difficoltà in base alla potenza hardware disponibile, offrendo così una protezione durevole nel tempo [80, 81].

In aggiunta, ogni password deve essere combinata a un salt casuale, unico per ciascun utente, memorizzato insieme all'hash nel database. Questa tecnica ha il duplice vantaggio di prevenire l'uso di rainbow table precalcolate e di garantire che utenti con la stessa password non generino lo stesso digest [82], come spiegato in precedenza. È altrettanto cruciale implementare politiche di gestione delle credenziali che impongano password robuste, lunghe e sufficientemente complesse, in linea con le raccomandazioni più recenti del NIST [81]. Tali politiche riducono sensibilmente la probabilità che le password possano essere individuate attraverso dizionari comuni o attacchi mirati.

Un'ulteriore misura di mitigazione riguarda la gestione delle informazioni sensibili all'interno del codice sorgente. Dati come chiavi segrete, credenziali o stringhe di connessione non dovrebbero mai essere inclusi nel codice né tanto meno nei file di backup. Devono essere invece estratti e gestiti attraverso sistemi sicuri di secret management, come vault dedicati o variabili d'ambiente, in modo da ridurre al minimo il rischio di leakage in caso di compromissione del repository o del server [97].

Nella difesa contro scenari come quello illustrato nella terza challenge è assolutamente necessario un approccio olistico alla sicurezza, che includa sia buone pratiche di sviluppo e deployment sia scelte crittografiche aggiornate. È la conferma del principio secondo cui la sicurezza non è garantita da singole misure isolate, bensì dal coordinamento di più controlli sovrapposti, capaci di intercettare errori o dimenticanze prima che questi diventino vulnerabilità sfruttabili.

Capitolo 4

Integrazione con la piattaforma CTF

4.1 Obiettivi dell'integrazione

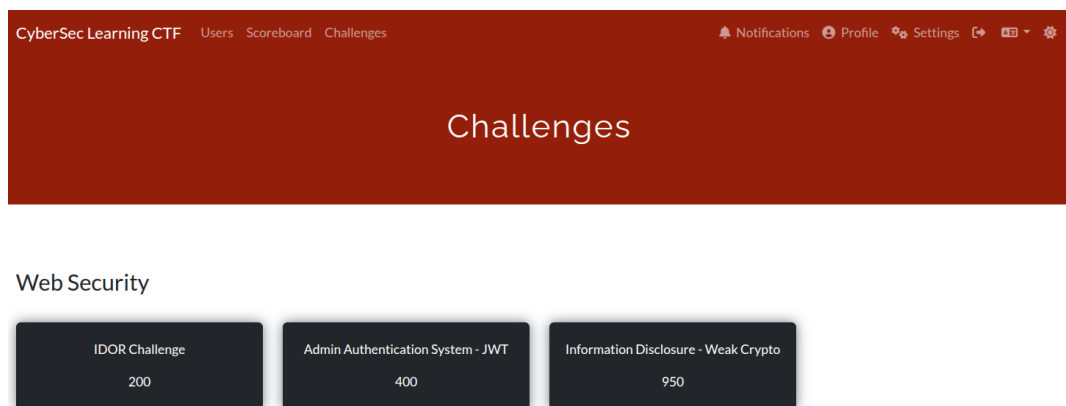


Figura 4.1: Challenge presenti su CTFd

L'integrazione delle challenge sviluppate in questo lavoro visibili in Figura 4.1, all'interno di una piattaforma Capture The Flag (CTF), ha rappresentato un'evoluzione fondamentale per trasformare semplici esercizi tecnici in un'esperienza formativa strutturata e replicabile. Sebbene la progettazione delle singole challenge costituisca il cuore tecnico del progetto, è proprio l'inserimento in un contesto organizzato come CTFd a consentire di passare da attività isolate a un vero percorso didattico, capace di coinvolgere gli studenti in maniera interattiva e motivante [98, 99].

Uno degli obiettivi principali dell'integrazione è stato quello di offrire un'interfaccia unificata attraverso la quale gli studenti potessero accedere alle diverse challenge, risolverle e tenere traccia dei propri progressi. La letteratura nel campo della cybersecurity education sottolinea come uniformità e standardizzazione siano elementi fondamentali per garantire condizioni di apprendimento e di valutazione comparabili tra gli studenti, evitando variabili ambientali che potrebbero influenzare la percezione di difficoltà delle prove [99].

Dal punto di vista didattico, l'integrazione consente inoltre di raccogliere in modo sistematico dati preziosi sull'andamento degli studenti. I parametri visibili in figura 4.2 come il tempo medio di risoluzione delle challenge, il numero di tentativi effettuati per individuare la flag corretta o la frequenza di utilizzo degli hint, offrono agli insegnanti una base oggettiva per analizzare le vulnerabilità più ostiche, le tecniche meno comprese o i punti su cui sia opportuno intervenire con materiale integrativo o spiegazioni supplementari.

Questo approccio si inserisce perfettamente nelle moderne strategie educative, che raccomandano un monitoraggio costante delle attività didattiche per ottimizzare i percorsi di apprendimento [100].

L'obiettivo non è stato solo ottimizzare la gestione operativa delle challenge, ma anche offrire agli studenti un'esperienza didattica quanto più vicina a scenari reali. Le competizioni CTF, anche simulate in ambito universitario, sono infatti riconosciute come uno degli strumenti più efficaci sia nel contesto accademico sia in quello professionale. Organizzazioni come OWASP e (ISC)² sottolineano come il modello CTF favorisca lo sviluppo di competenze pratiche difficilmente acquisibili con la sola teoria, offrendo un banco di prova concreto per applicare strumenti e metodologie tipiche del penetration testing [74, 100].

L'integrazione mira a garantire scalabilità e ripetibilità dell'ambiente formativo. Una piattaforma ben configurata consente di distribuire rapidamente nuove challenge, supportare molti utenti contemporanei e gestire in modo ordinato elementi come hint, punteggi e tracciamento delle attività. Questo è cruciale

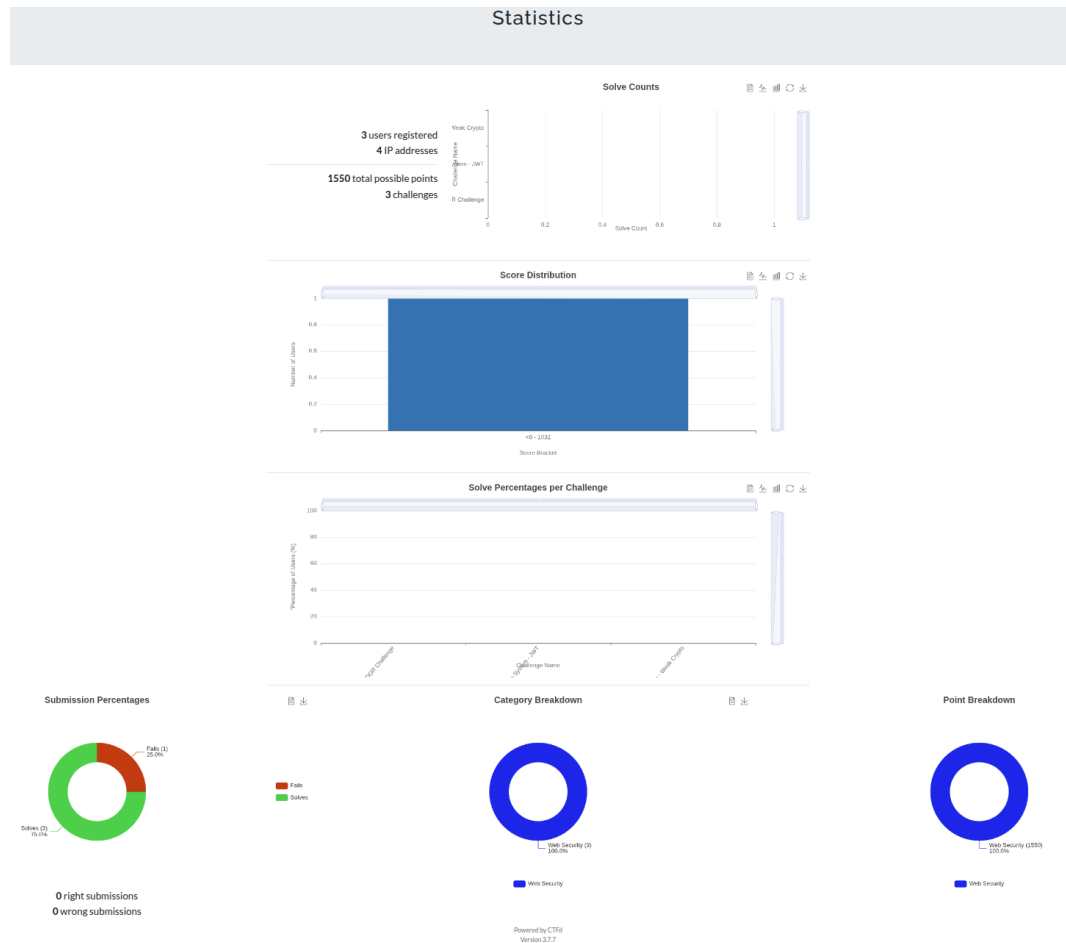


Figura 4.2: Statistiche CTFd

per estendere il progetto a nuove edizioni, diversi gruppi di studenti o altri contesti accademici, senza necessità di riconfigurazioni complesse. L'uso di CTFd non è stato solo una scelta tecnica, ma una decisione strategica per costruire un ambiente formativo solido, scalabile e conforme agli standard professionali nel settore della sicurezza informatica. L'obiettivo ultimo è stato offrire un'esperienza didattica pratica, valutabile oggettivamente e replicabile in contesti futuri.

CTFd si integra perfettamente con i principi della gamification, incentivando la partecipazione e l'apprendimento attivo. Studi evidenziano che il formato Jeopardy-style, su cui si basa CTFd, aumenta l'engagement e può essere utilizzato anche come strumento di valutazione [101, 102]. Le sue dinamiche (leaderboard, hint a costo, punteggi automatici) si sposano bene con il modello

motivazionale del “flow” [17], in cui attività sfidanti, adeguate al livello di competenza, stimolano un coinvolgimento profondo. Inoltre, la piattaforma è stata suggerita dal mio relatore, prof. Sergio Esposito, per la sua velocità, il supporto al formato Jeopardy e l’integrazione con container Docker e ambienti di laboratorio.

Il collegamento tra le challenge containerizzate e CTFd avviene tramite la creazione, nella piattaforma, di entità challenge configurate con titolo, descrizione dello scenario tecnico o aziendale, punteggio calibrato sulla difficoltà e flag corretta. È possibile aggiungere hint con eventuale costo in punti per bilanciare l’aiuto con la riduzione del punteggio massimo [103].

Nel progetto, l’URL verso il servizio containerizzato è inserito nella descrizione della challenge, consentendo agli studenti di accedere facilmente, tramite browser o strumenti di penetration testing, all’applicazione vulnerabile isolata nel container Docker. In questo modo, componente infrastrutturale e componente didattica rimangono separate ma integrate. Ogni container espone una porta specifica definita nel file `docker-compose.yml`, mentre CTFd svolge il ruolo di interfaccia e scoring, delegando l’esecuzione delle challenge ai servizi isolati.

CTFd gestisce anche la validazione automatica delle flag, confrontando quella inserita dallo studente con quella memorizzata nel sistema, aggiornando in tempo reale la classifica [104]. Come detto, il collegamento al container avviene tramite link ipertestuale nella descrizione.

L’integrazione con CTFd ha portato diversi vantaggi. Ha permesso una gestione centralizzata degli utenti e delle sessioni, semplificando sia la didattica sia l’organizzazione di eventi competitivi. Inoltre, l’assegnazione automatica dei punteggi e la leaderboard hanno reso l’ambiente più coinvolgente e motivante per gli studenti. Sul piano operativo, la possibilità di eseguire sia CTFd sia le challenge in Docker ha semplificato notevolmente il deploy dell’ecosistema CTF, riducendo la complessità tipica degli ambienti multi-servizio e garantendo uniformità tra i vari ambienti di esecuzione [103].

Capitolo 5

Conclusioni

Il lavoro svolto in questa tesi ha avuto come obiettivo principale la creazione di un framework documentato per la progettazione, implementazione e distribuzione di challenge Capture The Flag (CTF) containerizzate, con l'intento di colmare il divario formativo spesso riscontrato nell'insegnamento della cybersecurity, dove la didattica tradizionale fatica a trasmettere competenze pratiche realmente spendibili in scenari operativi complessi.

Dal punto di vista tecnico, il progetto ha portato alla realizzazione di una raccolta di challenge CTF pronte all'uso, corredate dei relativi Dockerfile e di una documentazione dettagliata per il deployment e la manutenzione. Tutte le challenge si fondano su un'infrastruttura comune che comprende, tra gli altri componenti, un sistema di autenticazione basato su JSON Web Token (JWT), la gestione di utenti hardcoded e uno stack tecnologico omogeneo basato su Node.js ed Express.js per il backend, e HTML5, CSS3 e JavaScript vanilla per il frontend. L'adozione della containerizzazione tramite Docker e Docker Compose ha garantito isolamento, portabilità e semplicità di distribuzione, risolvendo le problematiche di inconsistenza ambientale che spesso compromettono l'efficacia delle esercitazioni didattiche.

Sul piano pratico, sono state implementate tre challenge significative, ciascuna progettata per rappresentare diverse tipologie di vulnerabilità e scenari di attacco. La prima challenge ha illustrato lo sfruttamento di una vulnerabilità

di tipo Insecure Direct Object Reference (IDOR) e le relative tecniche di bypass di un Web Application Firewall (WAF) attraverso meccanismi come l'URL encoding, proponendo al contempo mitigazioni specifiche. La seconda challenge si è concentrata su una vulnerabilità di JWK Header Parameter Injection nei JSON Web Token, mettendo in luce i rischi legati alla gestione impropria delle chiavi pubbliche fornite dal client e indicando contromisure quali la whitelisting degli algoritmi e il rifiuto di chiavi esterne non autorizzate. Infine, la terza challenge ha dimostrato come la combinazione tra l'esposizione involontaria di file di backup e l'utilizzo di algoritmi crittografici deboli come MD5 senza salt possa condurre a compromissioni critiche, sottolineando l'importanza di buone pratiche lungo l'intero ciclo di vita dello sviluppo software.

Dal punto di vista metodologico, il lavoro ha sistematizzato il processo di dockerizzazione delle challenge CTF, accompagnandolo con una documentazione approfondita sulle decisioni progettuali e sulle difficoltà incontrate durante lo sviluppo. Questo approccio punta a facilitare il riuso delle challenge e a ridurre la duplicazione di sforzi, promuovendo la creazione di un patrimonio condiviso di risorse didattiche. La metodologia adottata ha seguito un'impostazione sperimentale e iterativa, fondata sui principi del learning by doing e su una progressione graduale della difficoltà, con l'obiettivo di permettere agli studenti di sviluppare competenze pratiche in modo strutturato.

Sul piano didattico, ciascuna challenge è stata concepita non solo come esercizio tecnico, ma come caso di studio finalizzato a mostrare come progettare esperienze di apprendimento efficaci. L'integrazione delle challenge nella piattaforma CTFd ha trasformato le esercitazioni tecniche in percorsi formativi strutturati e replicabili, grazie a un'interfaccia unificata che consente di tracciare i progressi degli studenti e di gestire il contesto di gara in modo ordinato e motivante. La scelta di utilizzare il formato Jeopardy-style, insieme a scenari narrativi realistici come quello aziendale della BankSecure Corp., ha permesso di coniugare il realismo tecnico con l'accessibilità didattica, mantenendo alta la motivazione degli studenti e facilitando il trasferimento delle competenze al

contesto professionale.

Il lavoro presentato in questa tesi non si limita a fornire una serie di risorse tecniche pronte all'uso, ma propone anche una metodologia chiara e replicabile per la creazione di ambienti formativi pratici nel campo della cybersecurity. Contribuisce così a migliorare l'efficacia dell'insegnamento e dell'apprendimento delle competenze di sicurezza informatica, attraverso un'esperienza pratica e contestualizzata che risponde alle reali esigenze del settore.

Bibliografia

- [1] European Union Agency for Cybersecurity. *ENISA Threat Landscape 2023*. Available at <https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023>. 2023 (cit. a p. 5).
- [2] OWASP Foundation. *OWASP Top Ten 2021*. Available at <https://owasp.org/www-project-top-ten/>. 2021 (cit. a p. 5).
- [3] P. Casey, T. Le e A. Ghaffar. «Capture The Flag as Cybersecurity Education: A Literature Review». In: *IEEE Security & Privacy* 18.2 (2020), pp. 11–19. DOI: 10.1109/MSEC.2020.2967495 (cit. a p. 5).
- [4] Z. Schreuders, J. Payne e P. Chester. «Practical and Effective Teaching of Network Security Through Active Learning and Real-World Context». In: *IEEE Transactions on Education* 60.3 (2017), pp. 205–211. DOI: 10.1109/TE.2017.2658578 (cit. a p. 5).
- [5] Kaspersky Lab. *Kaspersky CyberSecurity CTF: Official Website*. Available at <https://ctf.kaspersky.com>. 2023 (cit. a p. 6).
- [6] Dirk Merkel. «Docker: lightweight Linux containers for consistent development and deployment». In: *Linux Journal*. 239. 2014, p. 2 (cit. a p. 6).
- [7] MITRE Corporation. *CWE Top 25 Most Dangerous Software Weaknesses*. Rapp. tecn. Available at https://cwe.mitre.org/top25/archive/2023/2023_cwe_top25.html. 2023 (cit. a p. 6).
- [8] National Institute of Standards and Technology. *Security and Privacy Controls for Information Systems and Organizations, SP 800-53 Revision*

5. Rapp. tecn. Available at <https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final>. 2020 (cit. a p. 7).
- [9] MITRE Corporation. *2024 CWE Top 25 Most Dangerous Software Weaknesses*. Accessed: 2025-07-03. 2024. URL: https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html (cit. alle pp. 9, 10).
- [10] OWASP Foundation. *OWASP Top Ten Web Application Security Risks 2021*. Accessed: 2025-07-03. 2021. URL: <https://owasp.org/www-project-top-ten/> (cit. a p. 9).
- [11] Forum of Incident Response and Security Teams (FIRST). *Common Vulnerability Scoring System v4.0 Specification Document*. Accessed: 2025-07-03. 2023. URL: <https://www.first.org/cvss/specification-document> (cit. a p. 10).
- [12] National Institute of Standards and Technology (NIST). *Technical Guide to Information Security Testing and Assessment*. Accessed: 2025-07-03. 2008. URL: <https://csrc.nist.gov/publications/detail/sp/800-115/final> (cit. a p. 10).
- [13] C. Cowan, S. Arnold, S. Beattie, C. Wright e J. Viega. «Defcon Capture the Flag: defending vulnerable code from intense attack». In: *Proceedings DARPA Information Survivability Conference and Exposition*. Vol. 1. 2003, 120–129 vol.1. DOI: 10.1109/DISCEX.2003.1194878 (cit. a p. 11).
- [14] Newton Lee. «Darpa’s cyber grand challenge (2014–2016)». In: *Counterterrorism and Cybersecurity: Total Information Awareness*. Springer, 2015, pp. 429–456 (cit. a p. 11).
- [15] Jean Piaget. *Piaget’s theory*. Springer, 1976 (cit. a p. 13).
- [16] Jerome Bruner. «Celebrating divergence: Piaget and vygotsky». In: *Human development* 40.2 (1997), pp. 63–73 (cit. a p. 13).
- [17] Mihaly Csikszentmihalyi. *Mihaly Csikszentmihalyi, Flow*. Into the Classroom Media, 2003 (cit. alle pp. 13, 77).
- [18] Lev Vygotsky. «Lev Vygotsky». In: *La psicología en la Revolución Rusa. Colombia: Ediciones desde abajo* (2018) (cit. a p. 14).

- [19] Mary Forehand. «Bloom’s taxonomy». In: *Emerging perspectives on learning, teaching, and technology* 41.4 (2010), pp. 47–56 (cit. a p. 14).
- [20] CTFd Authors. *CTFd Documentation*. <https://docs.ctfd.io>. Accessed: 2025-07-03 (cit. a p. 16).
- [21] Facebook, Inc. *FBCTF*. <https://github.com/facebookarchive/fbctf>. Accessed: 2025-07-03 (cit. a p. 16).
- [22] RootTheBox Developers. *RootTheBox*. <https://github.com/moloch--/RootTheBox>. Accessed: 2025-07-03 (cit. a p. 16).
- [23] Nakiami. *Mellivora*. <https://github.com/Nakiami/mellivora>. Accessed: 2025-07-03 (cit. a p. 17).
- [24] VulnHub. *VulnHub*. <https://www.vulnhub.com/>. Accessed: 2025-07-03 (cit. a p. 17).
- [25] Hack The Box Ltd. *Hack The Box*. <https://www.hackthebox.com/>. Accessed: 2025-07-03 (cit. alle pp. 17, 18).
- [26] TryHackMe Ltd. *TryHackMe*. <https://tryhackme.com/>. Accessed: 2025-07-03 (cit. alle pp. 17, 18).
- [27] Igor Bisio, Maurizio Mongelli, Francesco Lavagetto e Francesco Cuomo. «Cyber Ranges: An Overview of Features, Challenges, and Opportunities». In: *IEEE Communications Surveys & Tutorials* 23.1 (2021), pp. 464–489. DOI: 10.1109/COMST.2020.3032795 (cit. alle pp. 17, 27).
- [28] Antoine Besnard, Karima Boudaoud e Yassine Maleh. «Challenges and Directions for Cyber Range Development». In: *Computers & Security* 104 (2021), p. 102224. DOI: 10.1016/j.cose.2021.102224 (cit. a p. 17).
- [29] OWASP Foundation. *OWASP Docker Security Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html. Accessed: 2025-07-03 (cit. a p. 17).
- [30] PortSwigger Ltd. *Web Security Academy*. 2025. URL: <https://portswigger.net/web-security> (visitato il giorno 01/07/2025) (cit. a p. 20).

- [31] M. B. Jones, J. Bradley e N. Sakimura. *JSON Web Token (JWT)*. <https://datatracker.ietf.org/doc/html/rfc7519>. RFC 7519. 2015 (cit. alle pp. 21–23).
- [32] N. Ferguson. *Practical Cryptography in Python: Learning Correct Cryptographic Protocols*. Apress, 2022 (cit. alle pp. 21–23).
- [33] Dan Moore. *Components of JWTs Explained*. FusionAuth blog. consultato 10 luglio 2025. 2025. URL: <https://fusionauth.io/articles/tokens/jwt-components-explained> (cit. a p. 21).
- [34] E. Barker, A. Roginsky, M. Smid e D. Branstad. *Recommendation for Key Management – Part 1: General (Revision 4)*. Rapp. tecn. NIST SP 800-57 Part 1 Rev. 4. National Institute of Standards e Technology (NIST), 2015 (cit. alle pp. 22, 23).
- [35] Stack Overflow. *2024 Developer Survey – Technology*. Accessed: 2025-07-01. 2024. URL: <https://survey.stackoverflow.co/2024/technology> (cit. a p. 24).
- [36] Express.js Contributors. *Express.js Documentation*. Accessed: 2025-07-01. 2025. URL: <https://expressjs.com/> (cit. a p. 24).
- [37] Node.js Contributors. *Crypto / Node.js Documentation*. Accessed: 2025-07-01. 2025. URL: <https://nodejs.org/api/crypto.html> (cit. a p. 24).
- [38] Inc. Auth0. *jsonwebtoken - npm*. Accessed: 2025-07-01. 2025. URL: <https://www.npmjs.com/package/jsonwebtoken> (cit. a p. 24).
- [39] Nielsen Norman Group. *10 Usability Heuristics for User Interface Design*. Accessed: 2025-07-01. 2025. URL: <https://www.nngroup.com/articles/ten-usability-heuristics/> (cit. a p. 25).
- [40] Docker Inc. *Dockerfile Best Practices*. Accessed: 2025-07-01. 2025. URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ (cit. a p. 26).

- [41] OWASP Foundation. *Docker Security Cheat Sheet*. Accessed: 2025-07-01. 2025. URL: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html (cit. a p. 26).
- [42] Alpine Linux. *Alpine Linux Documentation*. Accessed: 2025-07-01. 2025. URL: <https://alpinelinux.org/about/> (cit. a p. 26).
- [43] Docker, Inc. *Dockerfile reference*. <https://docs.docker.com/reference/dockerfile/>. Accessed: 2025-07-05. 2025 (cit. a p. 26).
- [44] Autori vari. «A systematic review of current cybersecurity training methods». In: *Computers & Security* (2023). Analizza le problematiche pratiche nella creazione di esercitazioni vulnerabili (cit. a p. 27).
- [45] Prey Project. *Cybersecurity challenges in educational settings*. Accessed: 2025-07-01. 2024. URL: <https://preyproject.com/blog/cybersecurity-challenges-in-education> (cit. a p. 27).
- [46] David A. Kolb. *Experiential Learning Theory*. Accessed: 2025-07-01. 1984. URL: https://en.wikipedia.org/wiki/Kolb%27s_experiential_learning (cit. a p. 27).
- [47] Docker Inc. *Docker Compose*. Accessed: 2025-07-01. 2025. URL: <https://docs.docker.com/compose/> (cit. a p. 30).
- [48] OWASP Cheat Sheet Series. *Insecure Direct Object Reference Prevention Cheat Sheet*. Online. URL: https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html (cit. a p. 31).
- [49] PortSwigger Ltd. *Server-side request forgery (SSRF) — Web Security Academy*. Accessed: 2025-07-01. 2025. URL: <https://portswigger.net/web-security/ssrf> (cit. a p. 31).
- [50] Brian Krebs. *What We Can Learn from the Capital One Hack*. Accessed: 2025-07-01. 2019. URL: <https://krebsonsecurity.com/2019/08/what-we-can-learn-from-the-capital-one-hack/> (cit. a p. 31).

- [51] Novaes et al. «A Case Study of the Capital One Data Breach». In: *Information Institute Conferences* (2020). Analisi tecnica della violazione WAF/SSRF (cit. a p. 31).
- [52] OWASP Foundation. *Testing JSON Web Tokens (JWT)*. Accessed: 2025-07-01. 2025. URL: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/10-Testing_JSON_Web_Tokens (cit. a p. 33).
- [53] Auth0. *Validate JSON Web Tokens*. Accessed: 2025-07-01. 2025. URL: <https://auth0.com/docs/secure/tokens/json-web-tokens/validate-json-web-tokens> (cit. a p. 33).
- [54] Express.js Contributors. *Using middleware – Express*. Accessed: 2025-07-01. 2025. URL: <https://expressjs.com/en/guide/using-middleware.html> (cit. a p. 34).
- [55] 4Geeks Academy. *Mastering Web App Firewall Evasion: Techniques and Best Practices*. Accessed: 2025-07-01. 2024. URL: <https://4geeks.com/lesson/web-application-firewall-evasion-techniques> (cit. a p. 35).
- [56] Wikipedia contributors. *Double encoding*. Accessed: 2025-07-01. 2025. URL: https://en.wikipedia.org/wiki/Double_encoding (cit. a p. 35).
- [57] Radek Ošlejšek, Vít Rusňák, Karolína Burská, Valdemar Švábenský e Jan Vykopal. «Visual Feedback for Players of Multi-Level Capture the Flag Games: Field Usability Study». In: *arXiv:1912.10781* (2019). Accessed: 2025-07-01 (cit. a p. 37).
- [58] OWASP Foundation. *Testing for Insecure Direct Object References (IDOR)*. Accessed: 2025-07-01. 2025. URL: https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html (cit. a p. 38).

- [59] PortSwigger Ltd. *Burp Suite Documentation*. Online. URL: <https://portswigger.net/burp> (cit. a p. 38).
- [60] NCC Group. *The disadvantages of a blacklist-based approach to input validation*. Accessed: 2025-07-01. 2018. URL: https://www.nccgroup.com/media/ymqf1kww/_ncc-group-whitepaper_the-disadvantages-of-a-blacklist-based-approach-to-input-validation.pdf (cit. a p. 46).
- [61] YesWeHack. *Web application firewall bypass techniques*. Accessed: 2025-07-01. 2022. URL: <https://www.yeswehack.com/learn-bug-bounty/web-application-firewall-bypass> (cit. a p. 46).
- [62] The express-rate-limit contributors. *express-rate-limit: Basic rate-limiting middleware for Express*. npm package. 2025. URL: <https://www.npmjs.com/package/express-rate-limit> (cit. a p. 47).
- [63] Paul J. Leach, Rich Salz e Michael H. Mealling. *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122. Lug. 2005. DOI: 10.17487/RFC4122. URL: <https://www.rfc-editor.org/info/rfc4122> (cit. a p. 47).
- [64] Kyzer R. Davis, Brad Peabody e P. Leach. *Universally Unique IDentifiers (UUIDs)*. RFC 9562. Mag. 2024. DOI: 10.17487/RFC9562. URL: <https://www.rfc-editor.org/info/rfc9562> (cit. a p. 47).
- [65] PortSwigger Web Security Academy. *JWT authentication bypass via jwk header*. Accessed: 2025-07-03. 2025. URL: <https://portswigger.net/web-security/jwt-lab-auth-bypass-jwk> (cit. a p. 48).
- [66] J. Bradley, M. Jones, N. Sakimura et al. *JSON Web Signature (JWS)*. RFC 7515. Accessed: 2025-07-03. Mag. 2015. URL: <https://www.rfc-editor.org/rfc/rfc7515> (cit. alle pp. 49, 51–53).
- [67] Elaine Barker et al. *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Special Publication 800-131A Rev. 2. Accessed: 2025-07-03. National Institute of Standards e Technology, nov. 2019. URL: <https://doi.org/10.6028/NIST.SP.800-131Ar2> (cit. a p. 49).

- [68] Michael B. Jones. *JSON Web Algorithms (JWA)*. RFC 7518. <https://datatracker.ietf.org/doc/html/rfc7518>. 2015 (cit. alle pp. 51–53).
- [69] National Institute of Standards e Technology. *FIPS 198-1: The Keyed-Hash Message Authentication Code (HMAC)*. FIPS 198-1. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf>. 2008 (cit. a p. 52).
- [70] Community. «JWT: Choosing between HMAC and RSA». In: *Security StackExchange* (2018). <https://security.stackexchange.com/questions/220185/jwt-choosing-between-hmac-and-rsa> (cit. alle pp. 52, 53).
- [71] OWASP Foundation. *OWASP JSON Web Token Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_Cheat_Sheet_for_Java.html. Accessed: 2025-07-03 (cit. a p. 59).
- [72] Auth0, Inc. *JWT Security Best Practices*. <https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-security-guidelines>. Accessed: 2025-07-03 (cit. a p. 59).
- [73] Michael B. Jones, John Bradley e Nat Sakimura. *JSON Web Signature (JWS)*. <https://datatracker.ietf.org/doc/html/rfc7515>. Accessed: 2025-07-03. 2015 (cit. a p. 59).
- [74] OWASP Foundation. *OWASP Top Ten 2021*. <https://owasp.org/www-project-top-ten/>. Accessed: 2025-07-04. 2021 (cit. alle pp. 61, 75).
- [75] Xiaoyun Wang, Dengguo Feng, Xuejia Lai e Hongbo Yu. «Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD». In: *Advances in Cryptology - EUROCRYPT 2004*. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 562–577. DOI: 10.1007/978-3-540-24676-3_34 (cit. a p. 62).
- [76] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini e Yarik Markov. «The first collision for full SHA-1». In: *Cryptology ePrint*

- Archive* 2013 (2013). <https://eprint.iacr.org/2013/170>, pp. 1–16 (cit. a p. 62).
- [77] D. Turner. *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. RFC 6151. 2011. URL: <https://datatracker.ietf.org/doc/html/rfc6151> (cit. a p. 62).
- [78] CERT Coordination Center. *Vulnerability Note VU#836068: MD5 vulnerable to collision attacks*. <https://www.kb.cert.org/vuls/id/836068>. Accessed: 2025-07-04. 2008 (cit. a p. 62).
- [79] Philippe Oechslin. «Making a faster cryptanalytic time-memory trade-off». In: *Annual International Cryptology Conference*. Springer. 2003, pp. 617–630 (cit. a p. 62).
- [80] OWASP Foundation. *Password Storage Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html. consultato il 5 luglio 2025 (cit. alle pp. 63–66, 73).
- [81] NIST. *Digital Identity Guidelines: Authentication and Lifecycle Management (SP 800-63B-3)*. Rapp. tecn. revisioni successive fino al 2024. National Institute of Standards e Technology, 2017 (cit. alle pp. 63–65, 73).
- [82] NIST. «Verifiers SHALL store memorized secrets salted and hashed using a suitable one-way key derivation function». In: *NIST SP 800-63B* (2017) (cit. alle pp. 63, 64, 73).
- [83] Himanshu Kumar, Sudhanshu Kumar, Remya Joseph, Dhananjay Kumar, Sunil Kumar Shrinarayan Singh, Ajay Kumar e Praveen Kumar. «Rainbow table to crack password using MD5 hashing algorithm». In: *2013 IEEE conference on information & communication technologies*. IEEE. 2013, pp. 433–439 (cit. alle pp. 64, 65).
- [84] National Institute of Standards e Technology. *Secure Hash Standard (SHS) FIPS PUB 180-4*. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>. 2015 (cit. alle pp. 64–66).

- [85] Alfred J. Menezes, Paul C. Van Oorschot e Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996 (cit. alle pp. 65, 66).
- [86] Niels Ferguson, Bruce Schneier e Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley, 2010 (cit. alle pp. 65, 66).
- [87] National Institute of Standards e Technology. *NIST recommends discontinuing use of SHA-1*. <https://csrc.nist.gov/news/2022/nist-recommends-discontinuing-sha-1>. 2022 (cit. a p. 66).
- [88] Xiaoyun Wang, Hongbo Yu et al. «How to Break MD5 and Other Hash Functions». In: *EUROCRYPT 2005* (2005), pp. 19–35 (cit. a p. 66).
- [89] OWASP Foundation. *Password Cracking Tools*. https://owasp.org/www-community/attacks/Password_cracking. 2023 (cit. a p. 71).
- [90] Hashcat Developers. *Hashcat Documentation*. <https://hashcat.net/wiki/>. 2024 (cit. a p. 71).
- [91] Massimiliano Montoro. *Cain and Abel*. <https://sectools.org/tool/cain-and-abel/>. 2014 (cit. a p. 71).
- [92] Objectif Sécurité. *Ophcrack*. <https://ophcrack.sourceforge.net/>. 2022 (cit. a p. 71).
- [93] OWASP Foundation. *OWASP Cryptographic Storage Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html. Accessed: 2025-07-05. 2023 (cit. a p. 72).
- [94] Morris Dworkin. *Recommendation for Applications Using Approved Hash Algorithms (Revision 1)*. Rapp. tecn. Special Publication 800-107 Revision 1. National Institute of Standards e Technology (NIST), 2012. DOI: 10.6028/NIST.SP.800-107r1. URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-107r1.pdf> (cit. a p. 72).
- [95] OWASP. *Secure Deployment Cheat Sheet*. Accessed: 2025-07-05. 2021. URL: https://cheatsheetseries.owasp.org/cheatsheets/Secure_Deployment_Cheat_Sheet.html (cit. a p. 72).

- [96] National Institute of Standards e Technology. *Secure Software Development Framework (SSDF) Version 1.1 (SP 800-218)*. Rapp. tecn. Accessed: 2025-07-05. NIST, 2022. URL: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf> (cit. a p. 72).
- [97] OWASP. *Secrets Management Cheat Sheet*. Accessed: 2025-07-05. 2022. URL: https://cheatsheetseries.owasp.org/cheatsheets/Secret_Management_Cheat_Sheet.html (cit. a p. 73).
- [98] William Casey, Jeffrey Glenn, Gaurav Shah, Jarred Yurek e David Stanfield. «Capture the Flag: A Competency-Based Learning Platform for Cybersecurity Education». In: *IEEE Security & Privacy* 18.3 (2020), pp. 84–88. DOI: 10.1109/MSEC.2020.2974882 (cit. a p. 74).
- [99] National Institute of Standards e Technology. *National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework*. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-181r1.pdf>. Accessed: 2025-07-04. 2020 (cit. alle pp. 74, 75).
- [100] (ISC)². *Cybersecurity Workforce Study 2022*. <https://www.isc2.org/Research/Workforce-Study>. Accessed: 2025-07-04. 2022 (cit. a p. 75).
- [101] Jan Vykopal, Valdemar Švábenský e Ee-Chien Chang. «Benefits and Pitfalls of Using Capture the Flag Games in University Courses». In: *SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2020, pp. 752–758. DOI: 10.1145/3328778.3366893 (cit. a p. 76).
- [102] D. Meinsma, S. Barjas, M. Gilhespy e M. Björkqvist. *Effectiveness of CTF Education: Measuring the Learning Outcomes of Jeopardy-Style CTFs*. Technical Report. The Hague, Netherlands: The Hague University of Applied Sciences, 2022 (cit. a p. 76).
- [103] CTFd Team. *CTFd Documentation*. <https://docs.ctfd.io/>. Accessed: 2025-07-04. 2024 (cit. a p. 77).
- [104] CTFd Team. *CTFd API Documentation*. <https://api.ctfd.io/>. Accessed: 2025-07-04. 2024 (cit. a p. 77).