

UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

*Corso di Laurea Triennale in
Sicurezza dei Sistemi e delle Reti Informatiche*

DESIGN E SVILUPPO DI UN TOOL PER IL
VULNERABILITY ASSESSMENT DI PROTOCOLLI
AVANZATI DI RETE

RELATORE
Prof. Claudio A. ARDAGNA

CORRELATORE
Prof. Nicola BENA

ELABORATO FINALE DI
Falgiani Veronica
Matr. Nr. 21191A

ANNO ACCADEMICO 2024-2025

*Ai miei genitori, per avermi supportata e incoraggiata con amore.
A Roberto e alla sua famiglia, per avermi sempre accolto a braccia aperte.
A tutte le persone che hanno lasciato un segno in questi 3 anni.*

Indice

1	Introduzione	1
2	Stato dell'arte	3
2.1	Nessus	3
2.2	Nmap	5
2.3	Metasploit	6
3	Protocolli analizzati	9
3.1	Protocolli Internet Layer	9
3.2	Protocolli Transport Layer	10
3.3	Protocolli Application Layer	11
3.3.1	Protocolli semplici	12
3.3.2	Protocolli avanzati	19
4	Progettazione	22
4.1	Obiettivo dell'elaborato	22
4.2	Analisi delle funzionalità	23
4.2.1	Scansione dell'host	23
4.2.2	Scansione delle porte	24
4.2.3	Scansione dei protocolli e servizi	26
4.2.4	Esecuzione dei test	26
4.2.5	Stampa dei risultati	26
4.3	Protocolli	27
4.4	Tecnologie utilizzate	28
4.4.1	Python	28
4.4.2	JSON	29
5	Implementazione	30
5.1	main.py	31
5.2	Agent	32
5.2.1	host_scan.py	32

5.2.2	port_scan.py	33
5.2.3	service_scan.py	35
5.2.4	results.py	38
5.2.5	execute_tests.py	39
5.3	Utilities	45
5.3.1	parser.py	45
5.3.2	terminal_colors.py	46
5.3.3	write_results.py	47
6	Scrittura dei test	49
6.1	Struttura dei file	49
6.2	Elementi dei file	50
7	Risultati	53
7.1	Risultati intermedi e procedimento delle operazioni	54
7.2	Riconoscimento dei protocolli e servizi	56
7.3	Riconoscimento delle vulnerabilità	57
8	Conclusioni	62
8.1	Sviluppi futuri	62
8.1.1	Scrittura di test automatizzata	62
8.1.2	Ampliamento dei protocolli supportati	63
8.1.3	Scansione di più host contemporaneamente	63
8.1.4	Esecuzione automatica e periodica	63
8.1.5	Evasione dei firewall	63
Bibliografia		64
A Codice		66
A.1	main.py	66
A.2	agent/	67
A.3	utils/	81
B Esempio file di test		89

Elenco delle tabelle

1	Control Messages principali di ICMP	10
2	Comandi principali di FTP	12
3	Comandi principali di SMTP	13
4	Comandi principali di SMTP	14
5	Metodi di HTTP	15
6	Comandi POP3	17
7	Comandi IMAP	18
8	Porte standard dei protocolli	28

Elenco delle figure

1	Lista di template per scansioni nell’interfaccia web di Nessus	4
2	Pagina dei risultati contenente le vulnerabilità presenti sull’host	5
3	Risultati della scansione tramite Nmap	6
4	Esempio di esecuzione di un modulo Metasploit	7
5	Three-Way Handshake tra Client e Server	10
6	Risultati del riconoscimento dei protocolli in formato HTML	60

Capitolo 1

Introduzione

L'avvento di Internet ha portato grandi cambiamenti nella vita di tutti i giorni. Molte aziende hanno avuto la necessità di proporre i propri servizi sul web, per aumentare la portata e la comodità d'uso. Questa novità ha spinto le aziende alla creazione di grandi infrastrutture di rete per sopperire all'evoluzione digitale. Nella maggior parte dei casi una rete aziendale è composta da due elementi: una rete esterna "demilitarizzata" che offre i servizi al resto del mondo e una rete interna privata in cui sono connessi i dispositivi aziendali ed i server che forniscono dati alle applicazioni esposte all'esterno.

Queste due reti sono interconnesse tra loro, ma quella interna viene protetta da meccanismi di difesa avanzati, permettendo l'ingresso solo a chi è effettivamente autorizzato. Questo però non è sempre garantito, in quanto vulnerabilità presenti sulle macchine esterne potrebbero far evadere i controlli e concedere accessi indesiderati. Se le macchine interne dell'azienda sono a loro volta vulnerabili c'è la possibilità che un attaccante si insidi all'interno dei computer aziendali o peggio ancora acceda a database o informazioni sensibili. Se un malintenzionato riuscisse ad entrare in una rete interna vulnerabile sarebbe in grado di esfiltrare una grande mole di dati all'azienda, recando gravi danni economici ed alla reputazione.

Gli esperti di cybersicurezza sono in costante allerta per poter proteggere i sistemi di aziende e società da costanti minacce informatiche. Per fare ciò, come primo passo, è necessario verificare se i sistemi aziendali presentano versioni vulnerabili ad attacchi oppure configurazioni errate che portano a comportamenti indesiderati dei servizi. Questi controlli vengono fatti tramite un Vulnerability Assessment, una fase in cui vengono individuate le macchine che presentano o possono presentare falliche di sicurezza. Svolgere tutto questo a mano richiederebbe troppo tempo e risorse, per questo molte aziende forniscono strumenti chiamati Vulnerability Scanner che compiono questa mansione in modo automatico. I risultati degli scanner non sempre sono completi o corretti, per questo sarà necessario testare a mano i risultati ottenuti -per poter verificare la presenza o meno delle vulnerabilità descritte nei report.[1]

I vulnerability scanner sono strumenti molto complessi e in costante aggiornamento, utilizzati di solito nelle prime fasi di un Penetration Testing. Inviando pacchetti ai servizi e analizzando le risposte è possibile scoprire nuove informazioni o sottolineare la presenza di vulnerabilità. Per fare ciò devono poter riconoscere e scambiare dati con una grossa mole di protocolli e servizi, necessitando un continuo aggiornamento. Questi scanner sono molto utili per facilitare e velocizzare l'individuazione di criticità all'interno della rete.

Il seguente elaborato si occuperà dello studio e dello sviluppo di un vulnerability scanner, cercando di proporre una soluzione alternativa a quelle già presenti sul mercato. L'obiettivo è la creazione di un applicativo altamente modulare e leggero, per poter essere facilmente personalizzato e installato sui dispositivi da monitorare. Un software del genere è molto complesso da sviluppare e di conseguenza si è scelto di implementare solo le funzionalità principali, ma rendendo il tutto facilmente modificabile per agevolare l'aggiunta di nuove feature.

Il documento è suddiviso nei seguenti capitoli:

Il **Capitolo 2** fornisce una descrizione degli strumenti già presenti sul mercato e ne analizza le funzionalità principali.

Nel **Capitolo 3** vengono enunciati i principali protocolli della pila TCP/IP utilizzati all'interno del progetto, fornendone una breve descrizione.

Il **Capitolo 4** tratta l'analisi e la progettazione di un vulnerability scanner. Viene discusso innanzitutto il comportamento e le fasi di uno scanner, per poi trattare le tecnologie utilizzate.

Nel **Capitolo 5** viene trattata in dettaglio l'intera implementazione dell'applicativo, fornendo parti di codice e descrivendone il funzionamento.

Il **Capitolo 6** illustra la struttura dei file di test utilizzati dal software per poter verificare le vulnerabilità dei dispositivi.

Il **Capitolo 7** fornisce esempi sulle varie tipologie di file di risultati che vengono generati alla fine di una scansione.

Infine nel **Capitolo 8** vengono tratte le conclusioni e le criticità che potrebbero portare a sviluppi futuri dell'applicativo.

Capitolo 2

Stato dell'arte

Durante gli anni sono nati programmi di vulnerability assessment sempre più performanti e in continuo aggiornamento, per seguire l'avanzamento delle tecnologie e l'individuazione di nuove vulnerabilità. Un software molto accreditato e utilizzato commercialmente è Nessus, un vulnerability scanner che utilizza plug-in per svolgere test sui servizi in rete.

È anche necessario citare due programmi che non sono inerentemente vulnerability scanner, ma presentano funzionalità affini: Nmap e Metasploit. Il primo è uno scanner di rete in grado di riconoscere porte aperte e servizi. Utilizza degli script per individuare vulnerabilità molto semplici e informazioni specifiche sui servizi individuati. Il secondo è un programma utilizzato per svolgere penetration testing e presenta dei moduli che permettono di compiere le fasi di reconnaissance, footprinting, exploitation e post-exploitation. Le parti di questo software che sono di nostro interesse sono i moduli che svolgono scan di rete e individuazione di vulnerabilità.

Di seguito sono riportate informazioni più dettagliate riguardo ai programmi precedentemente descritti, in modo tale da fornire una visione d'insieme prima di trattare la soluzione proposta.

2.1 Nessus

Nessus¹ è un vulnerability scanner nato nel 1998 da parte di Tenable, una compagnia di cybersicurezza statunitense. Utilizza il paradigma client-server in quanto presenta un'interfaccia grafica in cui è possibile configurare la scansione e visualizzare i risultati (client), e un demone che svolge la scansione di host, porte, servizi e vulnerabilità (server).

Nell'interfaccia web è possibile selezionare dei template per svolgere scansioni di diverso genere, partendo da un semplice host discovery fino ad uno scan avanzato su servizi e configurazioni.

¹<https://www.tenable.com/products/nessus>

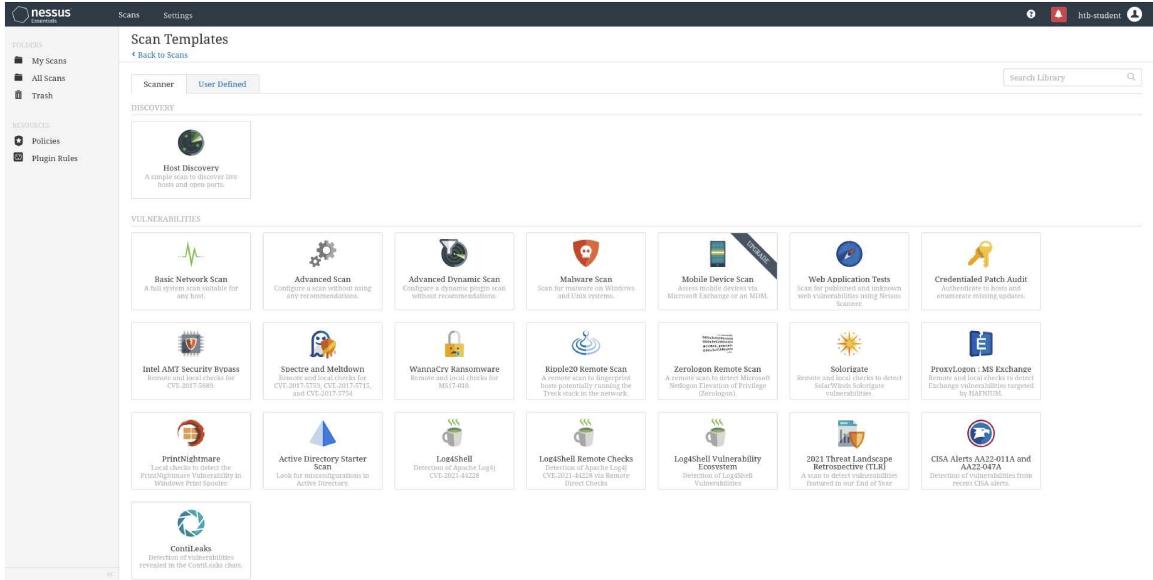


Figura 1: Lista di template per scansioni nell’interfaccia web di Nessus

Prima di iniziare lo scan vengono richieste informazioni sugli host e le modalità di svolgimento. È possibile scegliere l’aggressività dei test oppure abilitare delle opzioni per evadere i firewall e antivirus. Questo torna molto utile nelle scansioni in cui si vuole testare anche le prestazioni delle difese presenti nella rete.

Per svolgere il vulnerability scan vengono utilizzati plug-in scritti tramite NASL (Nessus Attack Scripting Language), un linguaggio di programmazione creato appositamente per Nessus, che permettono di verificare la presenza di vulnerabilità o configurazioni errate all’interno dei servizi. Questi plug-in sono in continua crescita, per rimanere aggiornati sulle innumerevoli falle di sicurezza scoperte ogni giorno.

Una volta concluse le scansioni viene stilato un report che contiene gli host individuati. Per ogni dispositivo sono presenti una lista di vulnerabilità in ordine di gravità, che a loro volta contengono una descrizione, il CVE² di riferimento e le modalità per mitigare. Successivamente i risultati possono essere esportati in vari formati, tra cui PDF, HTML, TXT e XML.

Nessus è diventato uno standard a livello professionale, ma sul mercato è anche presente un vulnerability scanner open source chiamato OpenVAS³. Presenta le stesse funzionalità di Nessus, tra cui host discovery, network scan e vulnerability scan, ma è una valida alternativa per chi cerca software senza licenza a pagamento.

²<https://www.cve.org/>

³Common Vulnerabilities and Exposures, <https://www.openvas.org/>

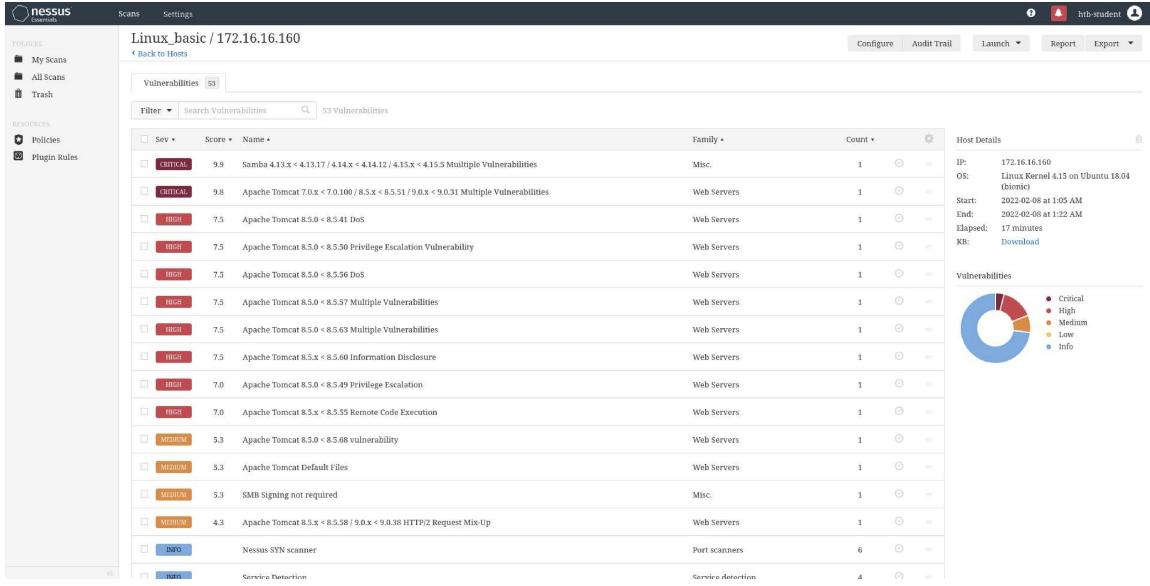


Figura 2: Pagina dei risultati contenente le vulnerabilità presenti sull'host

2.2 Nmap

Nmap⁴ è uno scanner di rete open source che permette la rilevazione di host attivi e il riconoscimento di porte aperte e servizi. È uno strumento che viene utilizzato esclusivamente da linea di comando (Zenmap è la versione che implementa una Graphical User Interface), ma permette il salvataggio dei risultati su file. Il funzionamento si basa sull'invio di pacchetti e la conseguente analisi delle risposte.

Una scansione completa di Nmap si basa sulle seguenti fasi:

- scoperta degli host: Nmap verifica se gli host sono attivi e raggiungibili inviando pacchetti e analizzando le informazioni contenute nella risposta. Sono presenti diverse modalità di scansione (ICMP, TCP-SYN, TCK-ACK, UDP...), ognuna utilizzata per svolgere analisi diverse, tra cui evasione di firewall o test delle difese nella rete.
- scansione delle porte: utilizzata per individuare quali porte sono aperte, filtrate (se è presente un dispositivo di difesa) o chiuse. Anche in questo caso Nmap può utilizzare modalità di scansioni differenti per ottenere risultati più precisi.
- scansione protocolli, servizi e sistema operativo: Nmap riconosce un protocollo quando all'invio di un pacchetto formattato secondo lo standard, riceve una risposta corretta che segue le regole stabilite dal protocollo stesso. Se è stata richiesta l'individuazione dei servizi, vengono inviati dei "probe" verso le porte e le risposte

⁴<https://nmap.org/>

sono comparate con un database contenente le relative firme e versioni. Inoltre è possibile riconoscere il sistema operativo utilizzato andando ad analizzare bit per bit il contenuto di pacchetti TCP e UDP inviati.

- esecuzione di script: il punto di forza di Nmap è l’NSE (Nmap Scripting Engine) che permette l’utilizzo di script per ampliare le funzionalità del network scanner, consentendo l’individuazione di informazioni dettagliate su servizi o testando la presenza di vulnerabilità. Non è paragonabile all’efficacia di un vulnerability scanner, ma può essere un ottimo indicatore di falle di sicurezza nelle fasi iniziali di scansione della rete.

Una volta terminate le scansioni i risultati vengono mostrati su linea di comando, ma utilizzando il parametro `-o` è possibile salvarli all’interno di diverse tipologie di file. Il report finale mostra le porte, i protocolli utilizzati e i servizi presenti sull’host. Se sono stati utilizzati degli script viene anche riportato il risultato della loro esecuzione.

```
> nmap -sV 192.168.100.175
Starting Nmap 7.97 ( https://nmap.org ) at 2025-09-14 14:38 +0200
Nmap scan report for 192.168.100.175
Host is up (0.0033s latency).
Not shown: 977 closed tcp ports (conn-refused)
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsftpd 2.3.4
22/tcp    open  ssh          OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet       Linux telnetd
25/tcp    open  smtp         Postfix smtpd
53/tcp    open  domain       ISC BIND 9.4.2
80/tcp    open  http         Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp   open  rpcbind     2 (RPC #100000)
139/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp   open  exec         netkit-rsh rexecd
513/tcp   open  login        login
514/tcp   open  tcpwrapped
1099/tcp  open  java-rmi   GNU Classpath grmiregistry
1524/tcp  open  bindshell   Metasploitable root shell
2049/tcp  open  nfs          2-4 (RPC #100003)
2121/tcp  open  ftp          ProFTPD 1.3.1
3306/tcp  open  mysql        MySQL 5.0.51a-3ubuntu5
5432/tcp  open  postgresql   PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp  open  vnc          VNC (protocol 3.3)
6000/tcp  open  X11          (access denied)
6667/tcp  open  irc          UnrealIRCd
8009/tcp  open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp  open  http         Apache Tomcat/Coyote JSP engine 1.1
Service Info: Hosts: metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.22 seconds
```

Figura 3: Risultati della scansione tramite Nmap

2.3 Metasploit

Metasploit⁵ è un framework impiegato da esperti di cybersicurezza per svolgere penetration test. Tramite moduli sviluppati dalla comunità è in grado di compiere diverse operazioni, suddivise in:

⁵<https://www.metasploit.com/>

- auxiliary: script per analizzare, raccogliere, scansionare o testare DoS sulla macchina avversaria.
- encoder: utilizzati per codificare caratteri da inviare che potrebbero creare problemi.
- evasion: utili per manipolare payload e renderli difficilmente rilevabili da meccanismi di difesa.
- exploit: sfruttano vulnerabilità che permettono l'esecuzione di un payload sulla macchina compromessa
- nop: inviano una sequenza di "No Operation". Utilizzato in associazione con buffer overflow.
- payload: contengono shellcode utile da iniettare successivamente all'exploitation di un host.
- post: moduli utili per raccogliere ulteriori informazioni a seguito della manomissione di un host.

```

shared-
+ ---[ metasploit v4.14.10-dev
+ ---[ 1640 exploits - 944 auxiliary - 289 post
+ ---[ 472 payloads - 48 encoders - 9 nops
+ ---[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use exploit/windows/smb/eternalblue_doublepulsar
msf exploit(eternalblue_doublepulsar) > set永恒bluepath /root/Tools/Eternalblue-Doublepulsar-Metasploit/deps
永恒bluepath => /root/Tools/Eternalblue-Doublepulsar-Metasploit/deps
msf exploit(eternalblue_doublepulsar) > set doublepulsarpath /root/Tools/Eternalblue-Doublepulsar-Metasploit/deps
doublepulsarpath => /root/Tools/Eternalblue-Doublepulsar-Metasploit/deps
msf exploit(eternalblue_doublepulsar) > set targetarchitecture x64
targetarchitecture => x64
msf exploit(eternalblue_doublepulsar) > set processinject lsass.exe
processinject => lsass.exe
msf exploit(eternalblue_doublepulsar) > set rhost 192.168.100.210
rhost => 192.168.100.210
msf exploit(eternalblue_doublepulsar) > set lhost 192.168.100.110
lhost => 192.168.100.110
msf exploit(eternalblue_doublepulsar) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf exploit(eternalblue_doublepulsar) > exploit

[*] Started reverse TCP handler on 192.168.100.110:4444
[*] 192.168.100.210:445 - Generating Eternalblue XML data
[*] 192.168.100.210:445 - Generating Doublepulsar XML data
[*] 192.168.100.210:445 - Generating payload DLL for Doublepulsar
[*] 192.168.100.210:445 - Writing DLL in /root/.wine/drive_c/eternal11.dll
[*] 192.168.100.210:445 - Launching Eternalblue...
[+] 192.168.100.210:445 - Pwned! Eternalblue success!
[*] 192.168.100.210:445 - Launching Doublepulsar...
[*] Sending stage (1189423 bytes) to 192.168.100.210
[*] Meterpreter session 1 opened (192.168.100.110:4444 -> 192.168.100.210:49158) at 2017-05-14 14:58:48 -0400
[+] 192.168.100.210:445 - Remote code executed... 3... 2... 1...

meterpreter > sysinfo
Computer : CLIENT-02
OS : Windows 7 (Build 7600).
Architecture : x64
System Language : en US
Domain : HACKABLE
Logged On Users : 2
Meterpreter : x64/windows
meterpreter >

```

Figura 4: Esempio di esecuzione di un modulo Metasploit

Il framework contiene strumenti aggiuntivi per poter ampliare le funzionalità e semplificare l'utilizzo. La console di Metasploit viene avviata da terminale tramite il comando

`msfconsole`. Questo strumento è utilizzato per interagire con il framework e per poter utilizzare i moduli. Per essere avviato è necessario richiamare `msfconsole` da shell e, dopo un rapido caricamento, comparirà `msf >`, indicandone il corretto avvio. Raggiunta questa schermata sarà possibile selezionare i moduli necessari per iniziare la fase di configurazione degli script, in cui verrà richiesta la compilazione di IP, porte, file, flag etc. Una volta inserite le informazioni il modulo potrà essere eseguito tramite `run` e, una volta ultimato, verranno mostrati a schermo i risultati.

`MSFdb` è uno strumento di gestione di database e si basa su PostgreSQL. Permette l'importazione di scansioni da parte di scanner, come Nmap o Nessus, oppure l'esportazione di risultati ottenuti dai moduli di Metasploit.

`MsfVenom` offre la possibilità di creare payload personalizzati per varie tipologie di target. Un punto di forza di questo strumento è la possibilità di codificarli utilizzando tecniche semplici o più avanzate. In questo modo il contenuto dei pacchetti inviati risulta meno sospetto agli occhi dei sistemi di difesa.

`Meterpreter` è un payload altamente avanzato e pieno di funzionalità che può essere installato all'interno della memoria della vittima. Oltre a stabilire una connessione con l'host, offre la possibilità di crittografare i pacchetti inviati, recuperare hash e manipolare i file all'interno del dispositivo. In generale è il payload utilizzato di default dagli exploit di metasploit.

L'ultima funzionalità trattata è Armitage, la GUI⁶ scritta in Java per poter interagire con il framework. A seguito di scansioni mostra, tramite un grafico, i collegamenti tra gli host di una rete. Per ogni dispositivo indica anche gli exploit che potrebbero essere utilizzati, oltre a fornire funzionalità per la navigazione di file e il recupero di hash.

Metasploit è uno strumento molto potente che può essere utilizzato anche per scopi illeciti. Molti attaccanti sfruttano le sue potenzialità per cercare di compromettere dispositivi e reti in tutto il mondo a scopo di lucro.

⁶Graphical User Interface

Capitolo 3

Protocolli analizzati

Prima di analizzare le fasi di un vulnerability scanner è buona cosa descrivere i protocolli alla base del suo funzionamento. Le reti di computer riescono a scambiarsi dati grazie ad uno standard chiamato Internet Protocol Suite, che definisce delle regole per poter comunicare tra host eterogenei. È composto da 4 livelli, ovvero Datalink, Internet, Trasporto e Applicazione, ma all'interno di questo progetto verranno trattati solamente gli ultimi tre. Per ogni livello sono presenti protocolli che svolgono funzioni diverse, che variano dalla semplice trasmissione dei pacchetti ad una gestione dei file condivisi sulla rete. Per motivi di complessità e tempo, nel progetto sono stati studiati e applicati i protocolli più utilizzati, partendo dai più semplici fino ad arrivare a protocolli più avanzati.[2]

3.1 Protocolli Internet Layer

Il secondo livello della pila TCP/IP è rappresentato dall'Internet layer. I protocolli, metodi e specifiche definiti permettono l'instradamento di pacchetti tra host appartenenti a reti diverse. I protocolli più utilizzati in questo livello sono l'IP e l'ICMP.

ICMP

L'Internet Control Message Protocol è principalmente utilizzato per l'invio di codici di stato per dare informazioni o indicare errori quando due dispositivi comunicano. Il suo scopo è quello di offrire uno strumento per diagnosticare e verificare lo stato dei computer connessi alla rete.[3]

Questo protocollo si basa sull'invio di "Control Messages", numeri che indicano degli stati ben definiti in cui si trova il sistema. Sono suddivisi in "Type", che definisce una categoria generica, e "Code", che rappresenta in modo dettagliato lo stato. Nello sviluppo di questo progetto viene tenuto conto di un numero ristretto di Control Messages, riportati nella tabella seguente:

Type	Descrizione	Utilizzo
0	Echo Reply	Risposta del server ad un echo request del client
3	Destination Unreachable	Il server è spento o non è raggiungibile
8	Echo Request	Client richiede lo stato del server

Tabella 1: Control Messages principali di ICMP

3.2 Protocolli Transport Layer

Il terzo livello della pila TCP/IP è rappresentato dal Transport Layer. I protocolli a livello di trasporto sono utilizzati per fornire un canale di comunicazione tra due host per l'invio di pacchetti. I due più utilizzati sono TCP(Transmission Control Protocol), che stabilisce una connessione affidabile, e UDP(User Datagram Protocol) che è rapido a discapito dell'inaffidabilità.

TCP

Il Transmission Control Protocol è un protocollo orientato alla connessione che permette l'invio di pacchetti in modo affidabile ed ordinato. Viene utilizzato principalmente per l'invio di file, in quanto è necessario ricevere tutte le informazioni nella giusta sequenza per poterlo ricostruire in modo corretto. [4]

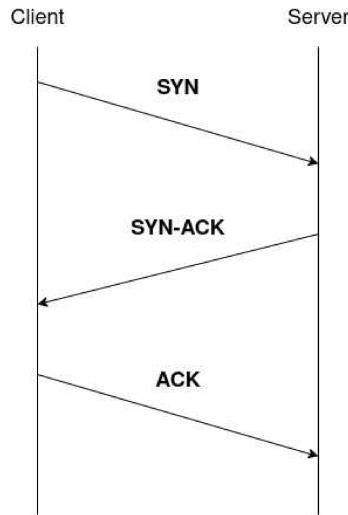


Figura 5: Three-Way Handshake tra Client e Server

All'interno dell'header di un pacchetto TCP sono presenti 8 bit detti "flags" che indicano lo stato della connessione. Questo è molto importante per svolgere l'operazione

di "Three-Way Handshake", il processo per stabilire una connessione con l'interlocutore. Il Client ed il Server si scambiano rispettivamente le flag di SYN, SYN-ACK, ACK nella sequenza descritta, portando ad 1 i bit corrispondenti all'interno dell'header TCP. Nella figura 5 viene illustrato il procedimento per l'apertura di una connessione TCP.

Oltre a queste flag, il protocollo TCP presenta altri valori che gli host si possono scambiare per ricevere maggiori informazioni sulla connessione. I principali utilizzati dal programma sviluppato sono:

- URG: indica l'urgenza del pacchetto.
- ACK: notifica al mittente che il pacchetto è stato ricevuto correttamente,
- PSH: richiesta di spingere i dati nel buffer dell'applicazione ricevente.
- RST: resetta la connessione.
- SYN: utilizzato per stabilire una connessione
- FIN: definisce l'ultimo pacchetto che il mittente vuole inviare prima della chiusura della connessione.

UDP

Il protocollo UDP, a differenza di quello TCP, non è orientato alla connessione e di conseguenza la ricezione dei pacchetti non viene garantita. L'unità che viene inviata tra due host via UDP si chiama "datagramma". Gli utilizzi principali riguardano la trasmissione di dati in tempo reale poiché non è necessario ricevere tutti i pacchetti in modo ordinato. A discapito dell'affidabilità garantita da TCP tramite diversi controlli, UDP risulta molto più leggero e veloce.[5]

3.3 Protocolli Application Layer

L'ultimo livello della pila TCP è occupato dall'Application layer. È il livello che permette la comunicazione tra applicazioni presenti su due host differenti. Di conseguenza è il livello che contiene il maggior numero di protocolli. Ognuno si occupa di operazioni differenti, tra cui file sharing, gestione email, connessione a terminali remoti ecc.

In questa sezione i protocolli sono divisi in due categorie: protocolli semplici, che rappresentano i più utilizzati e i più facili da implementare, e i protocolli avanzati, che sfruttano meccanismi più complessi.

3.3.1 Protocolli semplici

FTP

Il File Transfer Protocol permette lo scambio di file da un dispositivo ad un altro. Ha un'architettura di tipo client-server e utilizza comandi predefiniti per permettere la comunicazione. Il server può richiedere al client un'autenticazione prima di svolgere delle operazioni. Se è abilitato si può eseguire il login tramite lo user "anonymous" e lasciando la password vuota.[6]

Una volta stabilita una connessione con il server si utilizzano i comandi descritti nella seguente tabella per comunicare.

Comando	Descrizione
USER <username>	Inserire nome utente
PASS <password>	Inserire password dell'utente
PORT <host-port>	Specifica un indirizzo e una porta su cui il server dovrebbe connettersi

Tabella 2: Comandi principali di FTP

Il server utilizza codici di risposta per indicare il risultato dei comandi inviati:

- 1xx e 3xx: errore
- 2xx: successo
- 4xx e 5xx: fallimento nel rispondere

Telnet

Telnet è un protocollo client-server basato su connessioni TCP che viene utilizzato per interagire con i terminali di dispositivi remoti. È stato deprecato per via di grossi problemi di sicurezza, tra cui la trasmissione in chiaro dei dati e la mancata autenticazione tra host. [7]

Per poter interagire con terminali eterogenei tra loro Telnet utilizza il Network Virtual Terminal (NVT), un terminale virtuale che traduce i comandi nativi della macchina a comandi generici di Telnet. Una volta che client e server Telnet hanno stabilito una connessione, la comunicazione avviene tramite caratteri a 8 bit. Per inviare dati viene impostato il bit più significativo a 0 e i restanti bit vengono interpretati come caratteri ASCII. Per inviare comandi il bit più significativo viene impostato a 1. Telnet utilizza un'unica connessione per inviare dati e caratteri di controllo, ma all'interno dei messaggi, questi ultimi vengono preceduti da un carattere speciale chiamato Interpret as Control (IAC).

SMTP

Il Simple Mail Transfer Protocol è utilizzato principalmente per il trasferimento di mail tra server di posta. Finita la composizione di una mail, l'host invierà il messaggio tramite il protocollo ad un primo server. Quest'ultimo a sua volta può passare il messaggio ad altri host finché l'email non sarà recapitata al server di destinazione. Per poter scaricare le e-mail sul client sarà necessario utilizzare i protocolli POP e IMAP, che verranno trattati nelle sezioni successive.[8]

I principali comandi utilizzati per la comunicazione sono i seguenti:

Comando	Descrizione
HELO/EHLO	Permette di inizializzare la connessione tra due server
MAIL FROM <mittente@mail.com>	Inizializza il trasferimento di mail. È necessario specificare la mailbox del mittente da trasferire
RCPT TO <destinatario@mail.com>	Specifica il destinatario. È possibile specificarne più di una mailbox
DATA	Tramite questo comando il client chiede al server di poter spedire i dati contenuti nelle mail. Una volta che il server avrà risposto positivamente, i messaggi verranno inviati riga per riga

Tabella 3: Comandi principali di SMTP

In seguito all'invio di un messaggio ad un server, esso risponderà inviando un codice numerico che indica lo stato della transazione. Di seguito sono riportati i codici utilizzati con una piccola descrizione:

- 2xx: operazione compiuta con successo
- 3xx: operazione intermedia compiuta con successo
- 4xx: errore temporaneo
- 5xx: errore permanente

DNS

Il Domain Name System consente l'assegnazione di nomi, detti anche nomi di dominio,

agli host di una rete. Noi umani siamo molto più bravi a ricordarci nomi al posto di cifre e numeri. Per questo motivo si è voluto inventare un modo per poter assegnare dei nomi agli IP degli host. La traduzione da nome a IP è necessaria poiché i protocolli sottostanti hanno la necessità di utilizzare un indirizzo IP per poter funzionare secondo lo standard.[9]

Prima di passare al funzionamento del protocollo è buona cosa parlare della gerarchia dei server DNS. Se un client vuole collegarsi ad un sito web di cui conosce solo il nome di dominio e non il suo IP, sarà costretto ad interrogare il DNS resolver. Questo server è in grado di interagire con altri server DNS per poter recuperare le informazioni necessarie alla traduzione. Come primo passaggio il resolver interroga il Root DNS server, che fornisce l'indirizzo del Top Level Domain DNS server, ovvero colui che contiene i record per singoli TLD (per esempio .com o .net). Il resolver a questo punto interroga il TLD DNS server specificato precedentemente, che a sua volta restituisce l'indirizzo IP del nameserver autoritativo di dominio. A questo punto il resolver interroga come ultima cosa il server autoritativo che procoederà a restituire l'IP del server su cui si trova il servizio con il nome di dominio a cui il client voleva accedere. A questo punto il resolver invia al client l'IP, con cui riuscirà a collegarsi al server.[10]

Ogni server DNS contiene al suo interno un database con campi che possono essere di diverso tipo. Di seguito sono riportati i più utilizzati:

Tipo del campo	Descrizione
A e AAAA	Ritorna l'indirizzo IPv4 (A) oppure IPv6 (AAAA) di un host partendo dal suo nome
CNAME	Il nome di dominio specificato fa riferimento ad un altro nome
MX	Lista dei server che si occupano della ricezione di mail nel dominio
NS	Specifica il server autoritativo per il dominio specificato
SOA	Specifica le informazioni del server DNS autoritativo, fornendo il name server primario, la mail dell'amministratore e altre informazioni
AXFR	Utilizzato per trasferire l'intero file della zona contenente i record DNS tra un server DNS primario ed uno secondario

Tabella 4: Comandi principali di SMTP

Quando un client fa un richiesta ad un server viene inviato un messaggio che contiene al suo interno una query. Questa è composta da tre elementi: il nome, ovvero il nome di dominio che si vuole contattare, il tipo, cioè quale tipologia di campo DNS si vuole ricevere, e la classe, utilizzata per definire la classe del campo. Una volta ricevuta questa richiesta, il server risponderà a sua volta tramite un messaggio che contiene la risposta.

All'interno possono esserci più elementi che descrivono tutti i record analizzati per arrivare all'IP desiderato. Ogni record presenta il nome, il tipo, la classe e in aggiunta dei dati, che a loro volta possono contenere un CNAME a cui fare riferimento oppure l'indirizzo IP stesso.

All'interno di un vulnerability scanner si cerca di fare una richiesta di tipo SOA verso le varie porte dei server per individuare quali utilizzano il protocollo. Se rispondono inviando informazioni sul server autoritativo c'è la certezza che la porta analizzata sia impiegata per le richieste e risposte di un DNS.

HTTP

L'HyperText Transfer Protocol è uno tra protocolli più utilizzati e permette lo scambio di informazioni tramite Internet. Durante gli anni sono state implementate diverse versioni, rispettivamente HTTP/1.0, HTTP/1.1, HTTP/2 e HTTP/3. Il protocollo implementa l'architettura client-server di tipo request-response. Questa tipologia funziona nel seguente modo: un host client invia una richiesta ad un server che elabora le informazioni per restituire una risposta. [11]

Prima di analizzare i messaggi è necessario introdurre il concetto di metodi, header, body e codici di stato.

I metodi sono utilizzati all'interno delle richieste per specificare al server quale operazione si vuole svolgere su una risorsa. I metodi più utilizzati sono i seguenti:

Tipo del campo	Descrizione
GET	Ricevere il dato indicato, senza apportare modifiche
POST	Inviare informazioni verso la risorsa specificata
PUT	Creare o modificare la risorsa specificata
DELETE	Eliminazione della risorsa specificata
HEAD	Restituisce gli header di risposta senza i contenuti del messaggio
OPTIONS	Describe i metodi supportati dal server per la comunicazione

Tabella 5: Metodi di HTTP

Gli header sono principalmente suddivisi in header di richiesta e header di risposta. Possono fornire informazioni aggiuntive sul messaggio inviato e gli host interessati, ma anche definire regole di sicurezza per limitare l'accesso di risorse a host non autorizzati. L'unico header obbligatorio all'interno di una richiesta HTTP è Host: www.example.com, mentre i restanti possono essere omessi.

Il prossimo elemento da discutere è il body. Nelle richieste corrisponde alle informazioni o i dati che si vogliono comunicare al server per svolgere, ad esempio, un login o

l'upload di un file. Nelle risposte invece contiene i dati della risorsa che è stata richiesta. Può essere una pagina web, uno script o addirittura un'immagine.

Infine i codici di stato vengono utilizzati dal server per comunicare al client lo stato dell'elaborazione della richiesta. Sono rappresentati da numeri di 3 cifre, suddivisi in modo più generico in:

- 1xx informational: invia informazione sullo stato delle operazione verso la specifica richiesta
- 2xx successful: l'operazione è andata a buon fine
- 3xx redirection: per completare la richiesta è necessario svolgere ulteriori operazioni
- 4xx client error: la richiesta è errata e non può essere soddisfatta
- 5xx server error: il server non è in grado di compiere l'operazione

A titolo d'esempio vengono mostrati i messaggi scambiati tramite protocollo HTTP/1.1.

```
GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Codice 3.1: Richiesta del client

All'interno della richiesta sono presenti diversi elementi. Nella prima riga viene specificato il metodo di richiesta, il path della risorsa che si vuole accedere e la versione di HTTP che si vuole utilizzare. Le righe sottostanti sono composte dagli header della richiesta. Per quanto riguarda le richieste GET il messaggio dovrà terminare con due linee contenti i caratteri carriage return e line feed. Nelle richieste POST o PUT viene riportata solo una linea di terminazione, seguita dal contenuto dell'informazione che si vuole inviare al server.

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 155
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
```

```

Accept-Ranges: bytes
Connection: close

<html>
  <head>
    <title>An Example Page</title>
  </head>
  <body>
    <p>Hello World, this is a very simple HTML document.</p>
  </body>
</html>

```

Codice 3.2: Risposta del server

Nella prima riga della risposta è presente la versione di HTTP e il codice di stato a seguito dell'operazione. Nelle righe sottostanti sono riportati gli header di risposta e, dopo una linea contenente carriage return e line feed, viene riportata la risorsa richiesta.[12]

POP

Il Post Office Protocol viene utilizzato per recuperare, tramite un client, le e-mail presenti su un mail server. La versione più recente che viene utilizzata è la POP3. Questo protocollo, una volta collegato al server di posta, scarica tutte le e-mail presenti e le salva sul client. Successivamente le elimina definitivamente dal server. Questo metodo per scaricare la posta è un approccio che non rispecchia le necessità moderne di poter accedere alle medesime risorse da diversi dispositivi. Per questo è stato implementato il protocollo IMAP, discusso nella sezione successiva.[13]

Per poter comunicare con il server di posta remoto, l'utente deve autenticarsi e successivamente utilizzare una serie di comandi per comunicare.

Tipo del campo	Descrizione
STAT	Mostra il numero di messaggi di una mailbox
LIST	Ritorna la lista di messaggi nella mailbox
RETR	Recupera il messaggio specificato
DELE	Elimina il messaggio dal server, di solito dopo che l'utente l'ha scaricato in locale

Tabella 6: Comandi POP3

IMAP

L'Internet Message Access Protocol viene utilizzato per recuperare le mail da un server di posta. A differenza del protocollo POP3, il client scarica una copia delle mail presenti sul server, in modo tale da poterle visionarle su più dispositivi. Le modifiche apportate alla casella di un client vengono rispecchiate su tutti i restanti dispositivi. [14]

La comunicazione tra client e server avviene tramite l'utilizzo di comandi.

Tipo del campo	Descrizione
SELECT	Specifica a quale mailbox accedere
EXAMINE	Accede alla mailbox in modalità solo lettura
LIST	Mostra le cartelle presenti nella mailbox
FETCH	Recupera il contenuto delle e-mail

Tabella 7: Comandi IMAP

SMB

Il Server Message Block è un protocollo utilizzato per la condivisione di file attraverso dispositivi presenti sulla rete. I principali dialetti SMB sviluppati nel corso degli anni sono: SMB1, CIFS, SMB2 e SMB3.[15]

Anche in questo caso la comunicazione è di tipo client-server. Per poter accedere ad una risorsa è necessario compiere i seguenti passaggi, portati a termine inviando pacchetti formattati secondo lo standard SMB:

- Stabilire una connessione tra client e sever tramite l'utilizzo di NetBIOS, un protocollo che permette la comunicazione tra dispositivi appartenenti alla stessa LAN.
- Negoziazione del dialetto di SMB che si vuole utilizzare. Il pacchetto inviato in questo passaggio è SMB_COM_NEGOTIATE
- Il client svolge l'autenticazione verso il server.
- Connessione del client verso una share del server, ovvero una cartella condivisa. La richiesta viene portata a termine tramite il pacchetto SMB_COM_TREE_CONNECT_ANDX
- Il client ora è in grado di aprire e leggere i contenuti dei file presenti sulla share del server. I pacchetti utilizzati per svolgere queste due operazioni sono SMB_COM_OPEN_ANDX e SMB_COM_READ_ANDX

3.3.2 Protocolli avanzati

SSH

Il Secure Shell è un protocollo crittografico per svolgere operazioni su dispositivi presenti sulla rete. È stato sviluppato per sostituire i protocolli Telnet e Remote Shell per via delle loro comunicazioni non crittografate.

Come già accennato SSH utilizza crittografia a chiave pubblica per autenticarsi verso gli host remoti. L'autenticazione dell'utente può avvenire in due principali modalità: tramite password oppure publickey. Nella prima il protocollo genera automaticamente una coppia di chiavi e poi richiede all'utente una password. Nella seconda l'utente genera una coppia di chiavi che lo autentica automaticamente. Per potersi collegare ai dispositivi senza usare la password sarà necessario inserire la chiave privata al loro interno.

Il protocollo secure shell permette anche di svolgere diverse operazioni, rendendo obsoleti vecchi protocolli. Oltre alla possibilità di fare login su host della rete permette anche il trasferimento di file, il tunneling, il port forwarding e molto altro.[16]

L'architettura di SSH è suddivisa in tre livelli:

- Transport layer: gestisce lo scambio di chiavi, l'autenticazione con il server e la crittografia del canale.
- Authentication layer: gestisce l'autenticazione tramite un gran numero di algoritmi. Il dispositivo client sarà sempre il primo a richiedere l'autenticazione al server.
- Connection layer: ogni connessione SSH può essere suddivisa e gestita su più canali che sono in grado di scambiare dati in modo bidirezionale.

MQTT

Message Queuing Telemetry Transport è un protocollo leggero di tipo publish-subscribe utilizzato soprattutto sui dispositivi dell'Internet of Things.

L'MQTT definisce due dispositivi distinti: il broker e il client. I client per poter comunicare con i broker devono fare il subscribe a dei "Topic". Successivamente possono inviare i messaggi usando il comando publish, che invia al broker messaggi aventi un Topic associato. A questo punto il broker funge da intermediario, indirizzando i messaggi in base al loro Topic verso tutti i dispositivi sottoscritti. I client possono anche ricevere configurazioni o comandi data la bidirezionalità dal canale di comunicazione.[17]

NFS

Il Network File System permette agli utenti di un computer di accedere a file condivisi

sulla rete. Il protocollo si articola in 3 versioni differenti: NSFv2, NFSv3 e NFSv4. La comunicazione tra client e server avviene tramite l'utilizzo di Remote Procedure Call che servono per richiamare subroutine su un altro dispositivo.[18]

Per poter interagire con i file presenti sulla rete, il client ed il server devono svolgere i seguenti passaggi:[19]

- Il server presenta tre servizi: il port mapper, mountd daemon e NFSD daemon. Il port mapper rimane in ascolto sulle porte 1111 UDP e TCP. NFSD e mountd richiedono al port mapper di registrare il numero delle porte su cui operare.
- Il client a questo punto richiede di montare il filesystem presente sul server. Il servizio di port mapping restituisce al client la porta su cui risiede il servizio mountd
- A questo punto il client richiede a mountd di poter montare il filesystem. Il server lo autentica e svolge l'operazione di montaggio. Al termine comunica al client i risultati dell'operazione.
- Per poter svolgere operazioni di lettura e scrittura il client richiede al mapper la porta su cui risiede il servizio NFSD. Una volta restituito il valore, il client potrà svolgere operazioni di lettura e scrittura sul file system montato.

SSL/TLS

Il Secure Socket Layer e il Transport Layer Security sono dei protocolli crittografici che offrono la possibilità di cifrare comunicazioni attraverso la rete.[20] Il protocollo SSL ha avuto 3 versioni, ma è stato deprecato a favore delle varianti TLS1.2 e TLS1.3 che sono molto più sicure.[21] Questi protocolli offrono anche la possibilità di autenticazione grazie all'utilizzo di certificati stipulati tra due o più applicazioni che vogliono comunicare.

Per poter rendere sicura una connessione è necessario l'utilizzo di un certificato digitale. Questi sono utilizzati per associare un'identità ad una chiave pubblica e vengono rilasciati da Certification Authority. Il TLS ha un legame di fiducia verso alcuni CA di terze parti che utilizza per verificare l'autenticità dei certificati utilizzati.[22]

Per iniziare una connessione sicura il client ed il server devono svolgere le seguenti operazioni:

- Il client si connette ad una porta su cui è possibile iniziare una connessione sicura. Invia al server una lista che contiene tutti i cifrari che supporta.
- Il server a questo punto invia un messaggio al client specificando il cifrario scelto. Inoltre invia il certificato digitale per identificarsi.
- Il client verifica la correttezza dei dati e la validità del certificato.

- Successivamente il client deve creare una chiave di sessione cifrando un numero casuale oppure utilizzando lo scambio di chiavi Diffie-Hellman.
- Una volta portate a termine queste operazioni verrà inizializzata la connessione sicura.

Capitolo 4

Progettazione

I programmi precedentemente citati nello Stato dell'Arte sono molto complessi ed avanzati, con una curva di apprendimento abbastanza ripida. Per poter utilizzare un vulnerability scanner è necessario inizialmente scaricare una grande mole di dati e informazioni per poter svolgere i test sui dispositivi. Inoltre gli scripting engines utilizzati per la realizzazione dei test su vulnerabilità e configurazioni errate possono risultare complessi per un utente neofita.

4.1 Obiettivo dell'elaborato

L'obiettivo prefissato è quello di proporre un'architettura più leggera e personalizzabile, in modo tale da poter rispondere alle necessità di ogni utilizzatore. Un termine che ha caratterizzato lo sviluppo di questo applicativo è "modularità": ogni elemento del software deve essere in qualche modo modificabile con facilità utilizzando template prestabiliti, per semplificarne lo sviluppo, ma anche la creazione dei file di test. Questa ideologia ha portato alla scelta di suddividere ogni fase del vulnerability scan in moduli richiamabili singolarmente, per dare più flessibilità agli utenti e anche agli sviluppatori.

Continuando a seguire il concetto di "modularità", viene proposto un modo alternativo per poter svolgere i test sui protocolli/servizi. Al posto di utilizzare engine di scripting che possono essere molto pesanti e difficili da apprendere, si è optato per l'utilizzo di file JSON che seguono dei template prestabiliti. Questa scelta facilita di molto la creazione e la modifica di nuovi test, permettendo di soddisfare il maggior numero di scenari. Un altro punto di forza di questo formato è la possibile automazione, tramite programmi terzi, di scrittura e ottenimento dei dati per poter compilare correttamente i test. Questo si può ottenere grazie al supporto nativo del formato presente nella maggior parte dei linguaggi di programmazione.

Il progetto nello stato attuale è molto leggero e facile da scaricare e spostare da un dispositivo all'altro. Grazie alla facilità e alla rapidità di utilizzo, l'utente è in grado di

installare e utilizzare in breve tempo l'applicativo. Anche per una persona poco esperta sarà possibile fare scansioni e creare test senza troppe difficoltà.

4.2 Analisi delle funzionalità

Studiando il funzionamento di altri software di vulnerability scan è stato possibile suddividere il funzionamento in cinque fasi ben definite:

- *Scansione dell'host*: verificare che l'host sia raggiungibile prima di compiere ulteriori operazioni.
- *Scansione delle porte*: individuare le porte che sono esposte verso l'esterno sugli host raggiungibili. Si possono presentare in tre stati diversi: open, filtered e closed.
- *Scansione dei protocolli e servizi*: individuare i protocolli e i servizi utilizzati attraverso le porte aperte degli host.
- *Esecuzione dei test*: eseguire batterie di test inviando comandi alle porte aperte e analizzando i messaggi di risposta ricevuti. Ogni protocollo e ogni servizio presenta test personalizzati.
- *Stampa dei risultati*: generare risultati utilizzando diverse tipologie di file da poter essere utilizzati secondo le diverse esigenze dell'utente.

4.2.1 Scansione dell'host

In questa fase viene verificato che l'host analizzato dall'applicativo sia raggiungibile ed attivo durante l'esecuzione della scansione. Qualora il dispositivo non fosse contattabile, l'esecuzione del programma verrebbe immediatamente interrotta per evitare scansioni senza esito.

Per svolgere questa operazione possono essere impiegate diverse metodologie di scansione dell'host, attraverso l'utilizzo dei protocolli ICMP, TCP e UDP:

- ICMP: la scansione ICMP verifica che a seguito dell'invio di un echo request venga ricevuto un echo reply. Qualunque messaggio di tipo Destination Unreachable dimostrerebbe l'irraggiungibilità dell'host. È necessario tenere conto di dispositivi che potrebbero bloccare completamente le richieste di PING, portando la verifica a risultati incorretti. Per ovviare a questo problema è possibile utilizzare scansioni di tipo TCP.
- TCP: le scansioni sono suddivise in SYN scan e ACK scan. Per poter funzionare, devono essere specificate delle porte a cui il programma dovrà tentare una connessione. In questo caso si possono utilizzare una serie di porte spesso presenti sugli

host (come la 80), alzando la probabilità di trovarle aperte per poter proseguire la scansione.

- TCP SYN: per svolgere questa scansione viene inviato un pacchetto con la flag SYN attivata. Per determinare lo stato dell'host come raggiungibile si attende una risposta che contenga la flag SYN-ACK attiva. Molti firewall di rete tendono a bloccare questo tipo di interazioni se vengono applicate politiche di rifiuto su nuove connessioni verso l'interno della rete.
- TCP ACK: ad un pacchetto contenente una flag con valore scorretto, il protocollo TCP risponde con un RESET. Inviando un pacchetto con ACK in una connessione non ancora stabilita si genera un errore, che porta alla ricezione di un pacchetto con flag RESET. Questo è un indicatore che l'host è attivo sulla rete e che riesce a rispondere alle richieste che gli vengono inviate. È un metodo molto più efficace contro i firewall rispetto al TCP SYN, che potrebbe essere bloccato.
- UDP: se nei protocolli ICMP e TCP si rimane in attesa per una risposta da parte dell'host, in UDP viene fatto il contrario. In questo caso, per definire un host attivo, viene verificato quale dispositivo non restituisce nessun pacchetto. Questo avviene perché interrogando un host attivo sulla rete tramite UDP, questo invierà una risposta di tipo ICMP Host Unreachable o Port Unreachable. In caso contrario non si riceve alcuna risposta. Questo metodo è molto più lento perché è necessario inserire dei timeout di qualche secondo per tenere conto dei delay che possono avvenire all'interno della rete. Inoltre, essendo il protocollo UDP senza connessione, è possibile che si possano verificare perdite di pacchetti, che porterebbero alla creazione di risultati scorretti.

Sono presenti molte altre metodologie per verificare la raggiungibilità di un host, ma nella maggior parte dei casi vengono utilizzate per infrastrutture con necessità specifiche o semplicemente per verificare e testare la presenza di un firewall.

4.2.2 Scansione delle porte

A seguito della verifica dell'host, viene fatta una scansione per trovare le porte presenti e il loro stato. Vengono impiegati metodi simili alla fase precedente, sfruttando i protocolli TCP e UDP, ma anche la libreria socket per stabilire una connessione vera e propria con l'host. A seguito di questa fase viene stilata una lista di porte presenti sul dispositivo con il relativo stato:

- Closed: la porta dell'host non è aperta e non sta ascoltando.
- Open: la porta dell'host è aperta e sta aspettando la ricezione di dati.

- Filtered: la porta può essere aperta o chiusa, ma un firewall ne blocca le funzionalità.
- Open/Filtered: la porta molto probabilmente è aperta, ma un firewall ne blocca le connessioni.

Come anticipato precedentemente possono essere utilizzate diverse metodologie per riconoscere lo stato di una porta:

- Connect: utilizzando la libreria socket, implementata in quasi tutti i linguaggi di programmazione, viene creato un canale di comunicazione diretta con l'host. Per stabilire la connessione è necessario svolgere il processo di three-way handshake del protocollo TCP. Questa operazione può risultare problematica se nella rete è presente un firewall, in quanto può bloccare le richieste di connessione verso dei dispositivi specifici.
- TCP: la scansione si suddivide principalmente in SYN, FIN, NULL e XMAS. Anche in questo caso necessita l'utilizzo di una lista di porte che potrebbero essere aperte per potersi connettere all'host e svolgere la scansione.
 - SYN: viene inviato un pacchetto con flag SYN attiva e successivamente vengono analizzati i flag presenti nella risposta. Se viene ricevuto SYN-ACK la porta è aperta, se si riceve RST la porta è chiusa, mentre se non si riceve alcuna risposta o ICMP unreachable la porta potrebbe essere filtrata da un firewall. Anche in questo caso è possibile che il firewall ostacoli le richieste in entrata in base alle policy utilizzate, portando alla creazione di risultati poco attendibili.
 - FIN: in questo caso viene impostato il bit di FIN a 1 nella richiesta. Questa opzione viene normalmente utilizzata per dichiarare la volontà di terminare la connessione verso un host. In questo caso però analizza la risposta ricevuta e se è presente RST vuol dire che la porta è chiusa, se non c'è risposta la porta è aperta, mentre per ICMP unreachable la porta può essere open/filtered.
 - NULL: in questo caso tutti i bit della flag sono impostati a 0. Se la risposta contiene RST la porta è chiusa, se non riceviamo risposte è aperta e se riceviamo ICMP unreachable significa che è open/filtered.
 - XMAS: I bit attivi della flag sono alternati e di conseguenza sono attivate FIN, URG e PSH. Viene chiamato XMAS perché ricorda le luci ad intermittenza degli alberi di natale. Come nei casi precedenti se la risposta contiene RST la porta è chiusa, se non riceviamo risposta vuol dire che è aperta e se riceviamo ICMP unreachable significa che è open/filtered.
- UDP: Questa scansione consiste nell'inviare un pacchetto con 0 byte di dati di tipo UDP. Se non viene ricevuta alcuna risposta vuol dire che la porta è open/filtered,

altrimenti la porta è open. Inoltre se nella risposta è presente ICMP port unreachable la porta è sicuramente chiusa, ma se riceviamo qualsiasi altro codice ICMP la porta è probabilmente filtered.

Anche in questo caso sarebbero presenti ulteriori metodologie per svolgere le scansioni, ma sono state omesse perché non compiono operazioni necessarie.

4.2.3 Scansione dei protocolli e servizi

Questa fase si basa sull'interazione con le porte aperte o filtrate per individuare quali protocolli e servizi sono presenti sull'host.

In base al tipo di scansione svolta (TCP o UDP) si provvederà ad utilizzare liste di scansioni differenti. Ad ogni porta vengono inviati messaggi formattati secondo lo standard di un protocollo. Quando una delle porte risponde correttamente, ovvero stabilisce una connessione senza errori, allora si è certi della presenza di quel protocollo specifico. Questo processo viene ripetuto finché tutti i protocolli supportati dal vulnerability scanner sono stati testati. Successivamente è anche possibile interrogare il servizio sulla porta per ottenere maggiori informazioni su nome e versione utilizzati.

Durante queste scansioni è anche buona cosa tenere conto del protocollo di sicurezza TLS/SSL per poter comunicare correttamente con servizi messi in sicurezza. Per questo vengono utilizzati certificati auto firmati per poter eseguire le scansioni.

4.2.4 Esecuzione dei test

Per svolgere la fase di test delle vulnerabilità è necessaria la creazione di file contenenti informazioni e metodologie, per poter valutare la sicurezza dei protocolli e dei servizi. All'interno dei file dovrà essere riportato un nome e una descrizione che definiscono la vulnerabilità da testare. È necessario specificare i comandi per poterla innescare, ovvero le stringhe specifiche da inviare alla porta e i messaggi da ricevere per confermare la vulnerabilità. Per aiutare l'utente con l'interpretazione dei risultati è stata implementata una scala di "Rischio" suddivisa in 3 livelli: low, medium, high.

Questa procedura deve essere svolta per il protocollo, ma anche per il servizio che lo sfrutta. Oltre alle vulnerabilità si tiene anche conto delle configurazioni errate che possono essere state utilizzate.

4.2.5 Stampa dei risultati

Dopo aver svolto tutte le scansioni necessarie viene creato un rapporto dettagliato contenente tutte le vulnerabilità e le configurazioni errate sul sistema. Il file presenterà una lista di porte aperte e i relativi protocolli e servizi utilizzati. Per ognuno di questi elementi verranno indicate le vulnerabilità o le configurazioni errate riscontrate durante i test.

Buona norma è quella di presentare i risultati in file di diversa tipologia, per permettere all’utente di utilizzare quello che più rispecchia le sue esigenze. In questo caso vengono generati file di tre tipologie: uno testuale di facile lettura, uno grafico per visualizzare meglio i risultati e uno formattato per poter essere elaborato da altri programmi o sistemi.

4.3 Protocoli

Il progetto mira a svolgere analisi e test su una lista ristretta e definita di protocolli. La modularità delle librerie implementate permette di ampliarne successivamente l’elenco.

Si è deciso di utilizzare dodici protocolli che per semplicità vengono suddivisi in due categorie: protocolli semplici, ovvero quelli più datati e con implementazioni più facili, e protocolli avanzati, moderni e leggermente più complessi. I protocolli semplici analizzati sono FTP, Telnet, SMTP, DNS, HTTP, POP, IMAP, SMB; mentre i protocolli avanzati sono SSH, MQTT, NFS, SSL/TLS.

L’SSL/TLS può essere utilizzato in combinazione con altri protocolli per rendere la comunicazione sicura, per evitare attacchi di sniffing¹. I protocolli messi in sicurezza che vengono riconosciuti dall’applicativo sviluppato sono i seguenti: FTPS, SMTPS, HTTPS, POPS, IMAPS, MQTTS.[23][24][25][26]

Gli standard definiscono su quali porte i relativi protocolli dovrebbero risiedere. Ciò non vieta la possibilità di poter tenere aperti servizi anche su porte non standard per via di esigenze specifiche. Di seguito sono riportate le porte di default:

Protocollo	Porta standard
FTP	21
SSH	22
Telnet	23
SMTP	25
DNS	53
HTTP	80
POP	110
IMAP	143
HTTPS	443
SMB	445
SMTPS	587
FTPS	990
IMAPS	993
POPS	995

¹Intercettazione del traffico da parte di terzi

MQTT	1883
NFS	2049

Tabella 8: Porte standard dei protocolli

Nel caso in cui i protocolli siano serviti su porte alternative lo scanner sarà comunque in grado di individuarli. Questo perché l'applicativo testerà tutti i protocolli supportati su tutte le porte, fornendo risultati corretti anche in casi particolari.

4.4 Tecnologie utilizzate

Il linguaggio di programmazione utilizzato per lo sviluppo di questo progetto è Python e per quanto riguarda la lettura e scrittura di file viene impiegato il formato JSON. Queste due tecnologie si integrano perfettamente tra di loro grazie a librerie specifiche di Python e la loro semplicità di scrittura e interpretazione hanno reso lo sviluppo molto rapido ed efficace.

4.4.1 Python

Python² è un linguaggio di programmazione interpretato ad alto livello. Tra i vari paradigmi supportati è presente quello ad oggetti, utilizzato nella realizzazione del progetto. La scelta di utilizzare questo linguaggio deriva dai seguenti punti di forza:

- Portabilità: un programma scritto in Pyhton non necessita di essere compilato, per questo può essere interpretato su qualunque dispositivo che abbia Python installato.
- Semplicità di sviluppo: la sintassi semplice e pulita aiuta uno sviluppo più rapido e un debugging più efficace.
- Gran numero di librerie utilizzabili: le librerie sviluppate dalla community possono essere utilizzate per ridurre la mole di codice da dover effettivamente scrivere. Viene utilizzato pip per gestire l'installazione delle librerie necessarie, che rende la gestione del codice ancora più semplice.

²<https://www.python.org/>

4.4.2 JSON

JSON (JavaScript Object Notation)³ è uno standard per formati di file e per lo scambio di informazioni. Un file JSON può contenere diverse tipologie di dati, tra cui booleani, numeri, stringhe e liste. La struttura utilizzata è molto simile alla semantica delle classi usate nei moderni linguaggi di programmazione. Python infatti permette di interagire nativamente con lo standard JSON, grazie a funzioni che traducono automaticamente oggetti in strutture JSON e viceversa.

³<https://www.json.org/>

Capitolo 5

Implementazione

Dopo aver discusso in modo generico la progettazione di un vulnerability scanner è possibile trattare l'implementazione proposta. L'idea principale è stata quella di creare moduli e packages utilizzabili singolarmente per sviluppare altri progetti inerenti al networking. Questo facilita la modifica e l'aggiunta di nuove funzionalità, ma invita anche un utilizzo alternativo delle librerie.

Per semplificare l'implementazione sono state impiegate le seguenti librerie di Python:

- argparse: parsing automatico degli argomenti passati dall'utente da linea di comando.
- scapy: manipolazione dei pacchetti di rete a basso livello, per poter controllarne il contenuto ad ogni step della pila TCP/IP
- socket: connessione a dispositivi o servizi tramite le socket di sistema
- certifi: creazione di certificati root auto-firmati per la connessione a servizi che comunicano tramite SSL/TLS

La struttura del progetto è articolata come segue:

- agent/: package contenente i moduli principali per svolgere le fasi del vulnerability scanner.
 - host_scan.py: modulo contenente funzioni per testare la raggiungibilità di un host.
 - port_scan.py: modulo contenente una classe ed i metodi per verificare lo stato delle porte di un host.
 - service_scan.py: modulo contenente una classe ed i metodi per individuare il servizio utilizzato su una porta e la sua versione

- results.py: modulo contenente una classe che raccoglie tutte le informazioni e le vulnerabilità presenti su un determinato servizio.
- execute_tests.py: modulo contenente classe e funzioni per la verifica di vulnerabilità sui servizi.
- res/: cartella in cui vengono generati i file dei risultati.
- tests/: cartella contenitore per i file JSON di test.
 - prot/: test da svolgere sui protocolli di rete.
 - serv/: test da svolgere sui servizi.
- utils/: package contenente moduli per la creazione dei risultati e per gestire l’interazione con la linea di comando.
 - parser.py: grazie alla libreria argparse vengono definiti i parametri accettati da linea di comando e di conseguenza crea un testo di aiuto. Gestisce anche la verifica della correttezza dei parametri passati.
 - report_template.html: template in html utilizzato per la creazione di una pagina statica grazie alla libreria Python jinja2.
 - terminal_colors.py: definisce funzioni per poter stampare a linea di comando testo colorato.
 - write_results.py: contiene funzioni per la generazione di file TXT, JSON e HTML contenenti i risultati delle scansioni.
- main.py: entry point del programma. Richiama tutte le funzioni contenute nelle librerie implementate in agent/ e utils/ per svolgere le fasi di un vulnerability scanner.

5.1 main.py

File principale dal quale vengono richiamate le funzioni presenti nei moduli del progetto. Svolge il parsing dei dati inseriti da linea di comando e controlla la loro correttezza. Se non vengono individuati errori, vengono eseguite in cascata le funzioni per scannerizzare host, porte, protocolli/servizi e infine quelle per testare le vulnerabilità. Come ultima cosa genera i file di risultati grazie a `write_results.py`

5.2 Agent

All'interno di questo package sono presenti tutti i moduli per interagire con l'host e svolgere le fasi principali di un vulnerability scanner. Le funzioni utilizzate sono suddivise in moduli in base alle fasi descritte nella sezione 4.2, ad esclusione della scrittura dei risultati che è inserita nel package utils/.

5.2.1 host_scan.py

Il primo modulo analizzato si occupa della verifica dello stato dell'host. In questo caso ci limitiamo a implementare le soluzioni precedentemente descritte nella sezione 4.2.1. Per far selezionare all'utente la scansione desiderata viene passato alla funzione principale il parametro host_args di tipo stringa, che identifica tramite un carattere la modalità da utilizzare. Oltre a questo parametro vengono anche passati ip come stringa e verbose come booleano, quest'ultimo utilizzato per definire la verbosità dell'output stampato su linea di comando.

La funzione sarà implementata come segue:

```

1 def host_scan(host_arg: str, ip: str, verbose: bool):
2     if verbose:
3         verbose_print(f"Verifying {ip}")
4
5     match host_arg:
6         case "p":
7             res_status = ping_scan(ip)
8         case "s":
9             res_status = tcp_syn_scan(ip)
10        case "a":
11            res_status = tcp_ack_scan(ip)
12        case "u":
13            res_status = udp_scan(ip)
14        case None:
15            res_status = ping_scan(ip)
16        case _:
17            print_fail("Cannot find host scan type")
18            sys.exit()
19
20    # Clean line
21    print("\033[K", end="\r")
22
23    return res_status

```

Codice 5.1: Funzione principale per la scansione di host

La selezione avviene tramite uno switch case che verifica il carattere passato alla funzione. Se non viene passato nulla si utilizzata una funzione di default, mentre se

viene inserito un carattere scorretto viene mostrato a schermo un errore e si termina l'esecuzione.

Ad ogni caso equivale una funzione differente che rappresenta le tipologie di scan precedentemente analizzate nella sezione 4.2.1. A titolo d'esempio sarà analizzata una sola funzione: quella per lo scan TCP SYN.

```

1 def tcp_syn_scan(ip: str) -> bool:
2     res_status = False
3
4     for port in SCAN_PORTS:
5         packet = IP(dst=ip) / TCP(dport=port, flags="S")
6         res = sr1(packet, timeout=2, verbose=0)
7
8         if res is not None:
9             flag_res = res.sprintf("%TCP.flags%")
10
11         if flag_res == "SA":
12             res_status = True
13
14 return res_status

```

Per poter analizzare i pacchetti a livello di trasporto viene impiegata la libreria scapy, che permette di forgiare un pacchetto con le flag TCP necessarie utilizzando le sue funzioni di costruzione. Per concatenare i livelli della pila appena costruiti viene utilizzato il simbolo /. Il risultato è un'assegnazione come la seguente: `packet = IP(dst=ip) / TCP(dport=port, flags="S")`, nel quale viene specificato l'ip e la porta del destinatario a cui inviare il pacchetto e la flag SYN attivata.

L'invio avviene tramite la funzione `res = sr1(packet, timeout=2, verbose=0)` che successivamente salva il risultato in una variabile `res`. Il `timeout` è di 2 secondi e viene utilizzato per lasciare un ampio lasso di tempo per la ricezione della risposta.

Infine viene verificato il contenuto della flag della risposta e se equivale ad un SYN-ACK c'è la certezza che l'host sia attivo, altrimenti non è possibile raggiungerlo. Per determinare lo stato del dispositivo viene utilizzata una variabile booleana `res_status` inizializzata a `False`. Nel caso la scansione desse esito positivo, `res_status` diventerebbe `True`. Questa variabile viene ritornata dalla funzione `host_scan` e utilizzata nel main per decidere se continuare oppure se terminare l'esecuzione del programma in caso l'host non sia raggiungibile.

5.2.2 port_scan.py

Questo modulo è strutturato in maniera simile a quello descritto precedentemente. Lo switch case è pressoché identico, ma richiama funzioni per la scansione di porte. Il nuovo elemento presente all'interno del modulo è una classe, il cui costruttore è implementato nella seguente maniera:

```

1 class PortScan:
2     def __init__(self, ip: str):
3         self.ip = ip
4         self.ports = {}
5         self.type = ""
6         self.open_ports = []

```

PortScan contiene l'indirizzo ip dell'host, un dizionario contenente le porte e il loro stato, la tipologia di scan utilizzato (TCP o UDP) e una lista di porte aperte. All'interno della classe sono presenti diversi metodi, tra cui `__str__` che permette una corretta formattazione in stringa dell'oggetto quando viene richiamato all'interno di una funzione di stampa. Un altro è `port_scan` che presenta il medesimo switch case mostrato nella sezione precedente. La differenza sostanziale risiede nei parametri passati, in quanto è stato aggiunto `ports_list` che determina la lista di porte da scansionare. Il prototipo finale della funzione sarà il seguente: `port_scan(self, port_arg, ports_list, verbose)`

A titolo d'esempio viene mostrato solo uno tra i metodi implementati, ovvero quello per svolgere un TCP SYN scan:

```

1 def tcp_syn_scan(self, ports_list: list, verbose: bool):
2     self.type = "TCP"
3
4     for port in ports_list:
5         if verbose:
6             print("\033[K", end="\r")
7             verbose_print(f"Testing {port}")
8
9         packet = IP(dst=self.ip) / TCP(dport=port, flags="S")
10        res = sr1(packet, timeout=3, verbose=0)
11
12        if res is None or (
13            res.sprintf("%ICMP.type%") == 3
14            and res.sprintf("%ICMP.code%") in [1, 2, 3, 9, 10, 13]
15        ):
16            self.ports[port] = "filtered"
17
18        else:
19            flag_res = res.sprintf("%TCP.flags%")
20
21            if flag_res == "RA":
22                pass
23            elif flag_res == "SA":
24                self.ports[port] = "open"

```

Inizialmente viene aggiornato il valore del tipo di scan in TCP. Vengono esaminate ad una ad una le porte contenute nella lista passata da parametro `ports_list` per svolgere individualmente lo scan. Il principio è simile al TCP SYN scan dell'host, ma all'interno della risposta vengono analizzate più approfonditamente le flag ricevute. Il pacchetto

creato è identico, ma se viene ricevuto un ICMP di tipo 3 specifico o non si ricevono pacchetti c'è la certezza che la porta sia filtrata. In caso contrario in base alle flag che vengono ricevute la porta può essere aperta, in caso di SYN-ACK, oppure chiusa, in caso di RESET.

Terminato lo scan è possibile richiamare un ulteriore metodo che raccoglie all'interno di una lista tutte le porte open o open/filtered. Questo è utile nel caso si volesse continuare con le fasi del vulnerability scan, avendo la certezza di interagire con porte che sono sicuramente disponibili. Il metodo implementato è il seguente:

```

1 def get_open_ports(self):
2     for key, value in self.ports.items():
3         if value != "closed" or value != "filtered":
4             self.open_ports.append(key)

```

Viene valutato lo stato della porta rilevata e se non è closed o filtered viene salvato il numero della porta all'interno della lista `open_ports`.

5.2.3 service_scan.py

All'interno del modulo è stata implementata una classe che contiene i seguenti tre elementi: un context per permettere la comunicazione con protocolli che utilizzano SSL/TLS, l'ip dell'host e infine una lista denominata `services` che contiene dizionari per descrivere tutte le informazioni dei protocolli/servizi. Questa lista viene popolata ogni volta che un protocollo è riconosciuto dallo scanner.

```

1 class ServiceScan:
2     # Defining self signed certificate for tls/ssl
3     context = ssl._create_unverified_context(ssl.PROTOCOL_TLS_CLIENT)
4     context.options &= ~ssl.OP_NO_SSLv3
5     context.minimum_version = 768
6     context.load_verify_locations(certifi.where())
7
8     def __init__(self, ip: str):
9         self.ip = ip
10        self.services = []

```

Nel modulo è presente una funzione `__str__` che permette di definire una formattazione testuale alternativa quando l'oggetto viene richiamato in una funzione di stampa.

Dopo aver creato l'oggetto `ServiceScan` è possibile eseguire la scansione dei protocolli e dei servizi presenti. Per svolgere le scansioni vengono utilizzate librerie di python sviluppate apposta per la comunicazione attraverso i protocolli specificati. Degli esempi sono: `ftplib`, `smtplib`, `telnetlib`, `ssl` ecc. Per capire quali protocolli sono presenti sulle porte vengono inviati messaggi utilizzando le funzioni presenti nelle librerie appena citate e se non vengono rilevati errori vuol dire che il protocollo di comunicazione utilizzato è stato

individuato. Successivamente è possibile interrogare ulteriormente il protocollo per avere informazioni sul servizio utilizzato e la versione, ma non sempre sono dati facilmente reperibili.

Per poter utilizzare tutti gli scan dei protocolli supportati viene usata una lista che contiene i nomi delle funzioni impiegate per l'individuazione. Sono presenti due liste principali, una per analizzare protocolli su porte UDP e una per analizzare porte TCP. Di seguito è riportata la lista TCP, in formato ridotto:

```

1  tcp_check = [
2      ftp_check,
3      ssh_check,
4      telnet_check,
5
6      ... <codice omesso> ...
7
8      # SSL protocols
9      ftps_check,
10     https_check,
11
12     ... <codice omesso> ...
13
14     # Undefined
15     undefined,
16 ]

```

Le funzioni sono ordinate nel seguente modo: quelle non sicure sono poste prima di quelle con SSL/TLS. Questo avviene perché la connessione tramite SSL/TLS alle volte è ammessa anche con protocolli non sicuri e può portare a falsi positivi. L'ultima funzione presente nella lista deve essere quella `undefined`, che applica dei valori di default alle porte in cui non si è riconosciuto il protocollo utilizzato.

Per poter usare la lista di funzioni riportata precedentemente viene impiegato il seguente spezzone di codice.

```

1 def tcp_scan(self, open_ports: list, verbose: bool):
2     for self.check in self.tcp_check:
3         self.check(self, open_ports, verbose)
4
5     # Out of for because I needed third argument
6     self.nfs_check(open_ports, verbose, "T")
7
8     # Clean line for verbose print
9     print("\033[K", end="\r")
10
11    # Sorts list by port
12    self.services = sorted(self.services, key=itemgetter("port"))

```

Viene eseguito un semplice for che scorre la lista di metodi e li richiama uno ad uno. All'interno di ogni funzione è presente un ciclo che scorre le porte aperte e le testa utilizzando librerie specifiche dei protocolli. Viene testata la risposta dopo l'invio di un messaggio strutturato secondo lo standard. Se avviene un errore di qualunque tipologia significa che il protocollo utilizzato non è giusto, altrimenti vuol dire che è stato individuato quello corretto. Se è possibile, viene interrogata ulteriormente la porta per ricavare informazioni sul servizio utilizzato e la versione.

Nell'implementazione nfs_check è esterna alla lista delle funzioni, poiché è necessario poter passare un ulteriore parametro rispetto alle restanti funzioni.

Nel codice sottostante è riportato a titolo di esempio un estratto della funzione ftp_check, in cui viene utilizzata la libreria ftplib per testare la presenza o meno del protocollo TCP sulla porta.

```

1 def ftp_check(self, open_ports: list, verbose: bool):
2
3     ... <codice omesso> ...
4
5     try:
6         ftp = FTP()
7         ftp.connect(host=ip, port=port, timeout=3)
8         ftp.quit()
9
10        # smtp also responds to this, so we need to verify the banner
11        s = socket.socket()
12        s.connect((ip, port))
13        banner = s.recv(1024)
14        banner = banner.decode("utf-8", errors="ignore")
15
16        if "FTP" in banner:
17            service["port"] = port
18            service["protocol"] = "FTP"
19            service["service"] = str(banner).strip()[4:]
20
21            self.services.append(service)
22            s.close()
23
24    except Exception as e:
25        pass
26    ...

```

Per semplificare la lettura sono state rimosse linee di codice che riguardano la stampa verbosa e lo scorrimento delle porte.

La struttura principale della comunicazione con la porta è inserita all'interno di un try -except che termina l'esecuzione del frammento di codice all'avvenire di un'eccezione, per poter poi passare alla scansione di un'altra porta. I primi passaggi svolti consistono nel creare un oggetto FTP e di tentare la connessione con l'host sulla porta in analisi. Se non

avvengono errori la porta viene chiusa e si passa a svolgere il codice che segue. Una volta accertata la correttezza, vengono inseriti i parametri all'interno di un dizionario `service` che verrà poi inserito nella lista `services` presente nell'oggetto `ServiceScan` appena creato.

Nel caso di FTP, anche una porta che utilizza protocollo SMTP è in grado di rispondere. Per questo motivo viene utilizzata una socket per collegarsi all'host e ricevere il banner iniziale, ovvero una stringa che descrive il servizio utilizzato sulla porta, per individuare il protocollo utilizzato.

Le successive funzioni seguono lo stesso principio di `ftp_check`. Tutte utilizzano le eccezioni per decretare la presenza o meno di un protocollo sulla porta. Alcune librerie permettono di recuperare il banner tramite una semplice funzione, altre non ne sono in grado. Al termine dello scorrimento della lista di funzioni verrà ritornato al `main` il dizionario `services` con tutte le informazioni necessarie per svolgere il prossimo step: l'esecuzione dei test.

5.2.4 results.py

Al suo interno è presente una classe `Results` utilizzata per immagazzinare dati durante l'esecuzione dei test sulle vulnerabilità. Ogni oggetto creato rappresenta le informazioni di una porta e il suo relativo servizio.

- `self.port`: porta su cui vengono eseguiti i test
- `self.prot`: protocollo individuato sulla porta
- `self.service`: servizio individuato sulla porta
- `self.prot_max_misconfigs`: numero totale di test presenti nel file dei protocolli
- `self.prot_max_auth_misconfigs`: numero totale di test con autenticazione presenti nel file di protocolli
- `self.serv_max_misconfigs`: numero totale di test con autenticazione presenti nel file dei servizi
- `self.serv_max_auth_misconfigs`: numero totale di test con autenticazione presenti nel file dei servizi
- `self.vuln_misconfigs`: numero di test che sono risultati positivi a vulnerabilità
- `self.vuln_auth_misconfigs`: numero di test autenticati che sono risultati positivi a vulnerabilità
- `self.unsafe_ver`: determina se una versione del servizio è vulnerabile

- self.unsafe_ver_cve: CVE che riguardano la versione specifica del servizio
- self.unsafe_tls: determina se la versione utilizzata del protocollo TLS/SSL è deprecata
- self.prot_auth: determina se è necessario svolgere i test dei protocolli con autenticazione
- self.serv_auth: determina se è necessario svolgere i test dei servizi con autenticazione

Queste variabili vengono inizializzate con valori di default. Durante l'esecuzione dei test vengono utilizzati dei metodi definiti all'interno della classe per variare il valore degli attributi. Sono dei semplici set che prendono come parametro il valore da aggiornare e al loro interno contengono un'assegnazione del dato all'attributo dell'oggetto specificato.

Sono presenti due metodi che definiscono la formattazione del testo: `__str__` e `__json__`. Il primo ritorna una stringa in cui gli attributi della classe sono stati formattati per rendere l'output leggibile dagli utenti. La seconda è principalmente utilizzata per trasformare la classe in un dizionario facilmente convertibile in JSON. Entrambi i metodi sono cruciali nella fase di scrittura dei file dei risultati.

5.2.5 execute_tests.py

Il modulo seguente è il fulcro del vulnerability scanner. Svolge le funzioni di test dei protocolli e dei servizi, controllo della versione di SSL/TLS e verifica del banner rispetto alle versioni vulnerabili riportate. Le informazioni sulle vulnerabilità sono contenute all'interno di file JSON, formattati seguendo un template specifico. Questi sono suddivisi in due categorie: uno per i protocolli e uno per i servizi. La struttura specifica è trattata nel capitolo 6.

Il modulo presenta una classe `ExecuteTests` che contiene l'ip della macchina analizzata e una lista `report` con tutti gli oggetti `Results` creati per ogni porta.

Il metodo principale che richiama tutte le funzionalità è `execute_tests()` ed al suo interno sono svolte le seguenti operazioni:

- apertura, se è presente, del file `<nome_prot>.test.json` nella cartella `tests/prot/`. Se il file JSON non viene trovato verrà creato un oggetto `Results` contenente informazioni di default e viene terminata la fase di test, passando ad un'altra porta. Nel caso in cui il file esistesse vengono recuperate le informazioni contenute, ovvero i comandi da inviare e le risposte da ricevere per valutarne la vulnerabilità. Successivamente viene creato un oggetto di tipo `Results` che contiene ip, protocollo e servizio, popolato in un secondo momento con le informazioni dei test svolti.

- viene richiamato `check_misconfigs()` che varia il comportamento in base alla presenza o meno del protocollo SSL/TLS. In caso fosse rilevato, richiama inizialmente `check_tls()` per verificare se le versioni utilizzate non siano deprecate e successivamente utilizza `test_ssl()` per svolgere i test. Se non fosse presente SSL/TLS viene utilizzato direttamente il metodo `test()`.
- Se all'interno del file JSON sono presenti test da eseguire su un protocollo con autenticazione, viene richiamata la funzione `try_login()` per richiedere le credenziali all'utente. Se vengono inserite, il metodo `check_misconfigs()` viene richiamato passandogli come parametro i test da svolgere con autenticazione.
- Se oltre al protocollo è stato individuato anche il servizio, vengono svolti gli identici passaggi precedenti, ma su di esso. Si tenta quindi l'apertura del file `<nome_serv>.test.json` per recuperare i contenuti. Se non è presente si passa alla prossima porta da testare, altrimenti si verifica la versione del servizio contro le versioni vulnerabili contenute nel file di test tramite `check_banner()`. Successivamente si ripetono le medesime operazioni svolte per compiere il test sui protocolli.

Di seguito vengono riportati i metodi menzionati precedentemente. Ognuno verrà analizzato in dettaglio per descrivere chiaramente le metodologie sviluppate per svolgere la fase di test.

La prima funzione che viene discussa è `check_misconfigs()`

```

1 def check_misconfigs(
2     self,
3     misconfigs,
4     verbose,
5     i_mis,
6     max_misconfigs,
7     port,
8     prot,
9     service,
10    results,
11    auth,
12    login_list=[],
13):
14    for name, info in misconfigs.items():
15        vuln = {}
16
17        ... <codice omesso> ...
18
19        # Complex ssl/tls test: establishes a connection and then sends
20        # a message and compares results
21        if "SSL" in prot:
22            vuln = self.test_ssl(name, info, self.ip, port, service,
23            login_list)

```

```

22         self.check_tls(service, results)
23
24     # Complex test: sends a message and compares the results
25     elif "recv" in info or "not_recv" in info:
26         vuln = self.test(name, info, self.ip, port, service,
27                           login_list)
28
29     # Simple test: checks if the port is open
30     else:
31         vuln["name"] = name
32         vuln["service"] = service
33         vuln["description"] = info["description"]
34         vuln["severity"] = info["severity"]
35
36         if vuln and auth:
37             results.set_auth_misconfigs(vuln)
38         elif vuln and not auth:
39             results.set_misconfigs(vuln)

```

La prima operazione svolta è lo scorrimento degli elementi del dizionario `misconfigs`, che sono stati recuperati dal file JSON e contengono il nome della vulnerabilità e le informazioni per svolgere i test. Il passaggio successivo consiste nella verifica della versione di SSL/TLS e viene svolto solo se il protocollo originale lo supporta. Il metodo `check_tls` è molto semplice:

```

1 def check_tls(self, service: str, results: Results):
2     if not ("TLSv1.3" in service or "TLSv1.2" in service):
3         results.unsafe_tls = True

```

All'interno del banner del servizio, oltre al nome e alla versione, è contenuto il protocollo TLS/SSL utilizzato. Se la versione contenuta nel banner non è uguale a "TLSv1.3" o "TLSv1.2", vuol dire che la tecnologia utilizzata è deprecata. In questo caso nell'oggetto `results` viene aggiornata la variabile `unsafe_tls` con il valore `True`.

Successivamente nel metodo `check_misconfigs()` possono essere svolte tre tipologie di test:

- Test complesso con SSL/TLS: viene creato un collegamento con il servizio tramite una socket sicura utilizzando il context specificato all'inizio. Successivamente per ogni test riportato nel file JSON vengono inviati dei comandi al protocollo e la risposta viene comparata con quella contenuta nel file. Se la vulnerabilità è presente i dati della stessa vengono inseriti all'interno dell'oggetto `results`, altrimenti viene eseguito il test successivo.
- Test complesso: la modalità di operazioni è uguale al test con SSL/TLS, ma in questo caso il socket creato verso il servizio non utilizza protocolli di sicurezza.

- Verifica se la porta è aperta: questo è il caso più semplice. Se in una vulnerabilità non sono stati trovati comandi da eseguire sul protocollo vuol dire che si sta semplicemente testando l'apertura della porta. In questo caso vengono subito aggiunti i dati di nome, servizio, descrizione e gravità all'interno della lista misconfigs di results.

Di seguito è presentato un estratto della funzione test(), che differisce da quella di test_ssl() solo per il modo in cui vengono creati i socket, dato che per utilizzare SSL/TLS è necessario l'utilizzo di secure socket.

```

1 def test(self, name: str, info: dict, ip: str, port: int, service: str,
2         login_list):
3     recv = None
4     not_recv = None
5
6     send_str = info["send"]
7     send_list = send_str.split("~~")
8
9     if "recv" in info:
10        recv = info["recv"]
11    elif "not_recv" in info:
12        not_recv = info["not_recv"]
13
14    try:
15        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16        sock.settimeout(5)
17        sock.connect((ip, port))
18
19        for message in login_list:
20            sock.send(message.encode())
21
22        # Sends all the commands to the server
23        for send in send_list:
24            # print(send)
25            sock.send(send.encode())
26            res = sock.recv(1024)
27            # print(res.decode())
28
29        # Compares the received message to the one in the json
30        if (
31            recv is not None
32            and re.search(recv, res.decode())
33            or not_recv is not None
34            and not re.search(not_recv, res.decode())
35        ):
36            vuln = []
37            vuln["name"] = name
38            vuln["service"] = service

```

```

38         vuln["description"] = info["description"]
39         vuln["severity"] = info["severity"]
40         return vuln
41
42     sock.close()
43
44 except TimeoutError:
45     pass

```

Alla funzione viene passata una stringa contenente tutti i comandi da inviare al server per testare la vulnerabilità. Questa viene separata in un lista `send_list` eliminando i caratteri `~~` presenti nella stringa.

Successivamente viene stabilita una connessione tramite socket verso il protocollo. Se è presente una stringa di autenticazione, questa viene inviata per prima (il funzionamento verrà spiegato in un secondo momento), altrimenti si passa direttamente all'invio dei comandi specificati in `send_list`. Dopo aver ricevuto la risposta del server si valuta la presenza di vulnerabilità tramite due modalità differenti. Se nei test è presente una variabile `recv`, allora vuol dire che la vulnerabilità è presente solo se la risposta e `recv` sono uguali; se invece è presente `not_recv`, la vulnerabilità è confermata solo se le due stringhe non sono uguali.

Quando viene rilevata la presenza di una vulnerabilità, il nome, la descrizione e la gravità vengono riportati all'interno di `vuln`. Questo verrà successivamente inserito all'interno della lista `misconfigs` dell'oggetto `results`.

Una volta svolti i test iniziali sul protocollo, si passa alla fase dei test con autenticazione. Per fare ciò è necessario richiedere all'utente le credenziali da inserire nel protocollo per autenticarsi. Questo è svolto dal metodo `try_login()`.

```

1 def try_login(self, prot, port, service, login) -> list:
2     # Asks the user max 3 times for the password
3     for i in range(3):
4         # Opens SSL socket
5         if "SSL" in prot:
6             sock = socket.create_connection((self.ip, port), timeout=3)
7             sock = ExecuteTests.context.wrap_socket(sock,
8                 server_hostname=self.ip)
9
10        # Opens simple socket
11    else:
12        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13        sock.settimeout(5)
14        sock.connect((self.ip, port))
15
16        # Asks the user for login info
17        print(f"{prot} - {service} username: ", end="")
18        username = input()

```

```

19     if username == "" and password == "":
20         login_list = []
21         return login_list
22     else:
23         login_str = login["send_str"].replace("_username_", username)
24     )
25         login_str = login_str.replace("_password_", password)
26
27     # Sends the login strings to the server
28     login_list = login_str.split("~~")
29     for message in login_list:
30         sock.send(message.encode())
31         res = sock.recv(1024)
32
33     # Checks the response of the server
34     if re.search(login["recv_str"], res.decode()):
35         sock.close()
36         return login_list
37     else:
38         sock.close()
39         print(f"Failed login {i + 1}/3")
40
41     sock.close()
42     login_list = []
43     print("Max login failed")
44     return login_list

```

Il parametro `login` passato alla funzione è una stringa contenuta nel file JSON di test. Al suo interno sono presenti dei comandi per svolgere l'autenticazione sul protocollo. Utilizza i valori `__username__` e `__password__` per indicare i punti in cui verranno inserite dal programma le credenziali dell'utente.

In base all'utilizzo di SSL/TLS viene creato un socket con contesto o senza. A questo punto il programma chiede all'utente l'inserimento dello `username` e della `password`. L'utente ha tre tentativi per inserire correttamente le credenziali, altrimenti viene saltato lo svolgimento dei test tramite autenticazione. L'utente può anche decidere di bypassare questa fase lasciando vuoti i due campi richiesti.

Per sicurezza, ad ogni inserimento di credenziali, il programma testa che l'autenticazione venga svolta in modo corretto. Per fare ciò valuta la risposta del server a seguito dell'autenticazione con una stringa `recv_str` contenuta nel dizionario `login`. Se i contenuti sono identici vuol dire che il `login` è andato a buon fine.

Una volta svolta l'autenticazione viene ripetuto il metodo `check_misconfigs`, ma con vulnerabilità che necessitano esclusivamente l'autenticazione. Queste sono definite all'interno del file JSON in una sezione separata dai test senza autenticazione. Se il `login` è stato saltato o non è andato a buon fine, si passa all'esecuzione dei test sul servizio.

Come per le scansioni sui protocolli, anche per i servizi viene tentata l'autenticazione se è presente il parametro `login` nel file di test. I passaggi successivi sono pressoché identici a quelli svolti per l'individuazione di vulnerabilità con autenticazione per i protocolli.

Per quanto riguarda i servizi, oltre ai metodi precedentemente riportati, è anche presente la funzione `check_banner()`.

```

1 def check_banner(self, service: str, vuln_serv_version: dict,
2                  results: Results):
3     for version, cve in vuln_serv_version.items():
4         if version in service:
5             results.unsafe_ver = True
6             results.unsafe_ver_cve = cve

```

Questa funzione si occupa di verificare che la versione del servizio utilizzata non sia vulnerabile. Per fare ciò compara la versione individuata dall'applicativo con quelle presenti nel file JSON del servizio. Se sono identiche, l'attributo `unsafe_ver_cve` viene impostato a `True`, per specificare che la versione utilizzata è vulnerabile, e le CVE relative vengono inserite all'interno dell'oggetto `results`.

Terminati i test verso tutte le porte, all'interno di `report` saranno presenti tutti i dati inerenti ai protocolli, servizi e ai test svolti. Questo elemento verrà utilizzato nella fase di scrittura dei risultati.

5.3 Utilities

In questo package sono raccolti tutti i moduli inerenti alla comunicazione con l'utente da riga di comando e alla scrittura dei risultati su diverse tipologie di file. È una libreria di supporto che permette all'utente di interfacciarsi in modo semplice con le funzionalità dell'applicativo e di visionare il risultato delle scansioni.

5.3.1 parser.py

Utilizzando la libreria `argparse` viene semplificata la gestione dei parametri passati da linea di comando e permette creazione di un manuale accessibile tramite la flag `-h` o `--help`. Il manuale è simile a quelli già presenti in altri programmi da linea di comando ed è scritto nel seguente modo:

```

usage: main.py [-h] [-v] [-nt] [-hs HOST_SCAN]
                [-ps PORT_SCAN] ports host

Agent for Advanced Network Protocol Verification.
This program needs sudo privileges to run.

positional arguments:

```

```

ports           Single port [x], multiple ports [x,y,z],
               port range [x:y] to scan or all ports [all]
host           Host to scan using ipv4 address

options:
-h, --help      Show this help message and exit
-v, --verbose   Increase output verbosity
-nt, --no_tests Scans the target for services but doesn't
                 execute a vulnerability scan
-hs, --host_scan HOST_SCAN
                 Host scan to execute: [p]ing, [s]yn, [a]ck,
                 [u]dp (ping scan will be used by default)
-ps, --port_scan PORT_SCAN
                 Port scan to execute: [c]onnect, [s]yn, [f]in,
                 [n]ull, [x]mas, [u]dp (connect scan will
                 be used by default)

```

Da questo manuale si evince che gli unici parametri da passare obbligatoriamente al programma sono l'host e le porte. Per definire su quali porte svolgere la scansione sono presenti 4 metodi: specificare una singola porta, specificare una lista di porte separate da virgola, specificare un range di porte indicandone il numero iniziale e quello finale diviso da due punti o scegliere di scansionare tutte le porte del range utilizzando il valore "all". Per l'host è possibile specificare un indirizzo di tipo IPv4 oppure utilizzare "localhost" per svolgere la scansione sul dispositivo corrente.

I parametri facoltativi possono essere selezionati dall'utente per specificare le modalità di scan da svolgere in caso di necessità. La prima opzione che può essere selezionata è -nt o --no_tests, che permette la scansione dei dispositivi, ma termina l'esecuzione del programma prima dello svolgimento dei test sulle vulnerabilità. Questa modalità può essere utile per individuare porte e protocolli sulla rete o verificare preventivamente i servizi da analizzare. Il parametro -hs o --host_scan permette l'utilizzo di modalità alternative di scan dell'host. Basterà utilizzare i caratteri predefiniti "p","s","a" ed "u" per indicare la tipologia utilizzata per il riconoscimento dell'host. Se non viene specificato nulla, il programma eseguirà la scansione di tipo ping. L'ultimo parametro richiamabile è -ps o --port_scan che permette, anche in questo caso, la selezione della modalità di scansione delle porte da parte dell'utente. I caratteri da specificare possono essere "c", "s", "f", "n", "x" o "u" e se non viene passato nessun parametro viene utilizzata la scansione di tipo connect.

5.3.2 terminal_colors.py

Modulo molto semplice che permette la colorazione dell'output su riga di comando. Il testo stampato può apparire in 4 colorazioni:

- bianco: output normale che mostra i passaggi della scansione ed i risultati.

- verde: conferma un risultato positivo della scansione, viene utilizzato principalmente per comunicare la disponibilità dell'host alla comunicazione.
- rosso: avvisa quando avvengono errori, per esempio se l'host è irraggiungibile, le porte non sono state inserite correttamente o se non vengono utilizzati i privilegi di amministratore quando viene eseguita l'applicazione.
- azzurro: output attivabile tramite il parametro verbose. Mostra il progresso delle scansioni e dei test. È un testo effimero che viene sovrascritto dai risultati effettivi per non lasciare tracce sull'output e evitare confusione.

Per poter stampare in modo "effimero" è necessario utilizzare il carattere di escape \r, anche detto carriage return, che sposta il cursore all'inizio della linea corrente e sovrascrive tutti i caratteri precedentemente scritti. Se viene stampato qualcosa più corto di quello che era già presente, le parti finali di testo scritte precedentemente rimarranno visibili. Per questo alla fine di ogni stampa viene richiamato il comando `print("\033[K", end="\r")`, che ripulisce tutta la riga e permette ogni volta una scrittura pulita.

5.3.3 write_results.py

La funzione principale che viene richiamata per la scrittura dei file è `write_result()`. Il primo passo che compie è la creazione di cartelle, se non sono ancora presenti, per contenere i file con i risultati. La cartella principale è `res/`, che viene creata la prima volta in cui il programma viene eseguito. All'interno, per ogni scansione, vengono generate delle cartelle nominate `Result_<timestamp>/`, al cui interno verranno salvati i risultati. Per salvare le immagini utilizzate dal file HTML viene creata in aggiunta una cartella `img/`. Successivamente alla creazione delle cartelle vengono richiamate tre funzioni per la scrittura di file: `log_result()`, `json_result()` e `html_result()`. Ricevono come parametro una lista `report` che contiene tutti gli oggetti `results`.

`log_result` prende i risultati ottenuti dalle fasi precedenti e crea un file di testo contenente tutte le informazioni in formato leggibile. Questo avviene grazie al metodo `__str__()` presente nel modulo `results.py`. Basterà iterare tra gli oggetti contenuti nella lista di `report` e inserirli nella funzione `print()`. Questo richiamerà automaticamente `__str__()` che provvederà a formattare l'output nel modo corretto.

`json_result()` provvederà a creare un file json utilizzabile per il parsing da parte di altri programmi o strumenti automatizzati. Come per `log_results()`, la funzione si basa sul metodo `__json__()` presente in `results.py` per formattare l'oggetto `results` in un dizionario facilmente convertibile in json. Vengono anche riportate all'inizio del file tutte le informazioni riguardante l'host come l'ip e il timestamp dello scan.

`html_results()` è la funzione più lunga e complessa tra le tre. L'obiettivo è quello di partire dalle informazioni ricevute tramite `report` e successivamente generare un file html

contenente tutti i risultati. Ogni protocollo e servizio avrà una sezione dedicata visitabile tramite un menù contenente tutte le porte scoperte. All'interno delle sezioni saranno riportate le vulnerabilità trovate accompagnate da un grafico per visualizzarne meglio le informazioni. Per la generazione del file viene utilizzata la libreria `jinja2` di python che permette l'utilizzo di template html da popolare tramite variabili passate alla funzione `template.render()`. Per generare i grafici a torta che rappresentano il numero e la gravità delle vulnerabilità è stata impiegata la libreria `matplotlib`.

Capitolo 6

Scrittura dei test

I test utilizzati nella fase di individuazione delle vulnerabilità sono contenuti in file JSON. Al loro interno sono presenti descrizioni e comandi da eseguire sugli host per verificare la presenza di criticità. I file sono contenuti all'interno di `test/` e sono suddivisi a loro volta all'interno di due sottocartelle: `test/prot/` per i test sui protocolli e `test/serv/` per i test sui servizi. Questa suddivisione è nata a seguito della necessità di limitare le ridondanze e rendere più brevi i file. La struttura dei JSON deve seguire un template molto rigido, per permettere la corretta esecuzione del programma e per evitare di generare errori.

6.1 Struttura dei file

Test sui protocolli

Per poter scrivere i test da svolgere su un determinato protocollo è necessario creare all'interno della cartella `test/prot/` un file nominato `<protocol>.test.json`. Al posto del segnaposto `<protocol>` sarà necessario specificare il protocollo che si vuole testare. All'interno del file è necessario ricopiare il template sottostante nella sua interezza, per garantire la corretta esecuzione dei test. In un secondo momento sarà possibile compilare i campi del JSON con le informazioni necessarie alle scansioni.

```
{  
    "vulns": {},  
    "login": "",  
    "auth_vulns": {},  
    "serv_names": []  
}
```

Il file di JSON deve presentare obbligatoriamente questi 4 elementi:

- `vulns` e `auth_vulns`: oggetti JSON contenenti l'insieme di vulnerabilità da testare. Sono suddivisi in base a test normali e quelli in cui è necessario un'autenticazione.

- login: serie di comandi utilizzati per svolgere l'autenticazione sul protocollo e di conseguenza poter svolgere i test che necessitano autenticazione.
- serv_names: lista di servizi che si basano sul protocollo preso in considerazione.

Test sui servizi

I file che riguardano i test da svolgere sui servizi vengono inseriti all'interno della cartella `test/serv/`. Il JSON dovrà essere nominato `<service>.json`, riportando al posto di `<service>` il servizio desiderato. Come per il file di test sui protocolli è necessario compilare un template, ma che presenta leggere differenze, come riportato nel JSON sottostante. Successivamente sarà poi possibile popolare i singoli campi.

```
{
  "vulns": {},
  "login": {},
  "auth_vulns": {},
  "vuln_serv_version": {}
}
```

- vulns e auth_vulns: identico al file dei protocolli, un oggetto che contiene le informazioni per svolgere i test, anche in questo caso suddivisi in normali e autenticati.
- login: stesso principio del file dei protocolli, serie di comandi per autenticarsi sul servizio e svolgere test con l'autenticazione.
- vuln_serv_version: lista contenente il numero delle versioni vulnerabili che sono state individuate per il servizio e i CVE associati.

6.2 Elementi dei file

`misconfigs/auth_misconfigs`

All'interno di questi due oggetti sono riportati gli elementi per poter svolgere i test sui protocolli/servizi. Ogni test presenta un nome e, in base al tipo di verifica che si vuole svolgere, possono comparire descrizioni, comandi da inviare e ricevere e un indice di gravità. Di seguito sono riportate due modalità per poter condurre un test: una semplice per verificare la presenza di una porta aperta e una più complessa in cui è necessario specificare i comandi da inviare e ricevere per testare le vulnerabilità.

Verifica della presenza di porte aperte:

```

1 "IS OPEN": {
2   "description": "Telnet is an old network protocol that provides
      insecure access to computers over a network. Due to security
      vulnerabilities, its usage is not recommended, and more secure
      alternatives like SSH are preferred.",
3   "severity": "high"
4 }
```

Verifica della presenza di vulnerabilità inviando comandi:

```

1 "ANONYMOUS LOGIN ENABLED" :{
2   "description": "Anonymous login is enabled, everyone can access the
      service",
3   "send": "\n~~USER anonymous\n~~PASS\n",
4   "recv": "230",
5   "severity": "high"
6 },
```

In entrambi gli oggetti sono presenti o meno i seguenti elementi:

- **description**: descrizione della vulnerabilità, ovvero le motivazioni per cui la vulnerabilità può essere un pericolo per il dispositivo e la rete.
- **severity**: gravità della vulnerabilità, definisce la pericolosità di una vulnerabilità se è presente sulla macchina. Può essere "low", "medium" o "high".
- **send**: comandi da inviare al protocollo o servizio per testare la vulnerabilità. Per inviare più di un comando è necessario utilizzare i caratteri separatori ~~.
- **recv** o **not_recv**: recv specifica il messaggio da ricevere per confermare la vulnerabilità. not_recv indica che se non si riceve il messaggio specificato, la vulnerabilità è confermata.

login

Contiene le informazioni necessarie per svolgere l'autenticazione verso il protocollo/servizio interessato. Vengono utilizzate delle stringhe di template che vengono successivamente popolate a runtime dalle credenziali inserite dall'utente.

```

1 "login": {
2   "send_str": "\n~~USER _username_\n~~PASS _password_\n",
3   "recv_str": "230 Login successful."
4 },
```

- `send_str`: contiene comandi da inviare al protocollo/servizio per poter svolgere l'autenticazione. I caratteri utilizzati per separare i comandi e inviare più stringhe sono `~~`. Inoltre `_username_` e `_password_` sono utilizzate come credenziali provvisorie che vengono popolate durante l'esecuzione dei test, a seguito dell'interrogazione dell'utente.
- `recv_str`: la stringa da ricevere per confermare la corretta autenticazione dell'utente.

`serv_names`

Una lista utilizzata nel JSON dei protocolli per specificare i servizi che già presentano file di test all'interno della cartella `test/serv/`. Se un servizio presente in questa lista è stato individuato durante la scansione, viene aperto il file di test corrispondente e viene eseguito il testing del servizio.

```
1 "serv_names": [
2   "vsftpd",
3   "proftpd"
4 ]
```

`vuln_serv_version`

Oggetto presente nel file di test dei servizi. Ad ogni versione vulnerabile del servizio è riportata una lista contenente i rispettivi CVE.

```
1 "vuln_serv_version": {
2   "2.2.8": [
3     "https://www.cve.org/CVERecord?id=CVE-2008-2364",
4     "https://www.cve.org/CVERecord?id=CVE-2022-40309"
5   ]
6 }
```

La seguente è stata una breve descrizione dell'impostazione del JSON per i test. Un esempio completo è consultabile all'appendice B, che illustra la struttura dei test di FTP e del servizio vsFTPd.

Capitolo 7

Risultati

Durante l'esecuzione dell'applicativo l'utente può scegliere di stampare risultati intermedi. Per fare ciò è necessario specificare la flag `-v` o `--verbose`. Questo permette anche di mostrare l'avanzamento delle operazioni, utilizzando un testo effimero che viene sovrascritto dagli output definitivi,

Successivamente, una volta terminata la fase di testing dei protocolli, il programma procede a generare file contenenti i risultati ottenuti durante l'esecuzione del vulnerability scanner. Come precedentemente descritto nella sezione 5.3.3, i file generati dall'applicativo sono tre: un TXT, un JSON e un HTML. La scelta di questi tre formati si basa sull'idea di fornire il maggior numero di file all'utente per consentirgli di processare i dati nella maniera che più ritiene soddisfacente. Il formato TXT fornisce un file molto leggero e formattato in modo tale da poter essere letto rapidamente. JSON viene utilizzato principalmente per essere gestito da altri programmi che ne leggono il contenuto e lo processano. Infine l'HTML offre un risultato grafico che è semplice da navigare ed intuitivo, presentando dei diagrammi a torta che facilitano la comprensione.

Per mostrare i risultati dell'applicazione nel modo più esaustivo possibile sono stati utilizzati due host: una macchina virtuale che presenta servizi deprecati ricolmi di vulnerabilità e uno stack docker contenente la maggior parte dei servizi riconosciuti dallo scanner.

La macchina virtuale utilizzata è Metasploitable¹, una VM basata su Ubuntu che presenta gravi fallo di sicurezza e configurazioni errate. Viene impiegata principalmente come strumento di allenamento per aspiranti penetration tester, ma in questo caso è utilizzata per testare la fase di esecuzione dei test del vulnerability scanner. È molto utile perché contiene vulnerabilità banali che possono essere facilmente testate.

Lo stack docker è principalmente utilizzato per il riconoscimento dei protocolli, in

¹<https://docs.rapid7.com/metasploit/metasploitable-2/>

quanto quasi tutte le immagini dei servizi sono aggiornate all'ultima versione e non presentano vulnerabilità note. In questo modo è possibile testare il riconoscimento dei protocolli e dei servizi da parte dello scanner. I servizi contenuti sono i seguenti: FTPS, SMTP, POP, IMAP, SMTPS, POPS, IMPAS, SMB, DNS, NFS, MQTT.

7.1 Risultati intermedi e procedimento delle operazioni

Utilizzando il flag `-v` o `--verbose` vengono riportati sull'output risultati intermedi al termine di ogni fase del vulnerability scan. Le informazioni mostrate per ogni step sono le seguenti:

- Stato dell'host
- Porte trovate e il loro stato
- Servizi e versioni individuate sulle porte

```
--- Checking host ---
Host is up

--- Checking ports ---
PORT      STATUS
21        open
22        open
23        open
25        open
53        open
80        open

--- Checking protocols and services ---
PORT      PROTOCOL          SERVICE
21        FTP                (vsFTPd 2.3.4)
22        SSH                SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1
23        TELNET             undefined
25        SMTP               metasploitable.localdomain ESMTP Postfix (
Ubuntu)
53        DNS                undefined
80        HTTP               Apache/2.2.8 (Ubuntu) DAV/2

--- Testing protocols and services ---

--- Asking for protocols credentials ---
FTP - (vsFTPd 2.3.4) username: anonymous
FTP - (vsFTPd 2.3.4) password:
```

```
----- RESULTS -----
PORT      PROTOCOL      SERVICE
21        FTP           (vsFTPD 2.3.4)
|
| ----- VERSION CHECK -----
\--- THIS SERVICE VERSION IS VULNERABLE AND NEEDS TO BE UPDATED!
|   reference:
|     - CVE-2011-2523: https://www.cve.org/CVERecord?id=CVE-2011-2523
|
| ----- VULNERABILITIES -----
\--- ANONYMOUS LOGIN ENABLED
|   description: Anonymous login is enabled, everyone can access the
|     service
|   severity: high
|
| ----- AUTHENTICATED VULNERABILITIES -----
\--- BOUNCE ATTACK
|   description: If not correctly configured the PORT command can use
|     the victim machine to request access to port indirectly. This can be
|     used to scan hosts ports discretely.
|   severity: medium
...
```

Al termine vengono riportati a schermo i risultati finali con tutte i protocolli e le relative vulnerabilità, identici a quelli descritti nei file TXT che discuteremo successivamente.

Un'altra funzionalità che viene attivata tramite i flag di tipo verbose è la stampa dello stato di avanzamento delle operazioni. Questo avviene tramite la scrittura a schermo di un testo azzurrino che viene sovrascritto ad ogni cambio di operazione. Questo descrive in quale punto dell'analisi delle porte, dei protocolli o delle vulnerabilità si trova lo scanner. Esempio di testo effimero durante le scansioni di protocolli e servizi:

```
--- Checking protocols and services ---
Scanning 80 for FTP
```

Esempio di testo effimero durante i test:

```
--- Testing protocols and services ---
Scanning 21 with FTP - (vsFTPD 2.3.4) using BACKDOOR COMMAND EXECUTION
[1/1]
```

7.2 Riconoscimento dei protocolli e servizi

Per svolgere una semplice scansione dei protocolli e servizi, senza passare dalla fase di esecuzione dei test, viene abilitata la flag `-ns`. L'host scansionato è "localhost", in quanto i container docker sono stati fatti partire sulla macchina dove risiede l'applicativo sviluppato. Le porte analizzate sono comprese tra 1 e 10000, in modo tale da permettere la scansione di tutte le porte interessate. Il comando completo utilizzato è il seguente:

```
sudo ./main.py 1:10000 localhost -ns
```

Risultati della scansione di protocolli e servizi

PORT	PROTOCOL	SERVICE
21	FTP-SSL	(vsFTPD 3.0.3) - TLSv1.3
22	SSH	SSH-2.0-OpenSSH_9.9
25	SMTP	/172.18.0.2 GreenMail SMTP Service v2.1.3 ready
110	POP	+OK POP3 GreenMail Server v2.1.3 ready
143	IMAP	IMAP4REV1
445	SMB	undefined
465	SSL-TLS	TLSv1.3
993	IMAP-SSL	IMAP4REV1 - TLSv1.3
995	POP-SSL	+OK POP3 GreenMail Server v2.1.3 ready - TLSv1.3
1883	MQTT	MQTTv311
2049	NFS	NFSv4
5380	HTTP	undefined
8000	HTTP	Apache/2.4.62 (Debian)
8081	HTTP	undefined
9443	HTTPS	undefined - TLSv1.3

Il risultato finale mostra per ogni porta il protocollo utilizzato e, se è stato riconosciuto, il servizio. Nel caso in cui non è stato in grado di individuare la versione, verrà utilizzato `undefined` come valore di default. Nei protocolli messi in sicurezza con SSL/TLS viene anche specificata la versione del protocollo di sicurezza.

7.3 Riconoscimento delle vulnerabilità

In questa sezione vengono proposti i tre formati di file che vengono prodotti durante l'esecuzione del vulnerability scanner. Rispetto alla sezione precedente verrà svolta la fase di esecuzione dei test per la rilevazione delle vulnerabilità. In questo caso viene utilizzata la macchina virtuale Metasploitable 2 per simulare con efficacia i risultati derivati da una scansione di un host vulnerabile. Le porte analizzate sono comprese tra la 1 e la 1000 e l'host in questione è raggiungibile all'IP 192.168.100.175, all'interno di una sottorete creata ad hoc per la macchina virtuale. In questo caso il comando utilizzato è:

```
sudo ./main.py 1:1000 192.168.100.175
```

Risultati TXT

```
##### RESULTS FOR 192.168.100.175 #####
PORT      PROTOCOL      SERVICE
-----
21        FTP           (vsFTPd 2.3.4)
|
| ----- VERSION CHECK -----
|\__ THIS SERVICE VERSION IS VULNERABLE AND NEEDS TO BE UPDATED!
|   reference:
|     - CVE-2011-2523: https://www.cve.org/CVERecord?id=CVE-2011-2523
|
| ----- VULNERABILITIES -----
|\__ ANONYMOUS LOGIN ENABLED
|   description: Anonymous login is enabled, everyone can access the
|     service
|   severity: high
|\__ BACKDOOR COMMAND EXECUTION
|   description: Allows users to leverage a backdoor to make a command
|     execution.
|   severity: high
|
| ----- AUTHENTICATED VULNERABILITIES -----
|\__ BOUNCE ATTACK
|   description: If not correctly configured the PORT command can use
|     the victim machine to request access to port indirectly. This can be
|     used to scan hosts ports discretely.
|   severity: medium
22        SSH            SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1
|
| ----- VERSION CHECK -----
|\__ The service version is not vulnerable.
```

```

|
| ----- VULNERABILITIES -----
|\__ No tests found for this protocol
|
| ----- AUTHENTICATED VULNERABILITIES -----
|\__ No tests found for this protocol

23      TELNET      undefined
|
| ----- VERSION CHECK -----
|\__ The service version is not vulnerable.
|
| ----- VULNERABILITIES -----
|\__ IS OPEN
|   description: Telnet is an old network protocol that provides
|   insecure access to computers over a network. Due to security
|   vulnerabilities, its usage is not recommended, and more secure
|   alternatives like SSH are preferred.
|   severity: high
|
| ----- AUTHENTICATED VULNERABILITIES -----
|\__ No tests found for this protocol
...

```

Il file di testo generato riporta i risultati delle scansioni raggruppati secondo le porte individuate. Ogni sezione contiene le informazioni sul protocollo utilizzato, la versione e le vulnerabilità che sono state individuate dalle scansioni svolte. Se la versione del servizio utilizzata è vulnerabile, all'interno di VERSION CHECK viene sollecitato un aggiornamento immediato del servizio e il CVE di riferimento. VULNERABILITIES e AUTHENTICATED VULNERABILITIES contengono le vulnerabilità scovate dai test, con annessi il nome, una descrizione del funzionamento e un indice di pericolosità. Inoltre, se il servizio utilizza SSL/TLS, viene mostrata un'ulteriore sezione chiamata SSL/TLS CHECK che indica se il protocollo di sicurezza utilizzato è supportato o deprecato. Nel secondo caso invita l'utente ad un aggiornamento.

Risultati JSON

```
{
  "ip": "192.168.100.175",
  "timestamp": "2025-08-18_11:44:22",
  "services": [
    {
      "port": 21,
      "protocol": "FTP",
      "service": "(vsFTPD 2.3.4)",
      "unsafe_version": true,
      "unsafe_version_cve": [
        ...
      ]
    }
  ]
}
```

```

        "https://www.cve.org/CVERecord?id=CVE-2011-2523"
    ],
    "vulnerabilities": [
        {
            "name": "ANONYMOUS LOGIN ENABLED",
            "service": "(vsFTPD 2.3.4)",
            "description": "Anonymous login is enabled, everyone can access the service",
            "severity": "high"
        },
        {
            "name": "BACKDOOR COMMAND EXECUTION",
            "service": "(vsFTPD 2.3.4)",
            "description": "Allows users to leverage a backdoor to make a command execution.",
            "severity": "high"
        }
    ],
    "auth_vulnerabilities": [
        {
            "name": "BOUNCE ATTACK",
            "service": "(vsFTPD 2.3.4)",
            "description": "If not correctly configured the PORT command can use the victim machine to request access to port indirectly. This can be used to scan hosts ports discretely.",
            "severity": "medium"
        }
    ]
},
{
    "port": 22,
    "protocol": "SSH",
    "service": "SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1",
    "unsafe_version": false,
    "unsafe_version_cve": [],
    "vulnerabilities": [],
    "auth_vulnerabilities": []
},
...

```

Il file JSON presenta le medesime informazioni riportate nel file TXT, ma formattate in modo tale da poter essere facilmente interpretate da altri programmi. All'interno dei risultati è presente l'ip del dispositivo scansionato, un timestamp che stabilisce ora e data dell'esecuzione dei test e una lista dei protocolli individuati. All'interno di quest'ultima sono riportate informazioni sulla porta, il protocollo, il servizio e tutte le vulnerabilità scoperte durante il vulnerability scan. Come precedentemente annunciato questo file può essere utilizzato all'interno di altri programmi grazie al grande supporto che ha il formato.

Risultati HTML

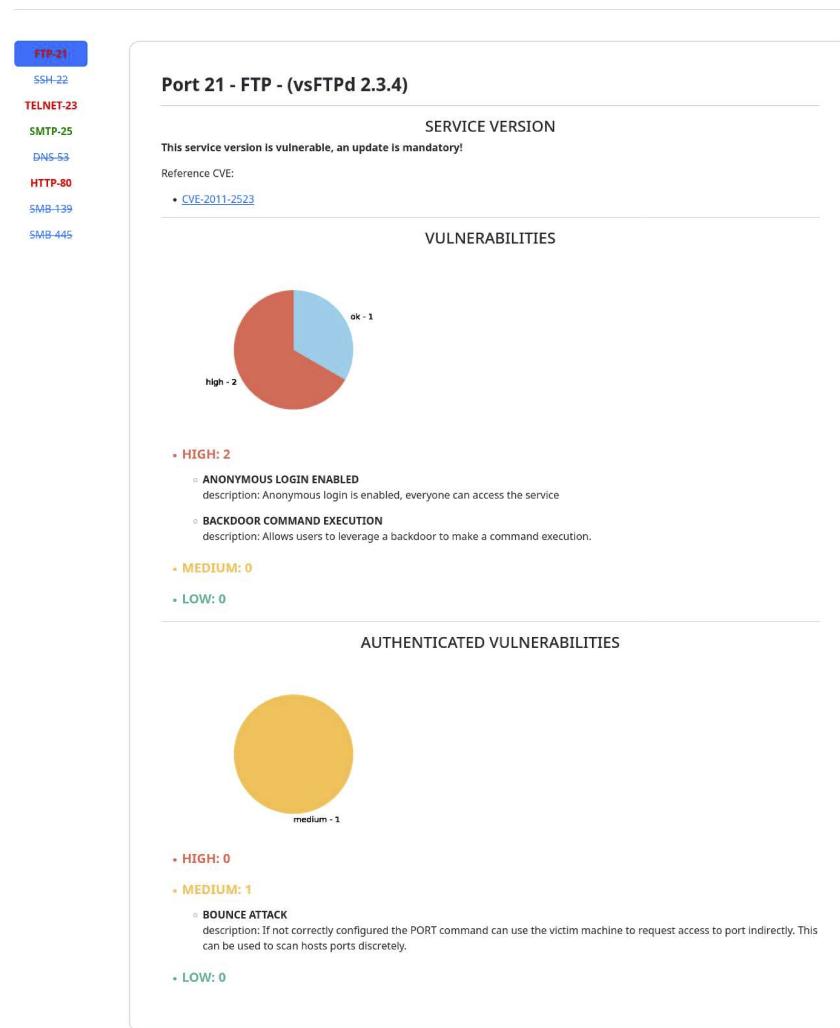


Figura 6: Risultati del riconoscimento dei protocolli in formato HTML

Il risultato HTML è utilizzato per esprimere i dati del report in modo semplice e visivo. In questo modo chi visiona i risultati ha più facilità ad orientarsi tra i dati e le informazioni. La pagina principale presenta come titolo l'IP della macchina scansionata e due sezioni sottostanti. La parte sinistra contiene dei bottoni che rappresentano tutti i protocolli individuati e le rispettive porte, colorati in base ai risultati dei test. Un bottone barrato significa che non sono stati trovati test da svolgere, un bottone rosso indica la presenza di vulnerabilità all'interno dei protocolli o dei servizi e infine il bottone verde indica che tramite

i test svolti non sono state individuate problematiche. A seguito del click su un bottone, nella sezione più grande a destra vengono mostrati i risultati in dettaglio. Inizialmente viene dichiarata se la versione utilizzata dal servizio è vulnerabile e successivamente, se presente, un'indicazione sul protocollo SSL/TLS utilizzato. La parte finale dei risultati riguarda le vulnerabilità scovate tramite i test. Sono suddivise in VULNERABILITIES e AUTHENTICATED VULNERABILITIES ed entrambe presentano un grafico che descrive il numero e la gravità delle vulnerabilità trovate.

Capitolo 8

Conclusioni

L'applicativo sviluppato è in grado di svolgere correttamente le principali funzionalità presenti in un qualsiasi vulnerability scanner. È in grado di riconoscere correttamente i servizi presenti sulle macchine di prova ed è anche capace di fornire un report dettagliato di vulnerabilità in seguito a test svolti sulle porte individuate. La modularità con cui è stato scritto il codice permette di ampliare in modo molto semplice le capacità dell'applicativo, permettendo aggiornamenti e miglioramenti in un tempo ridotto.

8.1 Sviluppi futuri

Per poter trasformare questo progetto in un vero e proprio software utilizzabile è necessario implementare nuove funzionalità e nuovi test. L'applicativo ora è in grado di riconoscere 12 tra i principali protocolli e servizi utilizzati nelle comunicazioni di tutti i giorni e riesce a svolgere test su una piccola parte di configurazioni e vulnerabilità. Nelle sezioni seguenti sono stati riportate delle idee per poter rendere questo applicativo sempre più completo e funzionale.

8.1.1 Scrittura di test automatizzata

La scrittura dei test per ora è svolta manualmente e può essere molto onerosa, in quanto è necessario ricercare nuove vulnerabilità e riportarle all'interno dei rispettivi file JSON. Per poter affrontare questo problema si potrebbe implementare uno scraper che analizza script e test di altri scanner, per poterli adattare successivamente in file JSON interpretabili dall'applicativo. In questo modo sarebbe possibile ridurre il tempo passato nella scrittura dei test, impiegando le forze sullo sviluppo di altre funzionalità.

8.1.2 Ampliamento dei protocolli supportati

I protocolli supportati fin ora sono un numero molto ridotto rispetto a quelli utilizzati da grandi infrastrutture di rete. Sarebbe dunque opportuno implementare il riconoscimento di altri protocolli, in modo tale da poter svolgere un vulnerability scan più dettagliato e completo. Grazie alla modularità del codice questo problema è facilmente risolvibile, in quanto basterà implementare nuove funzioni di riconoscimento utilizzando un template già presente nel codice. Inserendo il nome della funzione all'interno di una lista specifica, questa verrà eseguita insieme a tutte le altre funzioni per lo scan di protocolli e servizi.

8.1.3 Scansione di più host contemporaneamente

L'applicativo nello stato attuale permette la scansione di un solo host alla volta. Una soluzione potrebbe essere quella di implementare un'analisi parallela di più dispositivi nella rete. In questo modo è possibile far partire un'unica scansione, senza ulteriori interventi umani, in modo tale da poter coprire l'intera infrastruttura di un'azienda. Per implementare ciò è possibile utilizzare i thread che, lavorando in parallelo sulle stesse operazioni, ma su host diversi, possono velocizzare di molto la verifica degli host e il riconoscimento dei protocolli.

8.1.4 Esecuzione automatica e periodica

Il programma creato è altamente dipendente dall'uomo, in quanto è necessario richiamarlo da linea di comando ogni volta che si vuole eseguire una scansione. Per ovviare a questo problema sarebbe possibile sviluppare uno script che richiama periodicamente l'applicativo per eseguire gli scan, in modo tale da verificare lo stato della rete di giorno in giorno o di settimana in settimana. Un'ulteriore funzionalità potrebbe essere quella di avvisare tramite mail o messaggio l'amministratore di rete ogni qual volta vengano rilevate nuove vulnerabilità sulle macchine. Essendo l'applicativo scritto in python, l'implementazione di questa funzionalità potrebbe essere fatta in modo molto semplice.

8.1.5 Evasione dei firewall

Un'infrastruttura di rete a livello aziendale presenta sicuramente un firewall per la difesa della rete. Questo può essere un ostacolo per l'applicativo, poiché alcuni pacchetti potrebbero essere bloccati per via di alcune regole applicate. Di conseguenza è possibile implementare delle metodologie aggiuntive che permettano l'evasione dei filtri, in modo tale da poter svolgere uno scan senza troppe problematiche e allo stesso tempo verificare anche l'efficacia del firewall stesso. Per fare ciò è possibile implementare una nuova modalità di invio di pacchetti, in modo da renderli più discreti e difficilmente individuabili dagli strumenti di difesa.

Bibliografia

- [1] Karen. Scarfone, National Institute of Standards, and Technology (U.S.). *Technical guide to information security testing and assessment : recommendations of the National Institute of Standards and Technology.* NIST special publication ; 800-115. Computer security. U.S. Dept. of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, 2008.
- [2] David J. Wetherall Andrew S. Tanenbaum, Nick Feamster. *Computer Networks.* Pearson, 2021.
- [3] J. Postel. Internet Control Message Protocol. RFC 792, September 1981.
- [4] Carl Sunshine Vinton Cerf, Yogen Dalal. Specification of Internet Transmission Control Program. RFC 675, December 1974.
- [5] J. Postel. User Datagram Protocol. RFC 768, August 1980.
- [6] J. Reynolds J. Postel. File Transfer Protocol. RFC 959, October 1985.
- [7] J. Reynolds J. Postel. Telnet Protocol Specification. RFC 854, May 1983.
- [8] Dr. John C. Klensin. Simple Mail Transfer Protocol. RFC 5321, October 2008.
- [9] P. Mockapetris. Domain names - implementation and specification. RFC 1035, November 1987.
- [10] Cloudflare. Che cos'è il dns? | come funziona il dns, 2025.
- [11] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP/1.1. RFC 9112, June 2022.
- [12] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. RFC 9110, June 2022.
- [13] Dr. Marshall T. Rose and John G. Myers. Post Office Protocol - Version 3. RFC 1939, May 1996.

- [14] Mark Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501, March 2003.
- [15] Microsoft Corporation. Server Message Block (SMB) Protocol Versions 2 and 3, July 2025.
- [16] Chris M. Lonnqvist and Tatu Ylonen. The Secure Shell (SSH) Protocol Architecture. RFC 4251, January 2006.
- [17] A. Banks E. Briggs K. Borgendale and R. Gupta. MQTT version 5.0, March 2019.
- [18] Thomas Haynes and David Noveck. Network File System (NFS) Version 4 Protocol. RFC 7530, March 2015.
- [19] EventHelix. Network file system protocol (nfs protocol sequence diagram), May 2011.
- [20] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.
- [21] Richard Barnes, Martin Thomson, Alfredo Pironti, and Adam Langley. Deprecating Secure Sockets Layer Version 3.0. RFC 7568, June 2015.
- [22] David W Chadwick and Andrew Basden. Evaluating trust in a public key certification authority. *Computers & security*, 20(7):592–611, 2001.
- [23] Paul Ford-Hutchinson. Securing FTP with TLS. RFC 4217, October 2005.
- [24] Paul E. Hoffman. SMTP Service Extension for Secure SMTP over Transport Layer Security. RFC 3207, February 2002.
- [25] Eric Rescorla. HTTP Over TLS. RFC 2818, May 2000.
- [26] Chris Newman. Using TLS with IMAP, POP3 and ACAP. RFC 2595, June 1999.

Appendice A

Codice

All'interno della seguente appendice sono riportate parti di codice. Anche in questo caso alcune funzioni non verranno riportate per intero perché troppo complesse o per via di ripetizioni (per esempio verrà mostrata solo una funzione per il riconoscimento dei protocolli dato che le altre sono molto simili tra loro). Il codice completo è visionabile sulla mia repository personale al link <https://github.com/Veronica-Falgiani/Network-Protocol-Verification-Agent>

A.1 main.py

```
1 #!/bin/python3
2
3 # Imports
4 import os
5 import sys
6 from utils.parser import args_parse, ip_parse, port_parse
7 from utils.terminal_colors import print_ok, print_fail, print_cmd
8 from agent.host_scan import host_scan
9 from agent.port_scan import PortScan
10 from agent.service_scan import ServiceScan
11 from agent.execute_tests import ExecuteTests
12 from utils.write_result import write_result
13
14 if __name__ == "__main__":
15     if "SUDO_UID" not in os.environ:
16         print_fail("This program requires sudo privileges")
17         sys.exit()
18
19     # Get arguments from cmd line
20     args = args_parse()
21
22     host_arg = args.host_scan
23     port_arg = args.port_scan
24     ip = args.host
25     ports_str = args.ports
26     verbose = args.verbose
27     no_tests = args.no_tests
```

```

28     # Verify that user input is correct
29     ip_parse(ip)
30     ports_list = port_parse(ports_str)
31
32     # Host scan
33     print("--- Checking host ---")
34     if host_scan(host_arg, ip, verbose):
35         if verbose:
36             print_ok("Host is up\n")
37         else:
38             if verbose:
39                 print_fail("Host is down\n")
40             sys.exit()
41
42     # Port scan
43     print("--- Checking ports ---")
44     port_scan = PortScan(ip)
45     port_scan.port_scan(port_arg, ports_list, verbose)
46     port_scan.get_open_ports()
47     print_cmd(port_scan, verbose)
48
49     # Protocol - Service scan
50     print("--- Checking protocols and services ---")
51     service_scan = ServiceScan(ip)
52     if port_scan.type == "TCP":
53         service_scan.tcp_scan(port_scan.open_ports, verbose)
54     else:
55         service_scan.udp_scan(port_scan.open_ports, verbose)
56     if not no_tests:
57         print_cmd(service_scan, verbose)
58
59     # If the user has requested a simple scan, prints the results and interrupts the
60     # execution
61     if no_tests:
62         print(service_scan)
63         sys.exit()
64
65     # Testing all protocols
66     print("--- Testing protocols and services ---")
67     report = ExecuteTests(ip)
68     report.execute_tests(service_scan.services, verbose)
69     print("\n----- RESULTS -----")
70     print(report)
71
72     # Write to file results
73     write_result(report)

```

A.2 agent/

host_scan.py

Viene riportata solamente la funzione TCP_SYN scan dato che le altre sono molto simili nei contenuti.

```
1 import sys
```

```
2 from utils.terminal_colors import print_fail, verbose_print
3 from scapy.all import *
4
5 # Ports used for syn and ack scan
6 SCAN_PORTS = [21, 22, 80, 443]
7
8
9 # Selecting the right scan based on the user input
10 def host_scan(host_arg: str, ip: str, verbose: bool):
11     if verbose:
12         verbose_print(f"Verifying {ip}")
13
14     match host_arg:
15         case "p":
16             res_status = ping_scan(ip)
17         case "s":
18             res_status = tcp_syn_scan(ip)
19         case "a":
20             res_status = tcp_ack_scan(ip)
21         case "u":
22             res_status = udp_scan(ip)
23         case None:
24             res_status = ping_scan(ip)
25         case _:
26             print_fail("Cannot find host scan type")
27             sys.exit()
28
29     # Clean line
30     print("\u001b[K", end="\r")
31
32     return res_status
33
34
35 ... <codice omesso> ...
36
37
38 # -----
39 # TCP SYN
40 # -----
41 def tcp_syn_scan(ip: str) -> bool:
42     res_status = False
43
44     for port in SCAN_PORTS:
45         packet = IP(dst=ip) / TCP(dport=port, flags="S")
46         res = sr1(packet, timeout=2, verbose=0)
47
48         if res is not None:
49             flag_res = res.sprintf("%TCP.flags%")
50
51             if flag_res == "SA":
52                 res_status = True
53
54     return res_status
55
56 ... <codice omesso> ...
```

port_scan.py

Di seguito viene mostrato il codice in forma ridotta. Vengono riportate a titolo d'esempio le funzioni di scan Connect e TCP_SYN

```

1 import socket
2 import sys
3 from utils.terminal_colors import print_fail, print_warning, verbose_print
4 from scapy.all import *
5
6
7 class PortScan:
8     def __init__(self, ip: str):
9         self.ip = ip
10        self.ports = {}
11        self.type = ""
12        self.open_ports = []
13
14    def __str__(self):
15        string = f"{'PORT':<10s} {'STATUS':<15s}\n"
16
17        for key, value in self.ports.items():
18            string += f"{str(key):<10s} {value:<15s}\n"
19
20    return string
21
22    def get_open_ports(self):
23        for key, value in self.ports.items():
24            if value != "closed" or value != "filtered":
25                self.open_ports.append(key)
26
27 # Selecting the right scan based on the user input
28 def port_scan(self, port_arg: str, ports_list: list, verbose: bool):
29     match port_arg:
30         case "c":
31             self.tcp_connect_scan(ports_list, verbose)
32         case "s":
33             self.tcp_syn_scan(ports_list, verbose)
34         case "f":
35             self.tcp_fin_scan(ports_list, verbose)
36         case "n":
37             self.tcp_null_scan(ports_list, verbose)
38         case "x":
39             self.tcp_xmas_scan(ports_list, verbose)
40         case "u":
41             self.udp_scan(ports_list, verbose)
42         case None:
43             self.tcp_connect_scan(ports_list, verbose)
44         case _:
45             print_fail("Cannot find scan type")
46             sys.exit()
47
48     # Clean line
49     print("\033[K", end="\r")
50
51     if len(self.ports) == 0:
52         print_fail("No open ports found!")
53         sys.exit()
54
55     # Send: connect() (TCP with SYN)
```

```

56     # Rec:  TCP with SYN/ACK -> open
57     #           no response -> closed/filtered
58     def tcp_connect_scan(self, ports_list: list, verbose: bool):
59         self.type = "TCP"
60
61         for port in ports_list:
62             if verbose:
63                 print("\u001b[K", end="\r")
64                 verbose_print(f"Testing {port}")
65
66             s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
67             s.settimeout(3)
68             res = s.connect_ex((self.ip, port))
69
70             # Port open
71             if res == 0:
72                 # print(f"{port} \t open")
73                 self.ports[port] = "open"
74
75             # Port closed/filtered
76             else:
77                 pass
78                 # print(f"{port} \t closed/filtered")
79                 # found_ports[port] = "closed/filtered"
80
81             s.close()
82
83     # Send: SYN
84     # Rec:  SYN/ACK -> RST -> open
85     #           RST -> closed
86     #           no response/ICMP unreachable -> filtered
87     def tcp_syn_scan(self, ports_list: list, verbose: bool):
88         self.type = "TCP"
89
90         for port in ports_list:
91             if verbose:
92                 print("\u001b[K", end="\r")
93                 verbose_print(f"Testing {port}")
94
95             packet = IP(dst=self.ip) / TCP(dport=port, flags="S")
96             res = sr1(packet, timeout=3, verbose=0)
97
98             if res is None or (
99                 res.sprintf("%ICMP.type%") == 3
100                and res.sprintf("%ICMP.code%") in [1, 2, 3, 9, 10, 13]
101            ):
102                 # print(f"{port} \t filtered")
103                 self.ports[port] = "filtered"
104
105             else:
106                 flag_res = res.sprintf("%TCP.flags%")
107
108                 if flag_res == "RA":
109                     pass
110                     # print(f"{port} \t closed")
111                     # found_ports[port] = "closed"
112                 elif flag_res == "SA":
113                     # print(f"{port} \t open")
114                     self.ports[port] = "open"
115
116     ... <codice omesso> ...

```

service_scan.py

Nel seguente codice sono riportate solo le funzioni per riconoscere FTP e FTPS. Il contenuto è stato ridotto per via della lunghezza eccessiva del file.

```

1 import socket
2 from time import sleep
3 from utils.terminal_colors import verbose_print
4 from scapy.all import *
5 from urllib.parse import urlparse
6 from http.client import HTTPConnection, HTTPSConnection
7 from ftplib import FTP, FTP_TLS
8 from smtplib import SMTP, SMTP_SSL
9 from telnetlib import Telnet
10 import ssl
11 import certifi
12 import dns.message, dns.query
13 from poplib import POP3, POP3_SSL
14 from imaplib import IMAP4, IMAP4_SSL
15 from impacket.smbconnection import SMBConnection
16 import paho.mqtt.client as mqtt_client
17 from operator import itemgetter
18 from enum import *
19 import subprocess
20
21
22 class ServiceScan:
23     # Defining self signed certificate for tls/ssl
24     context = ssl._create_unverified_context(ssl.PROTOCOL_TLS_CLIENT)
25     context.options &= ~ssl.OP_NO_SSLv3
26     context.minimum_version = 768
27     context.load_verify_locations(certifi.where())
28
29     def __init__(self, ip: str):
30         self.ip = ip
31         self.services = []
32
33     def __str__(self):
34         string = f"{'PORT':<10s} {'PROTOCOL':<15s} {'SERVICE':<100s}\n"
35
36         for service in self.services:
37             string += f"[{str(service['port'])}]:<10s] [{service['protocol']}:<15s] [{service['service']}:<100s]\n"
38
39         return string
40
41     def test_scan(self, open_ports: list, verbose: bool):
42         # Write protocols to test here
43         for self.check in self.tcp_check:
44             self.check(self, open_ports, verbose)
45
46         # Clean line for verbose print
47         print("\033[K", end="\r")
48
49         # Sorts list by port
50         self.services = sorted(self.services, key=itemgetter("port"))
51
52     def tcp_scan(self, open_ports: list, verbose: bool):
53         for self.check in self.tcp_check:
54             self.check(self, open_ports, verbose)

```

```
55     # Out of for because I needed third argument
56     self.nfs_check(open_ports, verbose, "T")
57
58     # Clean line for verbose print
59     print("\033[K", end="\r")
60
61     # Sorts list by port
62     self.services = sorted(self.services, key=itemgetter("port"))
63
64 def udp_scan(self, open_ports: list, verbose: bool):
65     for self.check in self.udp_check:
66         self.check(self, open_ports, verbose)
67
68     # Out of for because I needed third argument
69     self.nfs_check(open_ports, verbose, "U")
70
71     # Clean line for verbose print
72     print("\033[K", end="\r")
73
74     # Sorts list by port
75     self.services = sorted(self.services, key=itemgetter("port"))
76
77 # -----
78 # FTP
79 # -----
80 def ftp_check(self, open_ports: list, verbose: bool):
81     rem_ports = []
82     ip = self.ip
83
84     for port in open_ports:
85         service = {}
86
87         if verbose:
88             print("\033[K", end="\r")
89             verbose_print(f"Scanning {port} for FTP")
90
91         try:
92             ftp = FTP()
93             ftp.connect(host=ip, port=port, timeout=3)
94             ftp.quit()
95
96             # smtp also responds to this, so we need to verify the banner
97             s = socket.socket()
98             s.connect((ip, port))
99             banner = s.recv(1024)
100            banner = banner.decode("utf-8", errors="ignore")
101
102            if "FTP" in banner:
103                rem_ports.append(port)
104
105                service["port"] = port
106                service["protocol"] = "FTP"
107                service["service"] = str(banner).strip()[4:]
108
109                self.services.append(service)
110
111            s.close()
112
113        except Exception as e:
114            pass
```

```
116     for port in rem_ports:
117         open_ports.remove(port)
118
119
120     # -----
121     # FTP/SSL
122     # -----
123     def ftps_check(self, open_ports: list, verbose: bool):
124         rem_ports = []
125         ip = self.ip
126
127         for port in open_ports:
128             service = {}
129
130             if verbose:
131                 print("\033[K", end="\r")
132                 verbose_print(f"Scanning {port} for FTP-SSL")
133
134             try:
135                 # FTP_SSL not properly working, need to find out why
136                 # ftps = FTP_TLS()
137                 # ftps.connect(ip, port, timeout=3)
138                 # banner = ftps.getwelcome()
139                 # ftps.quit()
140
141                 # smtp also responds to this, so we need to verify the banner ?
142                 sock = socket.create_connection((ip, port), timeout=3)
143                 ssock = ServiceScan.context.wrap_socket(sock, server_hostname=ip)
144                 ssl_version = ssock.version()
145
146                 sleep(1) # Banner was cut in half so we need ot wait
147                 banner = ssock.recv(2048)
148                 banner = banner.decode("utf-8", errors="ignore")
149
150                 if "FTP" in banner:
151                     rem_ports.append(port)
152
153                     service["port"] = port
154                     service["protocol"] = "FTP-SSL"
155                     service["service"] = str(banner).strip()[4:] + " - " + ssl_version
156
157                     self.services.append(service)
158
159             ssock.close()
160
161         except ssl.SSLCertVerificationError as e:
162             print(port, e)
163
164         except Exception as e:
165             pass
166
167         for port in rem_ports:
168             open_ports.remove(port)
169
170     ... <codice omesso> ...
171
172     # -----
173     # UNDEFINED
174     # -----
175     def undefined(self, open_ports: list, verbose: bool):
176         rem_ports = []
```

```
177     ip = self.ip
178
179     for port in open_ports:
180         service = {}
181
182         if verbose:
183             print("\033[K", end="\r")
184             verbose_print(f"Scanning {port} for SSL-TLS")
185
186         try:
187             pass # TODO
188         except Exception as e:
189             pass
190
191         for port in rem_ports:
192             open_ports.remove(port)
193
194     # -----
195     # TEMPLATE
196     # -----
197     def check(self, open_ports: list, verbose: bool):
198         rem_ports = []
199         ip = self.ip
200
201         for port in open_ports:
202             service = {}
203             if verbose:
204                 print("\033[K", end="\r")
205                 verbose_print(f"Scanning {port} for PROTOCOL")
206
207             try:
208                 # Insert code here
209
210                 rem_ports.append(port)
211                 service["port"] = port
212                 service["protocol"] = "MQTT"
213                 service["service"] = "undefined"
214
215                 self.services.append(service)
216
217             except Exception:
218                 pass
219
220         for port in rem_ports:
221             open_ports.remove(port)
222
223     # List of protocol scans to use
224     tcp_check = [
225         ftp_check,
226         ssh_check,
227         telnet_check,
228         smtp_check,
229         dns_check,
230         http_check,
231         pop_check,
232         imap_check,
233         smb_check,
234         mqtt_check,
235         # SSL protocols
236         https_check,
237         https_check,
```

```

238     smtps_check ,
239     pops_check ,
240     imaps_check ,
241     mqtts_check ,
242     ssltls_check ,
243     # Undefined
244     undefined ,
245 ]
246
247 udp_check = [
248     http_check ,
249     https_check ,
250     dns_check ,
251     # Undefined
252     undefined ,
253 ]

```

results.py

```

1 class Results:
2     def __init__(self, port, prot, service):
3         self.port = port
4         self.prot = prot
5         self.service = service
6         self.prot_max_vulns = 0
7         self.prot_max_auth_vulns = 0
8         self.serv_max_vulns = 0
9         self.serv_max_auth_vulns = 0
10        self.found_vulns = []
11        self.found_auth_vulns = []
12        self.unsafe_ver = False
13        self.unsafe_ver_cve = []
14        self.unsafe_tls = False
15        self.prot_auth = False
16        self.serv_auth = False
17
18    def __str__(self):
19        string = f"{str(self.port):<10s} {self.prot:<15s} {self.service:<100s}\n|\n"
20
21        # Print if the service version is vulnerable
22        string += "| ----- VERSION CHECK -----|\n"
23        if self.unsafe_ver:
24            string += (
25                "|\\__ THIS SERVICE VERSION IS VULNERABLE AND NEEDS TO BE UPDATED!\n"
26            )
27            string += "|      reference:\n"
28            for cve in self.unsafe_ver_cve:
29                cve_number = cve.split("?id=")[1]
30                string += f"|      - {cve_number}: {cve}\n"
31
32            string += "|\n"
33        else:
34            string += "|\\__ The service version is not vulnerable.\n|\n"
35
36        # Print if the ssl/tls version is outdated
37        if "TLS" in self.service or "SSL" in self.service:
38            string += "| ----- SSL/TLS CHECK -----|\n"
39            if self.unsafe_tls:
40                string += "|\\__ THE SERVICE USES A DEPRECATED SSL/TLS PROTOCOL! \n|\n"
41            else:

```

```
        string += "|\\_ The service uses the currently supported SSL/TLS\n"
protocol \n|\n"

43
44     # Print all the information about the tests
45     string += "| ----- VULNERABILITIES -----|\n"
46     if (self.prot_max_vulns + self.serv_max_vulns) == 0:
47         string += "|\\_ No tests found for this protocol\n"
48     elif len(self.found_vulns) == 0:
49         string += "|\\_ The protocol has been tested and no vulnerabilities have
been found\n"
50     else:
51         for vuln in self.found_vulns:
52             string += f"|\\_ {vuln['name']}|\n"
53             string += f"|     description: {vuln['description']}|\n"
54             string += f"|     severity: {vuln['severity']}|\n"
55
56     string += (
57         "|n| ----- AUTHENTICATED VULNERABILITIES -----|\n"
58    )
59     if (self.prot_max_auth_vulns + self.serv_max_auth_vulns) == 0:
60         string += "|\\_ No tests found for this protocol\n"
61     elif not self.prot_auth and not self.serv_auth:
62         string += "|\\_ No credentials were given\n"
63     elif len(self.found_auth_vulns) == 0:
64         string += "|\\_ The protocol has been tested and no vulnerabilities have
been found\n"
65     else:
66         for vuln in self.found_auth_vulns:
67             string += f"|\\_ {vuln['name']}|\n"
68             string += f"|     description: {vuln['description']}|\n"
69             string += f"|     severity: {vuln['severity']}|\n"
70
71     string += "\n\n"
72
73     return string
74
75 def __json__(self):
76     repr = {
77         "port": self.port,
78         "protocol": self.prot,
79         "service": self.service,
80         "unsafe_version": self.unsafe_ver,
81         "unsafe_version_cve": self.unsafe_ver_cve,
82         "vulnerabilities": self.found_vulns,
83         "auth_vulnerabilities": self.found_auth_vulns,
84     }
85
86     if "SSL" in self.service or "TLS" in self.service:
87         position = list(repr.keys()).index('misconfigurations')
88         items = list(repr.items())
89         items.insert(position, ("unsafe_tls", self.unsafe_tls))
90         repr = dict(items)
91
92     return repr
93
94 def add_prot_max(self, prot_max_vulns: int, prot_max_auth_vulns: int):
95     self.prot_max_vulns = prot_max_vulns
96     self.prot_max_auth_vulns = prot_max_auth_vulns
97
98 def add_serv_max(self, serv_max_vulns: int, serv_max_auth_vulns: int):
99     self.serv_max_vulns = serv_max_vulns
```

```

100     self.serv_max_auth_vulns = serv_max_auth_vulns
101
102     def set_vulns(self, vulns: dict):
103         self.found_vulns.append(vulns)
104
105     def set_auth_vulns(self, vulns: dict):
106         self.found_auth_vulns.append(vulns)

```

execute_tests.py

Viene riportato solo il codice che svolge i test sui protocolli, dato che le funzioni utilizzate per il test dei servizi sono identiche. Anche `test_ssl()` è stato omesso per via della somiglianza con `test()`

```

1 import json
2 import os
3 import socket
4 import ssl
5 import certifi
6 import getpass
7 import re
8 from utils.terminal_colors import verbose_print
9 from agent.results import Results
10
11
12 class ExecuteTests:
13     # Defining self.signed_certificate for tls/ssl
14     context = ssl._create_unverified_context(ssl.PROTOCOL_TLS_CLIENT)
15     context.options &= ~ssl.OP_NO_SSLv3
16     context.minimum_version = 768
17     context.load_verify_locations(certifi.where())
18
19     def __init__(self, ip):
20         self.ip = ip
21         self.report = []
22
23     def __str__(self):
24         string = f"{'PORT':<10s} {'PROTOCOL':<15s} {'SERVICE':<100s}\n"
25
26         for result in self.report:
27             string += str(result) + "\n"
28
29         return string
30
31     def execute_tests(self, services: list, verbose: bool):
32         # Protocol test
33         for service in services:
34             port = service["port"]
35             prot = service["protocol"]
36             service = service["service"]
37
38             # Tests the generic protocol
39             base_dir = os.path.dirname(__file__)
40             rel_path_prot = "../tests/prot/" + prot.lower() + "_test.json"
41             path_prot = os.path.join(base_dir, rel_path_prot)
42
43             try:
44                 with open(path_prot) as file:
45                     test_file = json.load(file)

```

```

46     vulns = test_file["vulns"]
47     login = test_file["login"]
48     auth_vulns = test_file["auth_vulns"]
49     serv_names = test_file["serv_names"]
50
51     # Create class
52     results = Results(port, prot, service)
53
54     prot_max_vulns = len(vulns)
55     i_mis = 1
56
57     prot_max_auth_vulns = len(auth_vulns)
58     i_auth = 1
59
60     results.add_prot_max(prot_max_vulns, prot_max_auth_vulns)
61
62     # Start testing for misconfigurations
63     auth = False
64     self.check_vulns(
65         vulns,
66         verbose,
67         i_mis,
68         prot_max_vulns,
69         port,
70         prot,
71         service,
72         results,
73         auth,
74     )
75
76     # If auth_vulns has tests, asks the user for login info and inserts
77     # the correct login messages in a list
78     if auth_vulns:
79         print("\n--- Asking for protocols credentials ---")
80
81         login_list = self.try_login(prot, port, service, login)
82
83         # Start testing for misconfigurations
84         if login_list:
85             results.prot_auth = True
86             auth = True
87             self.check_vulns(
88                 auth_vulns,
89                 verbose,
90                 i_auth,
91                 prot_max_auth_vulns,
92                 port,
93                 prot,
94                 service,
95                 results,
96                 auth,
97                 login_list,
98             )
99
100        # Services test
101        for name in serv_names:
102            ... <codice omesso> ...
103
104    except FileNotFoundError:
105        results = Results(port, prot, service)

```

```

106         self.report.append(results)
107
108     def check_vulns(
109         self,
110         vulns,
111         verbose,
112         i_mis,
113         max_vulns,
114         port,
115         prot,
116         service,
117         results,
118         auth,
119         login_list=[],
120     ):
121         for name, info in vulns.items():
122             vuln = {}
123
124             if verbose:
125                 print("\033[K", end="\r")
126                 verbose_print(
127                     f"Scanning {port} with {prot} - {service} using {name} [{i_mis}/{max_vulns}]"
128                 )
129             i_mis += 1
130
131             # Complex ssl/tls test: establishes a connection and then sends a message and
132             # compares results
133             if "SSL" in prot:
134                 vuln = self.test_ssl(name, info, self.ip, port, service, login_list)
135                 self.check_tls(service, results)
136
137             # Complex test: sends a message and compares the results
138             elif "recv" in info or "not_recv" in info:
139                 vuln = self.test(name, info, self.ip, port, service, login_list)
140
141             # Simple test: checks if the port is open
142             else:
143                 vuln["name"] = name
144                 vuln["service"] = service
145                 vuln["description"] = info["description"]
146                 vuln["severity"] = info["severity"]
147
148             if vuln and auth:
149                 results.set_auth_vulns(vuln)
150             elif vuln and not auth:
151                 results.set_vulns(vuln)
152
153             # Clean line
154             print("\033[K", end="\r")
155
156     def test(self, name: str, info: dict, ip: str, port: int, service: str, login_list):
157         recv = None
158         not_recv = None
159
160         send_str = info["send"]
161         send_list = send_str.split("~~")
162
163         if "recv" in info:
164             recv = info["recv"]

```

```

165     elif "not_recv" in info:
166         not_recv = info["not_recv"]
167
168     try:
169         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
170         sock.settimeout(5)
171         sock.connect((ip, port))
172
173         for message in login_list:
174             sock.send(message.encode())
175
176         # Sends all the commands to the server
177         for send in send_list:
178             # print(send)
179             sock.send(send.encode())
180             res = sock.recv(1024)
181             # print(res.decode())
182
183         # Compares the received message to the one in the json
184         if (
185             recv is not None
186             and re.search(recv, res.decode())
187             or not_recv is not None
188             and not re.search(not_recv, res.decode())
189         ):
190             vuln = {}
191             vuln["name"] = name
192             vuln["service"] = service
193             vuln["description"] = info["description"]
194             vuln["severity"] = info["severity"]
195             return vuln
196
197         sock.close()
198
199     except TimeoutError:
200         pass
201
202     def test_ssl(
203         self, name: str, info: dict, ip: str, port: int, service: str, login_list
204     ):
205         ... <codice omesso> ...
206
207     def check_banner(self, service: str, vuln_serv_version: dict, results: Results):
208         for version, cve in vuln_serv_version.items():
209             if version in service:
210                 results.unsafe_ver = True
211                 results.unsafe_ver_cve = cve
212
213     def check_tls(self, service: str, results: Results):
214         if not ("TLSv1.3" in service or "TLSv1.2" in service):
215             results.unsafe_tls = True
216
217     def try_login(self, prot, port, service, login) -> list:
218         # Asks the user max 3 times for the password
219         for i in range(3):
220             # Opens SSL socket
221             if "SSL" in prot:
222                 sock = socket.create_connection((self.ip, port), timeout=3)
223                 sock = ExecuteTests.context.wrap_socket(sock, server_hostname=self.ip)
224
225             # Opens simple socket

```

```

226     else:
227         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
228         sock.settimeout(5)
229         sock.connect((self.ip, port))
230
231 # Asks the user for login info
232 print(f"{prot} - {service} username: ", end="")
233 username = input()
234 password = getpass.getpass(f"{prot} - {service} password: ")
235 if username == "" and password == "":
236     login_list = []
237     return login_list
238 else:
239     login_str = login["send_str"].replace("_username_", username)
240     login_str = login_str.replace("_password_", password)
241
242 # Sends the login strings to the server
243 login_list = login_str.split("~~")
244 for message in login_list:
245     sock.send(message.encode())
246     res = sock.recv(1024)
247
248 # Checks the response of the server
249 if re.search(login["recv_str"], res.decode()):
250     sock.close()
251     return login_list
252 else:
253     sock.close()
254     print(f"Failed login {i + 1}/3")
255
256     sock.close()
257     login_list = []
258     print("Max login failed")
259 return login_list

```

A.3 utils/

parser.py

```

1 import argparse
2 import sys
3 import socket
4 import re
5 from utils.terminal_colors import print_fail, print_warning
6
7 # python3 main.py -hs p -ps s 100:200 192.168.0.1
8
9
10 def args_parse():
11     parser = argparse.ArgumentParser(
12         prog="main.py",
13         description="Agent for Advanced Network Protocol Verification. This program needs
14             sudo privileges to run.",
15         add_help=False,
16         formatter_class=argparse.RawTextHelpFormatter,
17     )
18
19     parser.add_argument(

```

```

19     "--h", "--help", action="help", help="Show this help message and exit"
20 )
21 parser.add_argument(
22     "-v", "--verbose", action="store_true", help="Increase output verbosity"
23 )
24 parser.add_argument(
25     "--nt",
26     "--no_tests",
27     action="store_true",
28     help="Scans the target for services but doesn't execute a vulnerability scan",
29 )
30 parser.add_argument(
31     "-hs",
32     "--host_scan",
33     help="Host scan to execute: [p]ing, [s]yn, [a]ck, [u]dp (ping scan will be used by
34     default)",
35 )
36 parser.add_argument(
37     "-ps",
38     "--port_scan",
39     help="Port scan to execute: [c]onnect, [s]yn, [f]in, [n]ull, [x]mas, [u]dp (connect
40     scan will be used by default)",
41 )
42 parser.add_argument(
43     "ports",
44     help="Single port [x], multiple ports [x,y,z], port range [x:y] to scan or all
45     ports [all]",
46 )
47 parser.add_argument("host", help="Host to scan using ipv4 address")
48
49 args = parser.parse_args()
50
51 # Parses ip from user input
52 def ip_parse(ip: str):
53     regex = "^(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\\.(?
54     {3})(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])$"
55
56     if ip == "localhost":
57         ip = "127.0.0.1"
58     if re.search(regex, ip):
59         return
60     else:
61         print_fail("IP not valid!")
62         sys.exit()
63
64 # Parsing ports we need to scan from user input
65 def port_parse(port_str: str) -> list:
66     ports = []
67
68     if port_str == "all":
69         ports = list(range(0, 65536))
70
71     # Contiguous port list
72     elif ":" in port_str:
73         p_range = port_str.split(":")
74
75         if (

```

```

76     p_range[0].isnumeric()
77     and p_range[1].isnumeric()
78     and int(p_range[0]) <= int(p_range[1])
79     and 0 <= int(p_range[0]) <= 65535
80     and 0 <= int(p_range[1]) <= 65535
81 ):
82     for i in range(int(p_range[0]), int(p_range[1]) + 1):
83         ports.append(i)
84     else:
85         print_fail("Ports are not valid!")
86         sys.exit()
87
88 # Random port list
89 elif "," in port_str:
90     p_list = port_str.split(",")
91     for item in p_list:
92         if item.isnumeric() and int(item) >= 0 and int(item) <= 65535:
93             ports.append(int(item))
94         else:
95             print_warning(f"port {item} not valid! Skipping it")
96
97 if len(ports) == 0:
98     print_fail("Ports are not valid!")
99     sys.exit()
100
101 ports.sort()
102
103 # Single port
104 elif port_str.isnumeric() and int(port_str) >= 0 and int(port_str) <= 65535:
105     ports.append(int(port_str))
106
107 # Generic error
108 else:
109     print_fail("Ports are not valid!")
110     sys.exit()
111
112 return ports

```

terminal_colors.py

```

1 class bcolors:
2     HEADER = "\033[95m"
3     OKBLUE = "\033[94m"
4     OKCYAN = "\033[96m"
5     OKGREEN = "\033[92m"
6     WARNING = "\033[93m"
7     FAIL = "\033[91m"
8     ENDC = "\033[0m"
9     BOLD = "\033[1m"
10    UNDERLINE = "\033[4m"
11
12
13 # Green print
14 def print_ok(string: str):
15     print(bcolors.OKGREEN + string + bcolors.ENDC)
16
17
18 # Yellow print
19 def print_warning(string: str):
20     print(bcolors.WARNING + string + bcolors.ENDC)
21

```

```

22 # Red print
23 def print_fail(string: str):
24     print(bcolors.FAIL + string + bcolors.ENDC)
25
26
27 # Cyan print
28 def verbose_print(string: str):
29     print(bcolors.OKCYAN + string + bcolors.ENDC, end="\r")
30
31
32 def print_cmd(string, verbose):
33     if verbose:
34         print(string)
35

```

write_results.py

La funzione utilizzata per creare il codice HTML è stata ridotta per via della lunghezza eccessiva e la semplicità del codice.

```

1 import os
2 from datetime import datetime
3 import matplotlib.pyplot as plt
4 import json
5 from jinja2 import Environment, FileSystemLoader
6 from agent.results import Results
7
8
9 TIME = "Result_" + datetime.today().strftime("%Y-%m-%d_%H:%M:%S")
10 RES_DIR = "res/" + TIME + "/"
11
12
13 def write_result(report: Results):
14     # Creates directories for result files (if they don't exist)
15     os.makedirs(os.path.dirname("res/"), exist_ok=True)
16     os.makedirs(os.path.dirname(RES_DIR), exist_ok=True)
17     os.makedirs(os.path.dirname(f"{RES_DIR}img/"), exist_ok=True)
18     os.chmod("res/", 0o777)
19     os.chmod(f"{RES_DIR}", 0o777)
20     os.chmod(f"{RES_DIR}img/", 0o777)
21
22     log_result(report)
23     json_result(report)
24     html_result(report)
25
26     print(f"Results can be found in: {RES_DIR}")
27
28
29 # Writes a human readable log
30 def log_result(report):
31     file_log = RES_DIR + f"{report.ip}_results.log"
32
33     with open(file_log, "w") as res_file:
34         res_file.write(f"##### RESULTS FOR {report.ip} #####\n\n")
35         res_file.write("PORT \t PROTOCOL \t SERVICE\n")
36         res_file.write("-----\n\n\n")
37
38         for result in report.report:
39             res_file.write(str(result))
40

```

```
41 # Creates a json file with all the results
42 def json_result(report):
43     file_json = RES_DIR + f"{report.ip}_results.json"
44
45     with open(file_json, "w") as res_file:
46         res_dict = {
47             "ip": report.ip,
48             "timestamp": datetime.today().strftime("%Y-%m-%d_%H:%M:%S"),
49             "services": [],
50         }
51
52         for result in report.report:
53             res_dict["services"].append(result.__json__())
54
55     json.dump(res_dict, res_file, indent=4)
56
57
58
59 # Creates a html page with results and graphs
60 def html_result(report: Results):
61     # Creates a directory (if it doesn't exist) and a result file
62     file_html = RES_DIR + f"{report.ip}_results.html"
63
64     html_pills = ""
65     html_title = ""
66     html_protocols = ""
67
68     for result in report.report:
69         html_version = ""
70         html_tls = ""
71         html_vulns = ""
72         html_auth_vulns = ""
73
74         # Protocol name and service version
75         html_title = f"""
76             <div id={result.prot}-{result.port} class="tab-pane fade">
77                 <h3><b>Port {result.port} - {result.prot} - {result.service}</b></h3>
78             """
79
80         ...
81
82         # Checks if there are vulns to print
83         if (result.serv_max_vulns + result.prot_max_vulns) == 0:
84             html_vulns += """
85                 <div class='my-3'>
86                     <hr>
87                     <h4 style="text-align:center"> VULNERABILITIES </h4>
88                     <p class="my-3">No tests found for the protocol</p>
89                 </div>
90             """
91
92         # Checks if the protocol has vulns or not
93         elif len(result.found_vulns) != 0:
94             severity_html = {
95                 "high": 0,
96                 "high_results": "",
97                 "medium": 0,
98                 "medium_results": "",
99                 "low": 0,
100                "low_results": "",
101                "ok": 0,
```

```

102     }
103
104     html_vulns += f"""
105         <div class='my-3'>
106             <hr>
107             <h4 style="text-align:center"> VULNERABILITIES </h4>
108             
109             <ul>
110             """
111
112     for vuln in result.found_vulns:
113         match vuln["severity"]:
114             case "high":
115                 severity_html["high"] += 1
116                 severity_html["high_results"] += f"""
117                     <li class='my-3'><b> {vuln["name"]} </b><br> description: {vuln["description"]}</li>
118                     """
119
120             case "medium":
121                 severity_html["medium"] += 1
122                 severity_html["medium_results"] += f"""
123                     <li class='my-3'><b> {vuln["name"]} </b><br> description: {vuln["description"]}</li>
124                     """
125
126             case "low":
127                 severity_html["low"] += 1
128                 severity_html["low_results"] += f"""
129                     <li class='my-3'><b> {vuln["name"]} </b><br> description: {vuln["description"]}</li>
130                     """
131
132         html_vulns += (
133             f"<li style='color:#EC6B56'><h5><b> HIGH: {severity_html['high']}

```

```
156         </div>
157     """
158
159     ... <codice omesso> ...
160
161     html_protocols += (
162         html_title
163         + html_version
164         + html_tls
165         + html_vulns
166         + html_auth_vulns
167         + "</div>"
168     )
169
170 # Setup html template via jinja2 and write to file
171 env = Environment(loader=FileSystemLoader("utils"))
172 template = env.get_template("report_template.html")
173
174 html = template.render(
175     page_title_text=f"Result {TIME}",
176     title_text=f"Report for {report.ip}",
177     html_pills=html_pills,
178     html_protocols=html_protocols,
179 )
180
181 with open(file_html, "w") as res_file:
182     res_file.write(html)
183
184
185 # Creates a png image of a pie chart based on the results of a protocol
186 def draw_graph(severity_html: dict, result: Results, max_vulns: int, type: str):
187     labels = "high", "medium", "low", "ok"
188     colors = ["#EC6B56", "#FFC154", "#47B39C", "skyblue"]
189
190     severity_html["ok"] = (
191         max_vulns
192         - severity_html["high"]
193         - severity_html["medium"]
194         - severity_html["low"]
195     )
196
197     # Creating pie chart for html
198     sizes = [
199         severity_html["high"],
200         severity_html["medium"],
201         severity_html["low"],
202         severity_html["ok"],
203     ]
204
205     wedges, texts = plt.pie(
206         sizes,
207         labels=labels,
208         startangle=90,
209         colors=colors,
210         textprops={"color": "w", "size": "x-large"},
211     )
212
213     # Updates labels with counters and removes labels that correspond to 0 vulns
214     for label in texts:
215         label_txt = label.get_text()
216         label.set_text(f"{label_txt} - {severity_html[label_txt]}")
```

```
217     if label_txt != "ok" and severity_html[label_txt] == 0:
218         label.set_text("")
219     elif label_txt == "ok" and severity_html["ok"] == 0:
220         label.set_text("")
221
222     plt.savefig(
223         f"{RES_DIR}/img/{result.prot}-{result.port}{type}.png", transparent=True
224     )
225     plt.clf()
```

report_template.html

```
1 <html data-bs-theme="dark">
2   <head>
3     <title>{{page_title_text}}</title>
4     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.5/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-SgOJa3DmI69IUzQ2PVdRZhwQ+dy64/BUTbMJw1MZ8t5HZApcHrRKUc4W0kG879m7" crossorigin="anonymous">
5       <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.5/dist/js/bootstrap.bundle.min.js" integrity="sha384-k6d4wzSIapyDyv1kpU366/PK5hCdSbCRGRCMv+epl0QJWyd1fbcau90CUj5zNLiq" crossorigin="anonymous"></script>
6   </head>
7   <body>
8     <div class="container">
9       <div class="mx-auto m-5">
10         <h1 class="text-center">{{title_text}}</h1>
11       </div>
12       <hr>
13       <div class="d-flex align-items-start mt-5">
14         <div class="nav flex-column nav-pills me-3">
15           {{html_pills}}
16         </div>
17         <div class="container-sm tab-content ms-5 p-5 border border-2 rounded-4">
18           {{html_protocols}}
19         </div>
20       </div>
21     </div>
22   </body>
23 </html>
```

Appendice B

Esempio file di test

I file di test riportati riguardano il protocollo FTP e il servizio vsFTPD. I contenuti dei JSON sono stati redatti a titolo d'esempio. Per tanto hanno ancora molte informazioni che possono essere aggiunte per svolgere in modo ancora più approfondito i test.

Test per il protocollo FTP

Il seguente file presenta i test per le vulnerabilità più conosciute del protocollo FTP. Queste informazioni sono state recuperate tramite pagine di CVE e script già presenti per altri software.

```
1  {
2      "vulns": {
3          "ANONYMOUS LOGIN ENABLED": {
4              "description": "Anonymous login is enabled, everyone can access
5                  the service",
6                  "send": "\n~~USER anonymous\n~~PASS\n",
7                  "recv": "230",
8                  "severity": "high"
9          },
10         "CVE-2010-1938": {
11             "description": "Allows remote attackers to cause a denial of
12                 service (daemon crash) or possibly execute arbitrary code via a long
13                 username.",
14                 "send": "\n~~user AAAAAAAAAAAAAAAAAAAAAAAA\n",
15                 "not_recv": "331 Please specify the password",
16                 "severity": "high"
17         },
18         "login": {
19             "send_str": "\n~~USER _username_\n~~PASS _password_\n",
20             "recv_str": "230 Login successful."
21         }
22     }
23 }
```

```

20 "auth_vulns": {
21   "BOUNCE ATTACK": {
22     "description": "If not correctly configured the PORT command can
23       use the victim machine to request access to port indirectly. This
24       can be used to scan hosts ports discretely.",
25     "send": "PORT\n",
26     "not_recv": "500",
27     "severity": "medium"
28   },
29   "serv_names": [
30     "vsftpd"
31   ]
32 }

```

Test per il servizio vsFTPD

Il seguente file contiene un test di esempio per il servizio vsFTPD. Le informazioni per ricreare le vulnerabilità che colpiscono questo servizio sono molto complesse e difficili da riportare. Per questo anche altre piattaforme preferiscono individuare le vulnerabilità presenti dalla versione del servizio.

```

1 {
2   "vulns": {
3     "BACKDOOR COMMAND EXECUTION": {
4       "description": "Allows users to leverage a backdoor to make a
5         command execution.",
6       "send": "\n~~USER X:)\n~~PASS X\n~~id\n",
7       "not_recv": "530",
8       "severity": "high"
9     }
10   },
11   "login": {},
12   "auth_vulns": {},
13   "vuln_serv_version": {
14     "1.1.3": ["https://www.cve.org/CVERecord?id=CVE-2004-0042"],
15     "2.0.5": ["https://www.cve.org/CVERecord?id=CVE-2004-0042"],
16     "2.3.4": ["https://www.cve.org/CVERecord?id=CVE-2011-2523"],
17     "3.0.2": ["https://www.cve.org/CVERecord?id=CVE-2015-1419"]
18   }
19 }

```