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

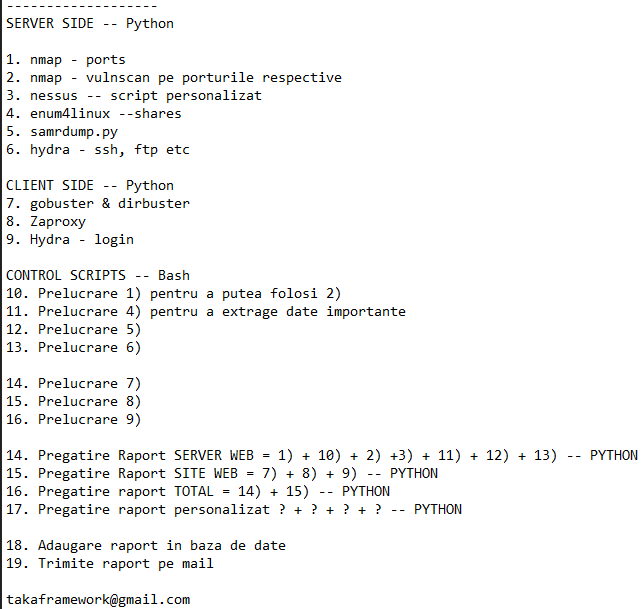
Selectarea programelor care urmeaza sa fie folosite : nmap, nessus, enum4linux, samrdump.py, hydra, gobuster, zaproxy. Pe baza acestora se vor crea scripturi ce le vor rula si prelucra datele de iesire.

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.

Ziua 4 ) 20**.mai.2021**

Scriptul shareEnum.py se foloseste de scriptul in perl enum4linux. Acesta la randul sau se foloseste de mai multe comenzi din pachetul Samba (smbclient rpcclient net ). La fel ca in scriptul pentru nmap, folosesc si in acesta functia verificare\_input() ce compara input-ul introdus cu regex pentru IPv4 si DNS, pentru a evita posibilii vectori de atac ce duc la Shell injection. Functia scanare\_samba() incepe scanarea samba. Problema cu aceasta comanda este ca, desi esti am setat un timeout pentru subprocesul respectiv dupa ce comanda se termina, firul de executie inca rula deoarele am folosit optiunea Shell=True si comanda se trata ca un child process in terminal. Dupa ce am incercat diferite metode am ajuns la concluzia ca cel mai bine sa ma folosesc de comanda timeout din bash. Astfel am setat ca scanarea sa se opreasca dupa 3 minute care sunt mai mult decat suficiente pentru enumerarea folderelor din samba, politica pentru parole, useri s.a.m.d. In general enum4linux enumera useri cu ajutorul SID care este un proces lung si lent si nu este in scopul lucrarii si anume, TakaFramework sa livreze un raport intr-un timp relativ scurt. Din experienta personala, cand enumar Samba primele informatii despre disk-uri, foldere si useri,mai ales daca este posibila enumerarea anonima sunt mai mult decat suficiente pentru a avea un vector de atac asupra infrastructurii serverului/retelei. De aceea am ales ca dupa 180 de secunde sa se forteze terminarea firului de executie. Faptul ca am ales sa fac acest lucru inseamna returnarea unei erori care a fost trada folosind structura try except din python si redirectionarea erorii catre dev null.

O alta problema intampinata a fost faptul ca scriptul enum4linux se folosea de functia print colored() ce coloreaza datele de iesire in terminal. Cum acest lucru este specific pentru bash, cand vreau sa salvez datele intr-un fisier acesta nu recunoaste caracterele speciale iar fisierul era corupt intrucat nu putea prelucra datele fara sa captez si caractele neintelese de Libreoffice. De aceea am modificat scriptul enum4linux.pl pentru a nu afisa datele colorate, inlocuind functia mentionata mai sus cu un print simplu.

Dupa terminarea scanarii folosesc functia verificare\_enum() care verifica daca s-a putut face conectarea la serverul samba sau nu. Acest lucru il realizeaza analizand datele de iesire ale comenzile folosind grep. Am observat ca stringurile   
“No reply” si “Can’t connect” sunt specifice cazului in care nu exista samba. De aceea folosesc grep in combinatie cu wc-l pentru a afla numarul de linii. In cazul in care scanarea a gasit rezultate, comanda va returna stringul 1 iar in cazul in care nu a reusit conectarea se va returna 2. In cazul in care se returneaza 2 se va face automat un raport scurt in care se specifica faptul ca nu s-a putut efectua conectarea ls serverul samba. In celalalt caz, inseamna ca scanarea a fost efectuata cu succes si avem rezultate, iar pentru pregatirea raportului se folosesc scripturi de prelucrare a datelor de iesire. Acest lucru este efectuat de scripturi in bash specifice pentru fiecare lucru cautat.

Ziua 5 ) 2**1.mai.2021**

Terminarea scripturilor

Ziua 6 ) **1.iun.2021**

Configurarea unei masini virtuale de tipul metasploitable si integrarea acesteia in aceeasi retea ca masina virtuala pe care se afla aplicatia de licenta. Acest lucru se poate obseva atunci cand executam in linia de comanda “ip a”. Metasploitable se afla la 192.168.254.136 iar Taka Framework se afla 192.168.254.128. Pentru a verifica conexiunea dintre cele doua folosim comanda ping pentru a putea observa daca cele doua sunt vizibile si accesibile una celeilalte. In urma executarii comenzii putem observa faptul ca cele doua masini pot comunica intre ele iar astfel putem efectua un audit asupra primei masini virtuale mentionate.

\* de explicat ce este metasploitable si DVWA \*

Urmatorul script creat este cel pentru renumitul atac de tip brute force. Pentru acest atac voi folosi liste de parole din renumita colectie “SecLists” de pe git hub.

\*de explicat ce este seclists \*

Din aceasta colectie voi folosi liste atat pentru combinatii de utilizatori si parole cat si pentru descoperirea de fisiere la nivelul site-ului web.

Pentru aceasta demonstratie a utilizarii scriptului am inserat in mod aleatoriu in lista top-usernames-shortlist.txt stringul msfadmin iar in lista 10-million-password-list-top-100.txt acelasi string de asemenea in mod aleatoriu. Aceasta combinatie de utilizator si parola sunt credentialele implicite ale masinii vulnerabile.

Scriptul prezinta aceeasi validare asupra inputului verificata la nivelul unui regex. Acesta contine 3 functii fiecare pentru scanarea unui anumit serviciu. Cele 3 functii contin aceeasi structura try-except ce verifica daca atacul de tip brute force este executat corect. Serviciile care sunt atacate sunt SSH ( Secure Shell ), FTP ( File Transfer Protocol ) si TELNET. Cel din urma este un serviciu instabil iar rezultatele oferite de hydra nu sunt intotdeauna cele corecte intrucat din testele pe care le-am efectuat, se intampla uneori sa nu descopere combinatia “utilizator-parola” corecta astfel rezultand intr-un rezultat null desi acesta fiind cea buna. In general hydra poate oferi valori fals-pozitive, tocmai de aceea este specificat in raportul testului ca fiecare combinatie gasita sa fie testata manual pentru a putea valida rezultatele. O alt pas de tratare a erorilor este acela de timeout al procesului de brute force. Am ales sa fac acest lucru intrucat, uneori procesul din hydra ramane blocat pe un thread astfel continuand sa ruleze chiar daca combinatiile au fost terminate. Comanda “timeout 180s” forteaza inchiderea procesului dupa 4 minute astfel putand evita un bug in proces.

Dupa ce scriptul isi termina executia, se apeleaza scriptul de generare al raportului ce contine rezultatele relevante pentru combinatiile gasite.

Ziua 7 ) 8**.iun.2021**

**\***Am modificat lista pentru dirb

Explica ce este nikto.

Scriptul nikto.py scaneaza cu ajutorul aceluiasi tool serverul primit ca parametru. Dupa ce se efectueaza validarile asupra datelor de intrare incepe scanarea prorpiu zisa. Exista doua functii care fac acest lucru iar singura diferenta dintre acestea este aceea ca cea din urma scaneaza si SSL in cazul in care serverul este hostat si pe portul 443. Pentru a evita ca datele de iesire sa fie prelucrare prost am ales ca aceste datele de iesire ale celor doua scanari sa fie prelucrate separat. Si dupa aceea concatenate intr-un raport final. Acest lucru este efectuat cu ajutorul scriptului de bash raport\_nikto.sh ce se ocupa de prelucrea celor doua fisiere. Acesta ofera de asemenea cateva detalii esentiale legate de ceea ce face. Dupa ce raportul este finalizat, pentru a evita solicitarea spatiului de stocare cele doua fisiere se sterg, astfel ramanand doar fisierul cu raportul alaturi de celelalte rapoarte.

Ziua 7 ) 15**.iun.2021**

**\***Explica ce este zaproxy

Scriptul zap.py este o interfata pentru zaproxy, programul descris mai sus. Diferenta dintre cele doua este ca cel folosit de aplicatia mea se foloseste de optiunea de rulare din linia de comanda, iar acest lucru contribuie la o scanare mai rapida. Datele de iesire ale comenzii nu pot fi salvate decat sunt forma unui fisier xml. Acesta este procesat cu ajutorul librariei XXXX. Raportul final este format din doua parti. Prima parte este un rezumat al vulnerabilitatilor gasite. Acestea sunt grupate dupa numele lor, gradul de risc, numarul de aparitii si o referinta la gradul de risc dupa care s-a efectuat sortarea acestora pentru a putea fi afisate in ordinea gradului de risc la care acestea expun aplicatia.

A doua parte a raportului este compusa din detaliile ce au dus la descoperirea vulnerabilitatii ( uri-ul la care aceasta a fost descoperita, atacul folosit si in ce mod este afectata aplicatia ) si cu o posibila solutie pentru acestea.