

PROGETTO

VAULT



Cliente:
Compagnia Theta

L'Azienda



Chi siamo

Fondata nel 2011 ad Edimburgo, in Scozia, WallaceVault Cybersecurity è oggi uno dei principali player europei nel settore della cybersecurity avanzata. Specializzata nella protezione di infrastrutture critiche, grandi imprese e istituzioni governative.

Cosa facciamo

Offriamo una piattaforma di sicurezza integrata, in grado di identificare, prevenire e neutralizzare minacce informatiche complesse prima che causino danni. I nostri servizi includono:

- Managed Detection & Response (MDR) 24/7
- Analisi delle vulnerabilità e penetration testing
- Protezione di reti OT/ICS per l'industria 4.0
- Formazione e simulazioni di attacco su misura (Red/Blue team)
- Consulenza per compliance normativa (GDPR, NIS2, ISO 27001)

Dove operiamo

Con oltre 600 dipendenti in 5 sedi strategiche (Edimburgo, Milano, Berlino, Dubai e Singapore), assiste oltre 300 clienti tra banche, utility, ospedali e aziende tech, offrendo copertura globale e supporto localizzato.

Missione

Proteggere l'innovazione e i dati delle organizzazioni più esposte, anticipando le minacce di domani attraverso tecnologia, talento e fiducia.



Team



SIMONE MARCHICA



SIMONE STARVAGGI



GIANLUCA AGOSTINI



ENRICO MEDDA



CHIARA RICCI



GIANLUCA CAMISCIA

Proposta



Descrizione progetto

Sviluppare un preventivo di spesa e un progetto di rete per l'infrastruttura.

Definizione del problema

La struttura oggetto del progetto presenta attualmente una rete informatica insufficiente a soddisfare le esigenze operative. Si richiede lo sviluppo di una nuova infrastruttura di rete mista (LAN e Wi-Fi), in grado di garantire la connessione simultanea di [numero] utenti, con adeguata copertura wireless, sicurezza dei dati, gestione centralizzata delle risorse e predisposizione per espansioni future.

Soluzione proposta

Nel progetto è prevista l'installazione di 20 computer per ciascun piano, per un totale complessivo di 120 postazioni di lavoro. A supporto dell'infrastruttura informatica e per garantire un adeguato livello di sicurezza della rete, verranno integrati diversi dispositivi fondamentali. Sarà presente un Web Server dedicato alla gestione dei servizi online e delle applicazioni interne, accompagnato da un Firewall perimetrale incaricato di monitorare e filtrare il traffico in ingresso e in uscita, proteggendo così la rete da potenziali minacce esterne.

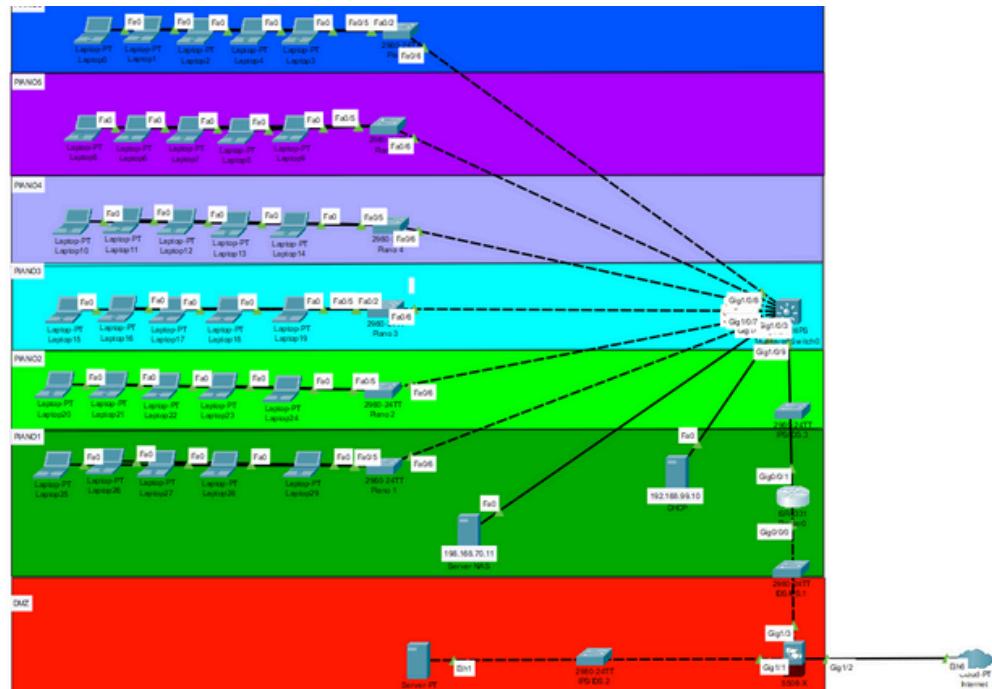
Per quanto riguarda l'archiviazione e la condivisione dei dati, verrà installato un sistema NAS (Network Attached Storage), che permetterà di centralizzare i file in modo sicuro e accessibile agli utenti autorizzati. Inoltre, la rete sarà monitorata e protetta da tre dispositivi IDS/IPS (Intrusion Detection System / Intrusion Prevention System), che avranno il compito di rilevare tempestivamente eventuali attività sospette o tentativi di intrusione e, se necessario, intervenire per bloccarli.

Vantaggi del progetto

Aumento dell'efficienza operativa Sicurezza avanzata dei dati e della rete Centralizzazione e condivisione efficiente dei dati Servizi web interni e accesso remoto controllato Infrastruttura scalabile e pronta per il futuro Maggior controllo e governance IT.



Piano d'azione



L'architettura di rete è stata concepita su un modello gerarchico, che garantisce scalabilità, sicurezza e facilità di gestione. La rete è logicamente divisa in diverse VLAN (Virtual Local Area Network) per isolare il traffico e migliorare le prestazioni.

Prima di tutto i pc saranno configurati mediante un Server DHCP Dedicato che assegna automaticamente gli indirizzi IP ai dispositivi, semplificando così la gestione della rete. Per la configurazione del Server DHCP, come è mostrato in figura, si assegnano un pool di indirizzi IP, suddivisi per piano (P1, P2, P3, P4).

DHCP								
Pool Name	Default Gateway	DNS Server	Start IP Address	Subnet Mask	Max User	TFTP Server	VLC Address	
NAS	192.168.70.1	0.0.0.0	192.168.70.17	255.255.255.0	239	0.0.0.0	0.0.0.0	
serverPool	0.0.0.0	0.0.0.0	192.168.99.11	255.255.255.0	245	0.0.0.0	0.0.0.0	
P6	192.168.60.1	0.0.0.0	192.168.60.16	255.255.255.0	240	0.0.0.0	0.0.0.0	
P5	192.168.50.1	0.0.0.0	192.168.50.15	255.255.255.0	241	0.0.0.0	0.0.0.0	
P4	192.168.40.1	0.0.0.0	192.168.40.14	255.255.255.0	242	0.0.0.0	0.0.0.0	
P3	192.168.30.1	0.0.0.0	192.168.30.13	255.255.255.0	243	0.0.0.0	0.0.0.0	
P2	192.168.20.1	0.0.0.0	192.168.20.12	255.255.255.0	244	0.0.0.0	0.0.0.0	
P1	192.168.10.1	0.0.0.0	192.168.10.11	255.255.255.0	245	0.0.0.0	0.0.0.0	



I PC verranno connessi agli switch che saranno installati in ogni piano. L'interfaccia dello switch è configurata in trunk, così che i computer possono dialogare tra loro, nonché scambiarsi informazioni tra i diversi piani dell'edificio. Gli switch verranno collegati al Multilayer Switch centrale, tramite collegamenti ad alta velocità.

Il Multilayer switch verrà configurato in trunk, dando la possibilità ai piani di poter comunicare tra di loro e con il server NAS e vi sarà fatta una configurazione dell'interfaccia VLAN tramite la cli.

Il Multilayer Switch verrà successivamente, connesso al firewall, che a sua volta si connette a Internet, formando un percorso logico e sicuro per il traffico di rete.

Il Firewall Perimetrale, posto al confine della rete, è responsabile della segregazione del traffico tra le zone di sicurezza (rete interna, DMZ, Internet) e dell'applicazione delle policy.

Nella pagina successiva verrà approfondita la struttura del Firewall.

Firewall

I PC verranno connessi agli switch che saranno installati in ogni piano. L'interfaccia dello switch è configurata in trunk, così che i computer possono dialogare tra loro, nonché scambiarsi informazioni tra i diversi piani dell'edificio. Gli switch verranno collegati al Multilayer Switch centrale, tramite collegamenti ad alta velocità.

Il Multilayer switch verrà configurato in trunk, dando la possibilità ai piani di poter comunicare tra di loro e con il server NAS e vi sarà fatta una configurazione dell'interfaccia VLAN tramite la cli.

Il Multilayer Switch verrà successivamente, connesso al firewall, che a sua volta si connette a Internet, formando un percorso logico e sicuro per il traffico di rete.

Il Firewall Perimetrale, posto al confine della rete, è responsabile della segregazione del traffico tra le zone di sicurezza (rete interna, DMZ, Internet) e dell'applicazione delle policy.

LAN

Le regole di firewall sulla LAN controllano quali tipi di traffico sono consentiti o bloccati in uscita dalla rete locale, tra segmenti specifici all'interno della LAN o verso destinazioni raggiungibili dalla LAN. Vengono gestite dall'alto verso il basso e la prima regola che corrisponde al traffico viene applicata.

Rules (Drag to Change Order)								
	States	Protocol	Source	Port	Destination	Port	Gateway	
<input checked="" type="checkbox"/>	✓ 1/217 KiB	*	*	*	LAN Address	80	*	
<input type="checkbox"/>	✓ 0/0 B	IPv4 TCP	LAN subnets	*	192.168.73.74 (HTTP)	80	*	
<input type="checkbox"/>	✗ 0/0 B	IPv4 *	LAN subnets	*	WEB SERVER subnets	*	*	
<input type="checkbox"/>	✓ 5/255 KiB	IPv4 *	LAN subnets	*	*	*	*	



Regola 1: Anti-Lockout Rule

Azione: Permessa.

Protocollo: Qualsiasi.

Sorgente: Qualsiasi.

Destinazione: L'indirizzo IP del firewall stesso sull'interfaccia LAN (es. 192.168.X.1).

Porta Destinazione: 80 (HTTP).

Scopo: Questa regola di sicurezza cruciale impedisce all'amministratore di perdere l'accesso

alla gestione del firewall tramite l'interfaccia web (HTTP) dalla LAN. Indipendentemente da altre regole più restrittive, l'accesso alla configurazione del firewall è sempre garantito, assicurando che non si venga "bloccati fuori" dal sistema.

✓	1/217	*	*	*	LAN Address	80	*	*
	KiB							

Regola 2: Consenti http da Lan a Web Server

Azione: Permessa.

Protocollo: TCP (IPv4).

Sorgente: LAN subnets (ovvero i dispositivi nei piani 1, 2, ecc.).

- Destinazione: 192.168.73.74 (Web Server).

Porta Destinazione: 80 (HTTP).

Scopo: Questa regola abilita specificamente i dispositivi connessi nei vari piani (le sottoreti LAN) a raggiungere il Web Server nella DMZ tramite il protocollo HTTP.

<input type="checkbox"/>	<input checked="" type="checkbox"/>	0/0 B	IPv4	LAN subnets	*	192.168.73.74	80	*
			TCP				(HTTP)	



Regola 3: Blocca LAN verso Web Server

Azione: Bloccata.

Protocollo: Qualsiasi (IPv4 *).

Sorgente: LAN subnets (dispositivi nei piani).

Destinazione: WEB SERVER subnets (sottoreti nella DMZ).

Scopo: Questa è una regola di blocco che nega tutto il traffico IPv4 proveniente dai piani LAN e diretto verso le sottoreti dei server web nella DMZ.

<input type="checkbox"/>		0/0 B	IPv4 *	LAN subnets	*	WEB SERVER subnets	*	*
--------------------------	---	-------	--------	-------------	---	--------------------	---	---

Regola 4: Consenti le connessioni LAN

Azione: Permessa.

Protocollo: Qualsiasi (IPv4 *).

Sorgente: LAN subnets (dispositivi nei piani).

Destinazione: Qualsiasi (*).

Scopo: Questa è una regola "catch-all" (cattura tutto). Il suo scopo è permettere tutto il traffico rimanente dalle sottoreti LAN verso qualsiasi altra destinazione non bloccata dalle regole precedenti.

È la regola di "permesso generale" per il traffico non altrimenti specificato o bloccato.

<input type="checkbox"/>		5/255	IPv4 *	LAN subnets	*	*	*	*
		KiB						

WAN

Le regole firewall sull'interfaccia WAN hanno il compito primario di proteggere la rete interna dagli accessi non autorizzati provenienti da Internet e di definire quali servizi interni possono essere resi disponibili all'esterno. Vengono valutate in ordine, e la prima regola che corrisponde al traffico viene applicata.

Al suo interno sono presenti anche due regole di NAT, che permette a molti dispositivi privati di connettersi ad Internet utilizzando un solo indirizzo pubblico instradandoli attraverso i protocolli HTTP e HTTPS.

Infatti, ogni volta che un utente esterno tenta di connettersi all'indirizzo IP pubblico del firewall sulle porte 80 o 443, il firewall intercetta la richiesta e la inoltra al web server situato all'interno della DMZ.

Ciò rende il Web Server accessibile da Internet per la navigazione sia non sicura (HTTP) che sicura (HTTPS).

	STATE	PROTOCOL	SOURCE	PORT	DESTINATION	PORT	GATEWAY
	0/0 B	*	RFC 1918 networks	*	*	*	*
	0/0 B	*	Reserved Not assigned by IANA	*	*	*	*
<input type="checkbox"/>		0/0 B	IPv4 TCP	*	*	192.168.73.74	80 (HTTP) *
<input type="checkbox"/>		0/0 B	IPv4 TCP	*	*	192.168.73.74	443 (HTTPS) *

Regola 1: Block private networks

Azione: Bloccata (X rossa)

Protocollo: Qualsiasi (*)

Sorgente: RFC 1918 networks (reti private come 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)

Destinazione: Qualsiasi (*)

Scopo: Questa è una regola di sicurezza fondamentale. Blocca qualsiasi traffico in ingresso dalla WAN che erroneamente (o maliziosamente) riporti un indirizzo IP sorgente appartenente a una rete privata (RFC 1918). Dato che tali indirizzi non dovrebbero mai essere instradati pubblicamente su Internet, questa regola previene attacchi di spoofing (falsificazione dell'indirizzo sorgente) o configurazioni errate che potrebbero tentare di instradare traffico privato sulla WAN.

 0/0 B	*	RFC 1918 networks	*	*	*	*
---	---	-------------------	---	---	---	---

Regola 2: Block bogon networks

Azione: Bloccata

Protocollo: Qualsiasi (*)

Sorgente: Reserved Not assigned by IANA (reti "bogon" ovvero indirizzi IP non ancora

assegnati o riservati a scopi speciali dall'IANA/ICANN)

Destinazione: Qualsiasi (*)

Scopo: Simile alla precedente, questa regola di sicurezza blocca il traffico in ingresso dalla WAN che utilizza indirizzi IP sorgente non validi o non assegnati. Questo aiuta a filtrare il traffico malevolo o malformato prima che raggiunga le reti interne.

 0/0 B	*	Reserved Not assigned by IANA	*	*	*	*
---	---	-------------------------------	---	---	---	---

Regola 3: NAT Redirect HTTP a Web Server

Azione: Permessà e applica NAT Redirect (Port Forwarding).

Protocollo: TCP (IPv4).

Sorgente: Qualsiasi (*)

Porta Sorgente: Qualsiasi (*).

Destinazione: Qualsiasi (*)

Porta Destinazione: 80 (HTTP).

Funzionamento NAT: Qualsiasi richiesta TCP in arrivo sull'indirizzo IP pubblico del firewall (connesso a Internet) sulla porta 80 viene reindirizzata internamente all'indirizzo IP privato 192.168.73.74 (Web Server) sempre sulla porta 80.

Scopo: Permettere ai client su Internet di accedere al Web Server in DMZ tramite il protocollo HTTP.

<input type="checkbox"/>		0/0 B	IPv4	*	*	192.168.73.74	80 (HTTP)	*
--------------------------	--	-------	------	---	---	---------------	-----------	---

Regola 4: NAT Redirect HTTPS a Web Server

Azione: Permessà e applica NAT Redirect (Port Forwarding).

Protocollo: TCP (IPv4).

Sorgente: Qualsiasi (*)

Porta Sorgente: Qualsiasi (*)

Destinazione: Qualsiasi (*)

Porta Destinazione: 443 (HTTPS).

Funzionamento NAT: Qualsiasi richiesta TCP in arrivo sull'indirizzo IP pubblico del firewall (connesso a Internet) sulla porta 443 viene reindirizzata internamente all'indirizzo IP privato 192.168.73.74 (il Web Server nella DMZ), sempre sulla porta 443.

Scopo: Permettere ai client su Internet di accedere al Web Server 192.168.73.74 in DMZ tramite il protocollo HTTPS, garantendo una comunicazione sicura.

<input type="checkbox"/>		0/0 B	IPv4	*	*	192.168.73.74	443	*
--------------------------	--	-------	------	---	---	---------------	-----	---

Vi sarà un Web Server ospitato nella DMZ ed isolato dalla rete interna per una maggiore sicurezza.

Il NAS fornirà spazio di archiviazione centralizzato, accessibile da tutta la rete interna.

Infine i tre sistemi IDS/IPS, saranno allocati in punti strategici (DMZ, perimetro di rete interno, e traffico inter-VLAN) per monitorare e prevenire intrusioni e attività malevole.

Un Intrusion Prevention System (IPS) è uno strumento di sicurezza di rete progettato per monitorare il traffico di rete alla ricerca di attività malevole, modelli di attacco noti, violazioni delle policy o anomalie.

Il software utilizzato nel progetto è Suricata, uno strumento open source molto potente e versatile.

Gli IPS verranno implementati nel seguente modo:

- Sulla WAN per proteggere dagli attacchi esterni.
- Sulla LAN per rilevare e prevenire minacce interne o da host compromessi.
- Sul Web Server (DMZ) per fornire una protezione mirata per i servizi più esposti.

Interface Settings Overview					
Interface	Suricata Status	Pattern Match	Blocking Mode	Description	Actions
WAN (vtnet0)	✓	AUTO	LEGACY MODE	WAN	
LAN (vtnet1)	✓	AUTO	LEGACY MODE	LAN	
WEBSERVER (em0)	✓	AUTO	LEGACY MODE	METASPLOITABLE	

Il modo in cui l'architettura è stata proposta, permetterà di poter aggiungere nuovi piani o utenti senza ristrutturare l'intera rete. Una sicurezza migliore, poichè le VLAN isolano il traffico, contenendo potenziali attacchi, il Web Server posizionata all'interno della DMZ, si troverà in un'area isolata, riducendo il rischio per la rete interna e di conseguenza IDS/IPS, permetteranno di rilevare e prevenire attacchi in punti critici della rete.

La gestibilità sarà più efficiente grazie all'uso DHCP. L'uso di un Multilayer Switch per il routing inter-VLAN garantisce velocità e efficienza, riducendo i colli di bottiglia.

La segmentazione e i collegamenti ad alta velocità migliorano le prestazioni complessive.



Testing della Rete

STRUTTURA DEL PROGETTO

Il progetto è strutturato in modo modulare e orientato agli oggetti. Il punto di ingresso principale è rappresentato dal file `main.py`, che gestisce l'interazione con l'utente tramite un menu testuale interattivo visualizzato nel terminale. Il menu consente la selezione di diverse funzionalità operative, ognuna delle quali è associata a una funzione definita all'interno dello stesso file `main.py`.

Le singole funzioni del menu, una volta selezionate, istanziano oggetti di classi definite in moduli separati e ne invocano i metodi specifici, passando i parametri necessari per l'elaborazione. Le classi e i file principali coinvolti sono i seguenti:

MAIN

Questo script Python fornisce un menu interattivo a riga di comando (CLI) per eseguire test di sicurezza e diagnostica di rete, tra cui:

- Scansione delle porte TCP su un host specifico
- Sniffing di pacchetti su una porta specifica
- Scraping di risorse web non PHP
- Test dei metodi HTTP abilitati (con l'header OPTIONS)
- Test funzionale di singoli metodi HTTP disponibili (GET, POST, ecc.)

Il programma è pensato per essere usato in contesto di analisi di rete locale o web, e

può essere utile per attività didattiche, penetration testing controllati, o esercitazioni su ambienti simulati.

Librerie utilizzate

- **questionary**, una libreria che serve per creare menu e input interattivi in console, con interfaccia utente semplice e intuitiva (domande a selezione e input testo).

Viene usata per chiedere all'utente:

- IP
- range di porte
- URL e Path
- scelta da elenco (es. risorse trovate)

- **port_scanner** (modulo locale) iol quale contiene la classe PortScanner con due metodi:

- scan(ip, porta_iniziale, porta_finale): esegue scansione TCP
- socket_sniffer(ip, porta, buffer): avvia sniffing di pacchetti su una porta (il buffer serve per sapere quanti dati voglio carpire da un pacchetto).

- **http_tester** (modulo locale)

Contiene la classe HttpMethodTester per:

Ottenere i metodi HTTP supportati da una risorsa via OPTIONS

Eseguire richieste con uno specifico metodo HTTP (GET, POST, ecc.)

- **scraper** (modulo locale)

Contiene la classe ScraperScanner, presumibilmente con:

scraping_pagina(path) → fa scraping HTML su una pagina per estrarre link o risorse, filtrando quelli non .php.

A seguire il codice di riferimento.

Main

```
# Funzione principale che gestisce il menu interattivo
def main():
    while True:
        # Menu delle operazioni disponibili
        scelta = questionary.select(
            "Cosa vuoi fare?",
            choices=[
                "1) Scansione porte TCP",
                "2) sniffing(intercettazione) della porta",
                "3) Scraping risorse non php",
                "4) Test metodi HTTP (OPTIONS)",
                "5) Esegui singolo metodo HTTP",
                "6) Esci"
            ]).ask()
        # esegue la funzione corrispondente alla scelta fatta nel menu delle opzioni
        if scelta.startswith("1"):
            porta_scanner()
        elif scelta.startswith("2"):
            sniffer()
        elif scelta.startswith("3"):
            scraper()
        elif scelta.startswith("4"):
            http_tester_metodi()
        elif scelta.startswith("5"):
            http_tester_completo()
        elif scelta.startswith("6"):
            print("Arrivederci! ")
            break
        else:
            print("scelta selezionata inesistente.")
            break

# Avvia il programma se eseguito direttamente sennò è abilitato pure per essere avviato da terminale
if __name__ == "__main__":
    main()
```



PORT SCANNER

Funzione del file

Questo modulo definisce la classe PortScanner, che fornisce due funzionalità principali:

- Scansione delle porte TCP di un indirizzo IP, per verificarne l'apertura o la chiusura.
- Sniffing di pacchetti TCP in entrata su una porta specifica, filtrando solo quelli provenienti da un IP target.

Librerie utilizzate

- **socket** - importata come so, una libreria standard per la comunicazione di rete in Python.
 - **Utilizzo**
 - Creare socket TCP (SOCK_STREAM) e RAW (SOCK_RAW)
 - Eseguire connessioni alle porte (connect_ex)
 - Ricevere pacchetti in ascolto (recvfrom)
 - Filtrare e analizzare il traffico in entrata
- **datetime** - importata come dt, una libreria standard per gestire date e orari.
 - **Utilizzo**
 - Stampare il timestamp quando viene attivato lo sniffer.

Conclusione

Questo file fornisce strumenti essenziali per testare e monitorare il comportamento di una rete TCP/IP.

A seguire il codice di riferimento.

Port Scanner

```
import socket as so
import datetime as dt

class PortScanner:
    def __init__(self):
        # imposto l'ip da scansionare a none (nessuno)
        self.ip_scansionato = None

    def scan(self, ip, inizio, fine):
        #ip scansionati è una variabile che viene usata per salvare l'indirizzo ip attualmente scansionato per tenere traccia degli ip.
        self.ip_scansionato = ip
        # crea una lista
        risultati = []
        # ciclo sul range da porta a porta che mando al metodo
        for porta in range(inizio, fine):
            # creo una variabile che funzionerà come canale di comunicazione per collegarsi ad un server via rete
            # gli specifico che ciò che sto passando sono IPV4 con AF_INET e che sto usando la connessione TCP e non UDP con SOCK_STREAM
            sock = so.socket(so.AF_INET, so.SOCK_STREAM)
            # imposto un limite di tempo in cui voglio che mi arrivi una risposta nel canale se entro quel limite non mi arriva
            # una risposta dalla porta la considero chiusa o irraggiungibile
            sock.settimeout(5) # 5 è il limite massimo di secondi che attenderò
            # questa variabile prova a connettersi all'ip e alla porta se la connessione va a buon fine il risultato sarà 0
            # se non va a buon fine qualsiasi altro numero
            risultato = sock.connect_ex((ip, porta))
            # a questa variabile do la stringa aperta se il risultato della variabile che tenta la connessione è 0 chiusa se è un altro numero
            stato = "aperta" if risultato == 0 else "chiusa"
            if stato == "aperta":
                #aggiungo alla lista la porta e lo stato della connessione (risposta)
                risultati.append({"porta": porta, "stato": stato})
            # chiudo il canale precedentemente aperto nella variabile sock.
            sock.close()
        return risultati

    # Metodo per sniffare pacchetti provenienti da un IP specifico su una porta specifica
    def socket_sniffer(self, ip_target, porta, dimensioni_buffer):
        try:
            #crea un socket RAW per ricevere pacchetti TCP a basso livello (quindi in byte)
            sniffer = so.socket(so.AF_INET, so.SOCK_RAW, so.IPPROTO_TCP)
            # effettua il bind del socket a tutte le interfacce locali sulla porta specificata
            sniffer.bind(("0.0.0.0", porta))
            # include anche l'header IP nei pacchetti ricevuti
            sniffer.setsockopt(so.IPPROTO_IP, so.IP_HDRINCL, 1)
            # stampa a console che lo sniffer è attivo
            print(f"[{dt.datetime.now()}] sniffer attivato sulla porta {porta}")
            # inizializza una lista
            risultato = []
            #inizializza un contatore di pacchetti catturati
            indice = 0
            max_pacchetti = 10
            # continua a ricevere pacchetti finchè non ne riceve 10 dal target
            while indice < max_pacchetti:
                # riceve un pacchetto (dati) e l' indirizzo sorgente (IP, Porta)
                dati, indirizzo = sniffer.recvfrom(dimensioni_buffer)
                # estrae l'indirizzo IP sorgente dal pacchetto
                ip_sorgente = indirizzo[0]
                # Filtra i pacchetti: accetta solo quelli provenienti dall'IP target
                if ip_sorgente == ip_target:
                    # Aggiunge alla lista un dizionario con IP, porta e primi 100 byte dei dati
                    risultato.append({
                        "ip": ip_sorgente,
                        "porta": indirizzo[1],
                        "dati": dati[:100]
                    })
                # incrementa il contatore dei pacchetti validi ricevuti
                indice += 1
            # se il programma non ha i permessi sufficienti per usare i raw socket (su linux usa sudo) va in errore
        except PermissionError:
            risultato = "Permesso negato: esegui con livello di autorizzazione superiore"
        # Gestione di eventuali altri errori
        except Exception as e:
            risultato = f"Errore nello sniffing del socket: {e}"
        return risultato
```

http_tester

Introduzione

I metodi HTTP sono comandi che il client (come un browser o un'app) utilizza per comunicare con un server. Seguono un modello richiesta-risposta: il client invia un'azione, il server la elabora e risponde. In pratica, i metodi HTTP definiscono che tipo di operazione il client vuole eseguire su una risorsa (es. leggere, creare, modificare, cancellare). Un esempio semplice: Sono come i verbi in una lingua. Senza di essi, il server non saprebbe che azione deve compiere.

Rischi di Sicurezza

Prendiamo come esempio un'applicazione come Instagram. Anche se sembra solo un'app sul telefono, in realtà comunica con un server tramite API REST, utilizzando i metodi HTTP per scambiare informazioni tra client e server. Ogni applicazione ha bisogno di: Un backend (server) Degli endpoint: indirizzi specifici che espongono risorse o servizi.

Il problema nasce quando questi endpoint non verificano correttamente chi sta effettuando la richiesta (autenticazione) e cosa è autorizzato a fare (autorizzazione). In questo scenario, un attaccante potrebbe:

- Accedere a dati sensibili (email, messaggi privati, hash delle password)
- Modificare o eliminare risorse di altri utenti
- Creare contenuti non autorizzati.

Queste vulnerabilità rientrano nella categoria Broken Access Control e possono portare a: Furti di dati personali, Estorsioni, ricatti e Vendita di informazioni nel dark web.

Funzione del file

Il file definisce la classe `HTTPMethodTester`, uno strumento per:

- Verificare quali metodi HTTP (GET, POST, PUT, DELETE, ecc.) sono abilitati su un endpoint tramite una richiesta OPTIONS.
- Eseguire richieste reali con questi metodi per testare la risposta del server.

Il file è pensato per essere integrato in un tool di network/web testing o in un laboratorio di penetration testing etico, per controllare le superfici HTTP esposte da un'applicazione web.

Libreria utilizzata

- **httpx**, una libreria moderna e potente per fare richieste HTTP in Python.
 - **httpx.get(...)**, **httpx.post(...)**, **httpx.options(...)**, ecc. → usati per inviare le varie richieste. Offre gestione degli errori, timeout, richieste asincrone, ed è compatibile con requests ma più performante.

Funzionamento generale

L'intera classe `HTTPMethodTester` fornisce strumenti per indagare la superficie HTTP esposta da un server.

Supporta metodi standard HTTP (GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE).

Ha gestione base degli errori, restituendo messaggi di fallback se la richiesta fallisce.

Conclusione

Il file `http_tester` serve per:

- Elencare i metodi HTTP supportati da un endpoint.
- Eseguire test reali su ogni metodo (GET, POST, PUT, ecc.).
- Verificare la risposta del server e diagnosticare eventuali aperture rischiose o malfunzionamenti.

A seguire il codice di riferimento.

HTTP Tester

```
import httpx
# definisce la classe HTTPMethodTester
class HTTPMethodTester:
    # il costruttore che salva l'url base che verrà eventualmente usato per comporre i link da testare ( non serve attualmente poichè i link disponibili vengono
    def __init__(self, url_base):
        self.url_base = url_base

    # Effettua una richiesta OPTIONS al link che gli viene passato dal main
    def metodi_permessi(self, link):
        try:
            #url = f"{self.url_base}{path}"
            response = httpx.options(link) # effettua la richiesta OPTIONS
            risposta = response.headers.get("Allow", "") # estrae el'header "Allow" con i metodi permessi
            risultato = [m.strip().upper() for m in risposta.split(",")] # restituisce una lista di metodi in Maiuscolo
        except Exception as e:
            risultato = f"Errore nella richiesta OPTIONS: {e}"
        return risultato

    # esegue una chiamata HTTP GET sul link indicato e restituisce un codice di stato e il contenuto della risposta
    def get(self, link):
        try:
            response = httpx.get(link)
            risultato = f"Lo stato della risposta è -> {response.status_code}\n e il suo contenuto è -> {response.text}"
        except Exception as e:
            risultato = f"Errore nella richiesta GET: {e}"
        return risultato

    # esegue una chiamata HTTP POST non invii dati da inserire quindi questo restituisce status code e testo come il get
    def post(self, link):
        try:
            response = httpx.post(link)
            risultato = f"Lo stato della risposta è -> {response.status_code}\n e il suo contenuto è -> {response.text}"
        except Exception as e:
            risultato = f"Errore nella richiesta POST: {e}"
        return risultato

    # effettua una chiamata put in teoria con dati hardcoddati (DATA)
    # Effettua una chiamata HTTP HEAD, che restituisce solo gli header e non il contenuto
    def head(self, link):
        try:
            response = httpx.head(link)# manda una richiesta HEAD all'link
            risultato = (f"Lo stato della risposta è -> {response.status_code}\n" # costruisce una stringa risultato formattata per contenere: status code.
                        f"Header ricevuti:\n"
                        + "\n".join([f"\t{k}: {v}" for k, v in response.headers.items()])) # tutte le coppie chiave valore degli header, vengono prese e mostrate a
            return risultato
        except Exception as e:
            risultato = f"Errore nella richiesta HEAD: {e}"
        return risultato

    # esegue una chiamata HTTP TRACE su un link (Trace usata solitamente per il debug) restituisce una copia della richiesta inviata
    def trace(self, link):
        try:
            response = httpx.request("trace", link)
            risultato = f"Lo stato della risposta è -> {response.status_code}\n e il suo contenuto è -> {response.text}"
            return risultato
        except Exception as e:
            risultato = f"Errore nella richiesta TRACE: {e}"
            return risultato

    # Un options aggiuntivo da richiamare quando sull' options superiore esce un ulteriore options da poter eseguire su un url.
    def options(self, link):
        try:
            response = httpx.options(link)
            risposta = response.headers.get("Allow", "")
            risultato = [m.strip().upper() for m in risposta.split(",")]
            return risultato
        except Exception as e:
            risultato = f"Errore nella richiesta OPTIONS: {e}"
            return risultato
```

SCRAPER

Funzione del file

Il file definisce la classe ScraperScanner, progettata per eseguire scraping HTML di una pagina web e restituire l'elenco di risorse collegate (immagini, script, fogli di stile, iframe) escludendo i file .php.

Utilizzo

Analizzare una pagina web (locale o remota)

Trovare link a risorse statiche

Filtrare solo quelle non PHP, potenzialmente utili per analisi di superficie esposta o mappatura contenuti

Librerie utilizzate

- **httpx**, una libreria per effettuare richieste HTTP moderne (alternativa più veloce e asincrona di requests).
 - **httpx.get(url)** → esegue una richiesta GET e restituisce la risposta della pagina HTML.
- **bs4** (BeautifulSoup), utilizzata per il parsing dell'HTML, ovvero per trasformare il codice HTML in una struttura navigabile a oggetti.
 - **BeautifulSoup(response.text, "html.parser")** → analizza il contenuto della risposta come codice HTML.
 - **soup.find_all([...])** → estrae tutti i tag HTML di tipo img, script, link, iframe.
- **urllib.parse(urljoin)**, serve a completare URL parziali (relativi) unendoli all'URL di partenza.
 - **urljoin(base_url, path)** → genera un URL assoluto anche se quello nel tag è relativo.

Conclusione

Questo modulo è utile per fare riconoscione in un contesto di security testing o sviluppo web.

Evitare i file dinamici .php, concentrandosi solo su immagini, script, CSS, ecc.

A seguire il codice di riferimento.

Scraper

```
class ScraperScanner:
    def __init__(self, url_base):
        self.url_base = url_base

    # scraping della pagina attraverso BeautifulSoup
    def scraping_pagina(self, path):
        url = f'{self.url_base}{path}'
        # esegue una richiesta get per farsi restituire in risposta html
        response = httpx.get(url)
        # analizza (fa il parsing (prende la stringa in questo caso tutto l'html e lo trasforma in una struttura a oggetti)) del contenuto HTML della risposta
        # usando beautifulsoup con parser html.parser
        soup = BeautifulSoup(response.text, "html.parser")
        # cerca tutti i tag html img, script , link, iframe che di solito contengono riferimenti a file esterni
        tags = soup.find_all(["img", "script", "link", "iframe"])
        #inizializza una lista
        percorsi = []
        # cicla la lista taga (quella con i tag con riferimenti a file esterni)
        for tag in tags:
            # se il tag è img, script, iframe ne prende il valore dell'attributo src (dove si trova la risposta)
            if tag.name in ["img", "script", "iframe"]:
                src = tag.get("src")
                # se src esiste, lo converte in un url assoluto in modo tale da far diventare i link relativi completi unenod url e src e lo aggiunge alla lista percorsi
                if src:
                    percorsi.append(urljoin(url, src))
            # se invece il tag è link prende l'attributo href dove si trovano i css
            elif tag.name == "link":
                href = tag.get("href")
                # se href esiste, lo converte in un url assoluto e lo aggiunge alla lista percorsi
                if href:
                    percorsi.append(urljoin(url, href))
        # usa set per eliminare duplicati dalla lista percorsi poi la converte di nuovo in una lista
        unici = list(set(percorsi))
        #filtra la lista: prende solo i link che non terminano con .php cosi da escludere gli script php
        risorse = [link for link in unici if not link.endswith(".php")]

    return risorse
```



Conclusioni e Raccomandazioni Future

L'architettura di rete della Compagnia Theta è robusta e ben progettata per le esigenze attuali. Le raccomandazioni future includono l'implementazione di ridondanza per i componenti critici (come il Multilayer Switch e il server DHCP) per aumentare l'affidabilità, migliorare il monitoraggio di rete per una gestione proattiva, e considerare soluzioni di gestione centralizzata per semplificare le operazioni su larga scala.



Simone Marchica



Simone Starvaggi Gianluca Agostini



Enrico Medda

Chiara Ricci



Gianluca Camiscia

Preventivo

WallaceVault

*Il Vault non è un software, è un santuario digitale impenetrabile dove la crittografia diventa un mistero.
Il suo funzionamento è segreto, la sua sicurezza è assoluta.*

Via Cuore Impavido, 18 PA5 9AA, Elderslie (Renfrewshire)
Telefono: 3887654312 Fax: wallacevault.commerciale@aruba.com

Offerta n.: 100

ID cliente: Theta

Via Turati 23

20121, Milano (MI)



QUANTITÀ	DESCRIZIONE	PREZZO UNITARIO	IMPORTO
80	MacBook M4 Pro chip 16" Apple	€ 2.999,00	€ 239.920,00
40	IMac 24" Chip M4 8GPU 8CPU 256GB SSD	€ 1.350,00	€ 54.000,00
6	Switch (2960-24TT)	€ 84,00	€ 504,00
1	Cisco Catalyst 3650 24 port	€ 9.286,00	€ 9.286,00
3	Server RackStation RS3621RPx	€ 3.416,00	€ 10.248,00
2	Firewall 5506-X ASA with FirePOWER services	€ 1.261,00	€ 2.522,00
1	Router Cisco ISR 4431	€ 359,00	€ 359,00
1	Costi implementazione*	€ 25.000,00	€ 25.000,00
			Tot. IVA incl.
			€ 341.839,00

I costi di implementazione* comprendono il Cabling & rack, UPS e alimentazione, configurazione firewall/ips, installazione e logistica + MDM Apple (Mobile device management), licenze software aggiuntivi, servizi professionali (assistenza), training e configurazione switch, documentazioni.

**GRAZIE PER AVERCI
SCELTO**