Ziua 1) **5.mai.2021**

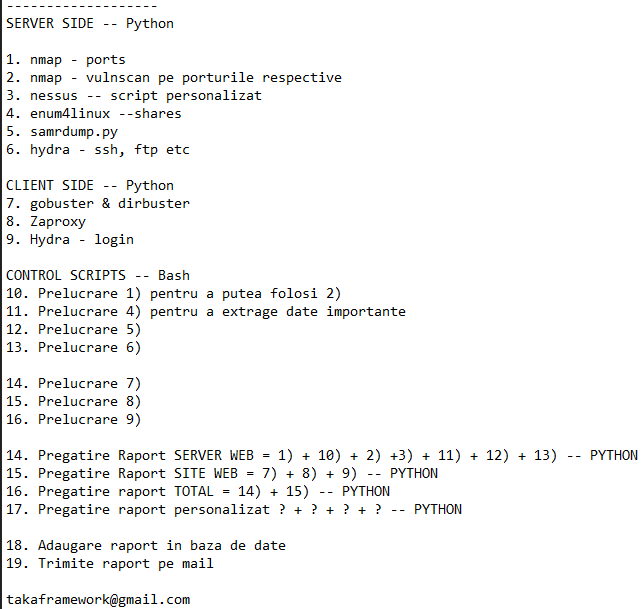
Selectarea programelor care urmeaza sa fie folosite : nmap, nessus, enum4linux, samrdump.py, hydra, gobuster, zaproxy. Pe baza acestora se vor crea scripturi ce le vor rula si prelucra datele de iesire.

Utilizatorul aplicatiei are 4 optiuni de scanare disponibile si anume : scanarea serverului, scanarea site-ului web, scanarea totala ( ce le include pe cele doua mentionare anterior ) si scanarea personalizata ( recomandata utilizatorilor avansati ) ce pune la dispozitie optiunile implicite de la scanarea serverului si a site-ului web ca optiuni separate ce pot fi selectate.

Dupa ce un script ruleaza un program, acesta va salva continutul datelor de iesire intr-un fisier de tipul text, csv, xml, json etc. Pentru a prelucra datele intr-un format mai usor de inteles pentru utilizator, dar si pentru a putea fi implementat in raportul final, acestea fiind de asemenea salvate intr-un alt fisier.

La final, dupa ce fiecare script isi termina executia iar datele sale de iesire sunt prelucrate, un alt script destinat combinarii fisierelor prelucrate va genera un raport final sub forma pdf,docx ce va fi salvat in baza de date a aplicatiei, ulterior fiind trimis pe mail utilizatorului folosind un script personalizat.

Schema scripturilor :



Ziua 2 ) 16**.mai.2021**

**nmap**

Scripturile pe care le folosesc trebuie sa poate executa comenzi de sistem intrucat trebuie sa foloseasca programele mentionate anterior. Aveam posibilitatea de a alege din cateva librarii in acest scop cum ar fi os.system si subprocess. Am optat pentru cea din urma pentru ca, imi ofera posibilitatea instiintarii in vederea terminarii comenzilor pe care le executa. Din nevoia de a prelucra datele de iesire ale programelor, trebuie ca, dupa terminarea lor sa se poata apela un alt script ce face acest lucru.

Pentru a face acest lucru, dupa ce execut o comanda folosesc returncode pentru a afla daca aceasta s-a executat intr-un mod corect ce poate facilitata prelucrarea outputului. In cazul in care returncode este 0 atunci pot trece mai departe si executa scriptul bash ce face prelucrarea fisierului iar, in caz contrar, arunca o exceptie. Intr-o situatie reala nu ar trebui sa apara erori, intrucat voi plasa filtre atat in frontend cat si in backend. Aceste erori ma ajuta in procesul de creare al scriptului.

Comenzile sunt formate pe baza unei combinatii cu inputul utilizatorului iar varianta recomandata initial in documentatie, unde subprocess.run primeste o lista de argumenti, fiecare reprezentand un singur cuvant din intreaga comanda, nu a mers. De aceea folosesc un alt argument, shell=True ceea ce face ca acea comanda sa fie tratata exact ca si cum ar fi fost executata intr-un terminal real.

Din punct de vedere al securitatii aceasta nu este cea mai buna optiune intrucat poate fi realizata o exploatare a binarelor din linux care ar putea duce la shell injection insa, am structurat comenzile astfel incat sa nu primeasca niciun argument extern care nu este sanitizat.

Dupa ce se termina prima scanare, datele de iesire sunt directionate intr-un fisier numit scanare, intr-un folder cu numele ip-ului/DNS care este scanat. Daca acest lucru s-a efectuat cu succes se apeleaza scriptul extern onlyports.sh care prelucreaza acel ouput pentru a primi doar numerele porturilor gasite. Pe baza acestora se va realiza o scanare completa, detaliata ce contine detalii despre serviciile care ruleaza pe acele porturi si, de asemenea, daca exista un CVE pentru acestea.

Ziua 3 ) 18**.mai.2021**

Dupa ce primu faza a scriptului este completa, primesc in variabila globala ports porturile asupra carora urmeaza sa se efectueze scanarea pentru vulnerabilitati.

Pentru ca acesta este un obiect de tipul Union Completed Process a trebuit de asemenea prelucrat pentru captarea datelor de iesire. Dupa prelucrare, acestea erau de tipul bytes si am folosit functia decode() pentru a transforma in text si functia split() pentru a nu primi linii noi intrucat poate arunca o eroare cand variabila aceasta este concatenata in comanda pentru scanare.

Aceasta este faza a doua, unde se verifica daca exista in vreo baza de date dedicata CVE-urilor pentru acestea. Dupa ce scanarea isi termina executia, output-ul este salvat intr-un fisier numit vuln\_scan. Acesta va fi prelucrat prin intermediul scriptului onlyvuln.sh in vederea adaptarii acestuia pentru raportul final ce urmeaza livrat utilizatorului.

De asemenea am creat cateva functii. Functia mkdir\_scans ce se asigura de creare unui director in folderul scans ce are numele tintei scanate.

O alta functie se asigura de exista directorului SCANS. In cazul in care acesta nu exista il creeaza. Pentru a verifica daca exista se testeaza cu comanda pentru listare a elementelor din interiorul acestuia ‘ls SCANS’. Apare o eroare, iar ca aceasta sa nu fie concatenata in output am redirectionat stream-ul pentru erori in DEVNULL ( 2>/dev/null ).

Ultima functie din acest script este cea pentru scanarea de vulnerabilitati cunoscute unde folosim aceeasi variabila globala pentru a micsora aria de scanare.

Ultimele doua scripturi arunca o exceptie in cazul in care apar erori in momentul executiei comenzilor respective.

Ultima functie este cea pentru verificarea input-ului introdus de utilizator. Deoarece folosesc optiunea shell=True pot aparea diversi vectori de exploatare al scriptului. De aceea datele de intrare ale utilizatorului sunt comparare cu 2 expresii de tipul regex pentru a verifica daca acestea reprezinta o adresa ip IPv4 sau o adresa DNS. Am folosit functia compile din libraria re. // de cautat info aici

Functiie sunt apelate in ordinea lor logica.