UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Embedded Devices Malware Detection, Prevention & Centralization

Andrei Grigoraș

Coordonator științific:
Prof. dr. ing. Cornel Popescu

BUCUREŞTI

2021

CUPRINS

Sinopsis 1

Abstract 1

1 Introducere 2

1.1 Context 2

1.2 Problema 2

1.3 Obiective 2

1.4 Structura lucrării 2

2 Analiza și specificarea cerințelor 3

3 Studiu de piață / Abordări existente 4

4 Soluția propusă 6

5 Detalii de implementare 8

5.1 Tehnologii, limbaje si utilitare client 8

5.1.1 Bash – limbaj de programare 8

5.1.2 Netcat – utilitar pentru transmiterea datelor către client 8

5.1.3 Systemd – responsabil de managementul serviciilor pe sistem 8

5.1.4 HybridAnalysis 8

5.1.5 chattr – utilitar folosit pentru schimbarea atributelor unui fișier. 9

5.1.6 wget – utilitar folosit pentru transferarea datelor de la server folosind protocoale precum HTTP, HTTPS, FTP etc. 9

5.1.7 awk – interpretor pentru limbajul de programare AWK (folosit în manipularea datelor dintr-un fișier) 9

5.1.8 date – utilitar folosind pentru afișarea și setarea datei sistemului 9

5.1.9 hostname – utilitar folosit pentru afișarea si setarea hostname-ului 9

5.1.10 whoami – utilitar folosit pentru afișarea id-ului utilizatorului curent 9

5.1.11 sed – editor de streamuri folosit pentru transformări asupra fișierelor, pipeline-urilor etc. 9

5.1.12 mv – utilitar folosit pentru mutarea sau redenumirea fișierelor. 9

5.1.13 chown – utilitar folosit pentru schimbarea owner-ului si grupului unui fișier 9

5.1.14 chmod – utilitar folosit pentru schimbarea permisiunilor unui fișier 9

5.1.15 mkdir – utilitar folosit pentru crearea de noi directoare 9

5.1.16 touch – utilitar folosit pentru a modifica timestamp-ul fișierelor 9

5.1.17 test – utilitar pentru verificarea tipului unui fișier si compararea valorilor 9

5.1.18 sleep – utilitar folosit pentru întârzierea sistemului 9

5.1.19 echo - utilitar folosit pentru afișarea unei linii de text 9

5.1.20 find – utilitar folosit în căutarea fișierelor într-o ierarhie de directoare 9

5.1.21 rm – utilitar folosit pentru ștergerea de fișiere sau directoare 9

5.1.22 cat – utilitar folosit în concatenarea și afișarea fișierelor 9

5.1.23 wc – utilitar folosind pentru numărarea de cuvinte, bytes sau newlines in fișiere 9

5.1.24 head – utilitar folosit în afișarea primei parți dintr-un fișier 9

5.1.25 exit – utilitar folosit pentru terminarea unui proces 9

5.1.26 service – utilitar folosit pentru rularea unui script de inițializare pentru System V 9

5.1.27 apt – utilitar folosit pentru managementul pachetelor de pe sistem 9

5.1.28 jq – utilitar folosit pentru procesarea de mesaje in formatul JSON 9

5.1.29 bash – interpretor pentru limbajul de programare Bash 9

5.2 Tehnologii, limbaje si utilitare server 10

5.2.1 Python 10

5.2.2 Flask 10

5.3 Instalare & Configurare 11

5.3.1 Server 11

5.3.2 Client 12

5.4 Detecție & Carantinare fișiere malițioase 15

5.5 Raportare si centralizare alerte 21

5.5.1 Transmitere client -> server 21

5.5.2 Recepționare server 22

5.5.3 Centralizare server 22

6. Studiu de caz / Evaluarea rezultatelor 27

7 Concluzii 28

8 Bibliografie 29

SINOPSIS

Scopul acestui proiect este de a implementa un mecanism rapid, portabil, ușor de instalat, și gratis de detecție a fișierelor malițioase ce pot apărea pe un sistem **Unix** ca urmare a descărcării de pe Internet sau de pe un dispozitiv extern. Astfel, am urmărit crearea unei soluții de detectare in limbajul **Bash**, ce este nativ tuturor sistemelor cu sistem de operare **Unix**, care va servi in implementarea unei arhitecturi client-server (in cazul nostru, mai mulți clienți, un singur server).

În urma unei investigări pentru a găsi software echivalent, am observat că nu există soluții gratis ce oferă compatibilitate pentru orice platforma Unix, lucru ce a reprezentat o oportunitate în acest sens. Din acest motiv, proiectul se adresează dispozitivelor embedded din categoria „Internet of Things", și urmărește monitorizarea unei rețele de acest tip de dispozitive (clienți IoT) prin intermediul unui software ce va detecta si carantina fișierele malițioase de pe sistem. Serverul va permite, printr-o aplicație web, colectarea si centralizarea alertelor la nivel de rețea.

Cuvinte cheie: **malware**, **Unix**, **gratis**, **embedded devices**, **IoT**, **rețea**, **detecție** si **prevenire**.

ABSTRACT

The goal of this project was to implement a fast, portable, easy to install, and free mechanism which can detect malicious files that enter a **Unix** system via Internet (download) or external devices (USB, CD). Therefore, I looked for creating a solution using the **Bash** scripting language, which is the native language used by **Unix** systems, that can serve in implementing a client-server architecture (multiple clients, one server).

After a thorough investigation to find similar software (free and deployable on any Unix system), I noticed that the market lacks any such solution. Therefore, the project targets embedded devices such as the one used in "Internet of Things" and performs continuous monitoring of a network of such devices. This is done via software that detects and monitors any newly malicious file on the systems as well as a web application that is deployed on the server which centralizes all alerts within the network.

Keywords: **malware**, **Unix**, **free**, **embedded devices**, **IoT**, **network**, **detection,** and **prevention**.

1 INTRODUCERE

1.1 Context

Dorința implementării acestei soluții a venit ca urmare a unei curiozități de creare a unui software de tip **antivirus**, folosind doar resurse gratuite de tipul **open-source** care, pe baza unui fișier încărcat, determină dacă fișierul este sau nu malițios. Pe de altă parte, proiectul implementat diferă de o soluție clasică de antivirus prin prisma pieței ce o urmărește, aceea a dispozitivelor embedded (precum cele din IoT) ce oferă resurse limitate la nivelul sistemului de operare, făcând astfel imposibilă instalarea unei soluții precum Avira/Bitdefender/Avast. (ce sunt specifice sistemului de operare Windows).

Deși există mai multe soluții open-source care pot detecta malware prezent pe un sistem, niciuna din aceste soluții nu pot detecta pasiv si continuu si nici nu oferă o modalitate de a centraliza astfel de alerte la nivel de rețea.

1.2 Problema

Prima problema pe care proiectul o abordează e cea a nevoii unui software de tipul antivirus care sa funcționeze pe orice sistem Unix. O a doua problema este cea financiara întrucât pe piață există astfel de soluții (mult mai complexe) denumite „Host based Intrusion Detection System" sau „Endpoint detection and response" dar care necesită achiziționarea unui abonament/licență.

1.3 Obiective

Obiectivul principal al proiectului este de a reuși cu succes să detectăm și eradicăm fișierele malițioase noi apărute pe mai multe sistem Unix cât și centralizarea grafică a acestor alerte printr-o aplicație web. Succesul proiectului este determinat de existența unei soluții open-source ce expune un API prin care se poate concluziona dacă un fișier este malițios sau nu.

1.4 Structura lucrării

În secțiunea următoare, analizăm și specificăm cerințele ce au fost luate în considerare în implementarea proiectului cât și abordarea care a garantat îndeplinirea lor.

După, facem o analiză a pieței, soluțiilor existente, cât și factorii prin care soluția propusă se evidențiază în acest domeniu. Pornim de la contextul atacurilor de tip malware și sfârșim prin a înțelege utilitatea soluției noastre în industria IoT.

În secțiunile 4 si 5 discutăm despre soluția propusa (arhitectura, diagrama, cazuri de utilizare) cât și detalii specifice precum bucăți de cod, funcționalități si limitări.

În secțiunea 6, evaluăm performanțele proiectului prin crearea unui mediu de test și simulând un utilizator. Astfel puteam deduce timpul de răspuns, eficiența dar și corectitudinea datelor.

Ultimele secțiuni sunt adresate concluziilor și bibliografiei folosite.

2 ANALIZA ȘI SPECIFICAREA CERINȚELOR

În vederea implementării unui proiect care să aducă un plus industriei IT, acesta trebuie sa îndeplinească anumite cerințe ce sunt direct proporționale cu nivelul curent de evoluție al industriei. Astfel, urmând specificațiile serviciilor de succes din industrie precum Amazon si Google, am compus următoarea listă de cerințe pentru partea ce **rulează pe client**, și care trebuie luate în considerare pe parcursul dezvoltării soluției:

- scalabilitate
- disponibilitate
- viteza cât mai ridicată a procesării
- costuri cât mai reduse
- resurse de calcul cât mai scăzute (evitare overhead)
- ușor de instalat si configurat

Abordarea propusă se pliază ușor pe aceste criterii întrucât, folosind un limbaj nativ sistemului de operare ce suporta și multi-threading (Bash), atât criteriul de scalabilitate cât și cel de viteză sunt ușor de îndeplinit. Pentru a facilitata instalarea ușoară a produsului, am realizat un script care, în urma execuției, va realiza toți pașii necesari pregătirii și instalării de software. De asemenea, întrucât produsul final este open-source, am adăugat comentarii pentru fiecare bloc de cod, dar și indicații în cazul în care se dorește configurarea produsului cu alți parametri/opțiuni.

Întrucât scopul proiectului a fost de a crea un produs folosind doar resurse gratuite open-source, costul instalării și folosirii este zero. De asemenea, deoarece procesarea se face prin intermediul soluțiilor open-source ce rulează în **cloud**, resursele necesare rulării sunt minime și sunt strict dependente de calitatea conexiunii la Internet și la rețea. (pentru interacțiunea cu API-ul și transmiterea alertelor de la clienți către server în rețea)

Nu în ultimul rând, pentru a asigura disponibilitatea, produsul a fost testat și remediat de bug-uri/probleme prin emularea unui mediu cu 4 clienți și 1 server. De asemenea, software-ul a fost instalat ca și serviciu ceea ce înseamnă că, va fi rulat automat atunci când sistemul pornește iar în cazul unei defecțiuni/erori, serviciul se va reporni automat.

Ce trebuie luat în considerare este că, principiile enumerate mai sus nu au fost prioritatea numărul unu, motiv pentru care, o abordare folosind alte tehnologii ar fi putut să îmbunătățească performanțele generale ale produsului final. Acest lucru este datorat faptului că, prioritatea principală a fost crearea unei soluții care să poată detecta fișiere malițioase pe **orice** sistem Unix (chiar dacă nu există **Python**, **PHP**, **SQL** sau un manager de pachete instalat) și să nu implice niciun cost aferent instalării/rulării. Totuși, s-au ținut cont de criteriile standard pentru un produs ce poate concura pe piață cu altele din aceeași categorie. (majoritatea fiind mult mai avansate si mai complexe)

3 STUDIU DE PIAȚĂ / ABORDĂRI EXISTENTE

Pentru a putea implementa o soluție cât mai bună și care să poată fi folosită mai mult decât un „Proof of Concept”, am analizat piața aferentă ariei pe care produsul o adresează, în cazul nostru fiind cea alcătuită din software ce ajută în detectarea de malware.

Înainte de a trece la analiza pieței, trebuie să înțelegem contextul actual din punct de vedere al securității și al alertelor de malware. În anul 2020, a crescut considerabil popularitatea virușilor pentru sistemele Linux întrucât acestea, de multe ori, sunt reprezentate de servere web sau baze de date, motiv pentru care pot cauza daune ridicate unei afaceri odată cu compromiterea lor. Malware precum RansonmEXX, Gitpaste-12 si IPStorm au făcut ca industria de soluții împotriva atacurilor cibernetice pentru sisteme Unix să crească.

În analiza pieței, am consultat tipurile de soluții ce pot concura cu proiectul nostru, acestea împărțind-se în mai multe categorii, următorul tabel evidențiind asemănările și diferențele cu proiectul prezentat în acest document:

	Host Based Intrusion Detection Systems	Endpoint Detection and Response	Proiect Licență	Alte tool-uri open-source
Gratis			✓	✓
Arhitectura client - server	✓	✓	✓	✓
Detectare malware	✓	✓	✓	✓
Scalabilitate > 20 client	✓	✓		✓
Centralizare alerte in aplicație web	✓	✓	✓	✓
Compatibilitate Unix	✓	✓	✓	✓
Fără dependențe de sistem			✓	

Un sistem de detectare a intruziunilor bazat pe gazdă (HIDS) este un sistem de detectare a intruziunilor care este capabil să monitorizeze și să analizeze internele unui sistem de calcul precum și pachetele de rețea de pe interfețele sale de rețea, similar modului în care un sistem de detectare a intruziunilor bazat pe rețea (NIDS) funcționează. Acesta a fost primul tip de software de detectare a intruziunilor care a fost proiectat, sistemul țintă original fiind computerul mainframe unde interacțiunile externe erau rare.

Tehnologia de detectare și răspuns a punctelor finale este utilizată pentru a proteja punctele finale, care sunt dispozitive hardware pentru computer, de amenințări. Creatorii platformelor bazate pe tehnologie EDR implementează instrumente pentru a culege date de pe dispozitivele endpoint și apoi analizează datele pentru a dezvălui potențiale amenințări și probleme cibernetice. Este o protecție împotriva încercărilor de hacking și furtului datelor utilizatorilor. Software-ul este instalat pe dispozitivul utilizatorului final și este monitorizat continuu. Datele sunt stocate într-o bază de date centralizată. Într-un incident în care se găsește o amenințare, utilizatorul final este imediat solicitat cu o listă preventivă de acțiuni.

Fiecare platformă EDR are setul său unic de capabilități. Cu toate acestea, unele capabilități comune includ monitorizarea punctelor finale atât în modul online, cât și offline, răspunderea la amenințări în timp real, creșterea vizibilității și transparenței datelor utilizatorilor, detectarea evenimentelor punctelor finale ale magazinului și a injecțiilor de malware, crearea listelor negre și a listelor albe și integrarea cu alte tehnologii.

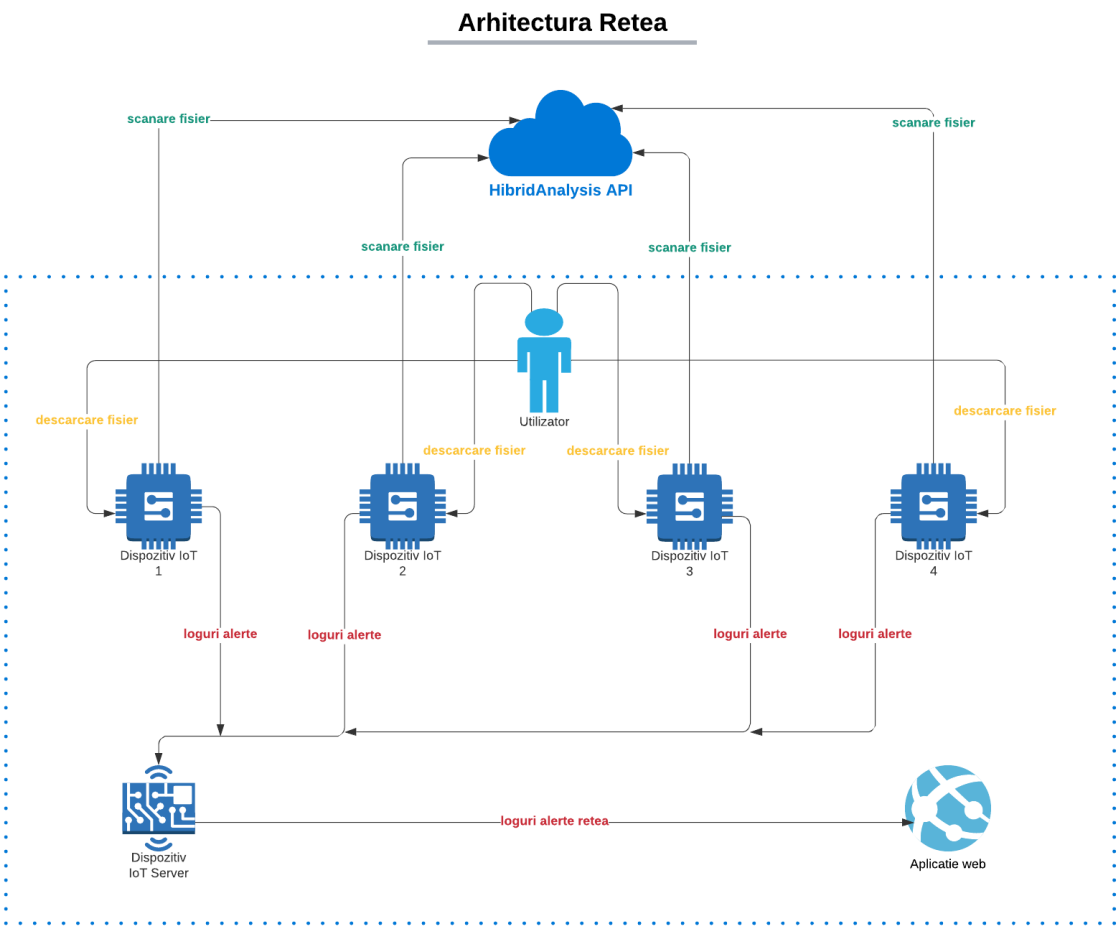
Categoriile de HIDS și EDR sunt adresate mediilor corporate întrucât sunt construite pentru a scala pentru mii de sisteme. De asemenea, acestea încorporează o sumedenie de funcționalități de prevenire și detecție a incidentelor de securitate pe un host (nu doar detecție de malware). Problema acestor soluții este reprezentată, în primul rând, de partea financiară întrucât implică costuri ridicate în funcție de dimensiune rețelei și de pachetul de funcționalități ales. Printre cele mai cunoscute astfel soluții, avem: SolarWinds, Splunk, RedCloak, CarbonBlack, FireEye. Costul mediu pentru o astfel de soluție este în medie 50 dolari / sistem / an de unde și necesitatea unei soluții ieftine de detecție malware pentru rețele mici – medii ca și dimensiune. (< 20 stații)

Pe de altă parte, există câteva soluții gratis, open-source care pot scana un sistem și detecta diverse tipuri de malware dar și alte tipuri de probleme precum configurații greșite, software cu versiuni ne-updatate etc. Una din cele mai cunoscute astfel de soluții este Wazuh/OSSEC. Deși această soluție este net superioară celei propuse de noi, ea nu se adresează tuturor sistemelor Unix și depinde de existența unui „package manager" sau a unei soluții precum **Docker** instalate la nivelul sistemului de operare (pentru a seta mediul de rulare al software-ului). În cazul dispozitivelor embedded ce au un kernel personalizat, aceste dependențe nu pot fi îndeplinite.

Putem concluzia astfel că, piața curentă reprezintă o oportunitate pentru un software ca cel propus de proiectul nostru care, deși nu va fi monetizat, va reprezenta un plus pentru industria securității cibernetice pentru dispozitive embedded.

4 SOLUȚIA PROPUSĂ

Soluția propusă se adresează unei arhitecturi cu mai mulți clienți, ce reprezintă dispozitive IoT, și de un server, tot un dispozitiv IoT. Software-ul ce rulează pe clienți va încărca periodic fișierele nou apărute în sistem pe platforma HybridAnalysis folosind API-ul expus de acesta. Dacă fișierul este detectat ca fiind malițios, se adaugă într-un fișier de log care, la rândul său, este transmis către server pentru fiecare alertă nou apărută. Acesta, serverul, are rol în colectarea log-urilor de la toți clienții (alerte de fișiere potențial malițioase) și de centralizare printr-o aplicație web ce va afișa informații precum: IP și hostname client, număr alerte în ultimele 24 ore cât și un sumar al acestor fișiere. Mai jos, în figura 1, avem arhitectura folosită în cadrul acestui proiect.



Figură 1 Arhitectura proiect

Workflow-ul arhitecturii propuse poate fi redus la următoarea lista de evenimente:

- Din cauza acțiunilor utilizatorului, un nou fișier apare pe sistem
- Fișierul este automat colectat de software și trimis automat pentru analiza, prin intermediul unui endpoint, către platforma HibridAnalysis ce are rol in compararea acestuia cu o bază de date ce conține semnături de fișiere malițioase.

- In urma analizei, verdictul se returnează către software.
- Daca fișierul a fost detectat ca malițios, acesta se mută, din directorul curent, într-un folder special la care doar administratorul de sistem are acces. De asemenea, se creează o alertă nouă ce este introdusa într-un fișier de log.
- La un interval de timp prestabilit, un serviciu instalat de software, verifică fișierul de log și, dacă există măcar o alertă, consumă și trimite informațiile către server-ul din rețeaua de dispozitive.
- Serverul, primește informațiile (alerte) de la toți clienții din rețea și le centralizează într-o aplicație web care facilitează monitorizarea tuturor alertelor de fișiere posibil malițioase din rețea.

Soluția propusă nu are nicio dependință sau componentă hardware întrucât este reprezentată de o multitudine de servicii (software) care trebuie instalate atât pe clienți cât și pe un dispozitiv ce va servi ca și server. Corectitudinea soluției este strâns dată de 2 factori:

- corectitudinea detecției soluției oferite de HybridAnalysis prin API-ul expus
- transmiterea corectă și în timp eficient a datelor de la client la API și la server (să nu fie corupte sau să nu ajungă deloc)

5 DETALII DE IMPLEMENTARE

Înainte de a trece propriu zis la detaliile de implementare, vom descrie pe scurt tehnologiile folosite în implementarea soluției atât pentru partea de client cât și pentru cea de server.

5.1 Tehnologii, limbaje si utilitare client

În implementarea clientului, am folosit următoarele utilitare, limbaje și tehnologii ce vin preinstalate pe un sistem Unix:

5.1.1 Bash – limbaj de programare

Bash este un interpretor de comenzi Unix scris inițial de către Brian Fox de la Fundația pentru Software Liber pentru Proiectul GNU.

Numele este un acronim, un joc de cuvinte și o descriere. Ca acronim, vine de la Bourne-again shell, referindu-se la obiectivul său ca înlocuitor liber pentru Bourne shell. Ca joc de cuvinte, exprimă acest obiectiv într-o formă ce sună similar cu sintagma naștere din nou. Numele descrie de asemenea realizarea sa, îngemănarea funcțiilor din sh, csh și ksh.

5.1.2 Netcat – utilitar pentru transmiterea datelor către client

Netcat – precum multe alte instrumente de hacking – a fost creat cu scopul de a analiza rețelele. Dezvoltat de cineva cunoscut doar ca “Hobbit”, el a oferit acest instrument comunității IT, fără compensație, dar a primit numeroase premii.

Ca atare, îl putem utiliza pentru a deschide conexiuni TCP și UDP între două computere, pe orice port specificat de utilizator. Acesta poate fi, de asemenea, folosit ca un instrument de scanare a porturilor, similar cu nmap. În plus, acesta poate fi utilizat pentru port forwarding, proxying, servere web simple, dar și lăsarea unui backdoor pentru atacatori.

5.1.3 Systemd – responsabil de managementul serviciilor pe sistem

Systemd este un manager de sistem și servicii pentru Linux, compatibil cu Initscript SysV și LSB. Systemd oferă o abilitate remarcabilă de a paraleliza utilizarea socket-urilor și activarea D-Bus pentru a porni servicii, permite pornirea demonilor la cerere, urmărirea proceselor cu utilizarea grupurilor de control Linux, asistență la instantanee și restabilirea stării sistemului, menține punctele de asamblare și serviciile de asamblare automată și pune în aplicare un sistem elaborat de gestionare a dependenței bazat pe un control logic al serviciilor.

5.1.4 HybridAnalysis

HybridAnalysis este un serviciu independent, alimentat de Falcon Sandbox și oferă un subset de capabilități aferente Falcon Sandbox. CrowdStrike Falcon Sandbox este o soluție automată de analiză malware ce efectuează analize profunde ale amenințărilor evazive și necunoscute, îmbogățește rezultatele cu informații despre amenințări și furnizează indicatori de compromis (IOC).

HybridAnalysis este o abordare de analiză a fișierelor care combină datele de execuție cu analiza de memorie pentru a extrage toate căile de execuție posibile chiar și pentru cele mai evazive programe malware.

5.1.5 chattr – utilitar folosit pentru schimbarea atributelor unui fișier.

5.1.6 curl&wget – utilitar folosit pentru transferarea datelor de la server folosind protocoale precum HTTP, HTTPS, FTP etc.

5.1.7 awk – interpretor pentru limbajul de programare AWK (folosit în manipularea datelor dintr-un fișier)

5.1.8 date – utilitar folosind pentru afișarea și setarea datei sistemului

5.1.9 hostname – utilitar folosit pentru afișarea si setarea hostname-ului

5.1.10 whoami – utilitar folosit pentru afișarea id-ului utilizatorului curent

5.1.11 sed – editor de streamuri folosit pentru transformări asupra fișierelor, pipeline-urilor etc.

5.1.12 mv – utilitar folosit pentru mutarea sau redenumirea fișierelor.

5.1.13 chown – utilitar folosit pentru schimbarea owner-ului si grupului unui fișier

5.1.14 chmod – utilitar folosit pentru schimbarea permisiunilor unui fișier

5.1.15 mkdir – utilitar folosit pentru crearea de noi directoare

5.1.16 touch – utilitar folosit pentru a modifica timestamp-ul fișierelor

5.1.17 test – utilitar pentru verificarea tipului unui fișier si compararea valorilor

5.1.18 sleep – utilitar folosit pentru întârzierea sistemului

5.1.19 echo - utilitar folosit pentru afișarea unei linii de text

5.1.20 find – utilitar folosit în căutarea fișierelor într-o ierarhie de directoare

5.1.21 rm – utilitar folosit pentru ștergerea de fișiere sau directoare

5.1.22 cat – utilitar folosit în concatenarea și afișarea fișierelor

5.1.23 wc – utilitar folosind pentru numărarea de cuvinte, bytes sau newlines in fișiere

5.1.24 head – utilitar folosit în afișarea primei parți dintr-un fișier

5.1.25 exit – utilitar folosit pentru terminarea unui proces

5.1.26 service – utilitar folosit pentru rularea unui script de inițializare pentru System V

5.1.27 apt – utilitar folosit pentru managementul pachetelor de pe sistem

5.1.28 jq – utilitar folosit pentru procesarea de mesaje in formatul JSON

5.1.29 bash – interpretor pentru limbajul de programare Bash

5.2 Tehnologii, limbaje si utilitare server

Pe lângă cele deja menționate pe partea de client, server-ul dispune si de o aplicație web prin care se centralizează alertele la nivel de rețea. Aplicația web a fost construită folosind următoarele tehnologii:

5.2.1 Python

Python este un limbaj de programare dinamic multi-paradigmă, creat în 1989 de programatorul olandez Guido van Rossum. Este un limbaj multifuncțional folosit de exemplu de către companii ca Google sau Yahoo! pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python. Popularitatea în creștere, dar și puterea limbajului de programare Python au dus la adoptarea sa ca limbaj principal de dezvoltare de către programatori specializați și chiar și la predarea limbajului în unele medii universitare. Din aceleași motive, multe sisteme bazate pe Unix, inclusiv Linux, BSD și Mac OS X includ din start interpretatorul CPython.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul object-oriented, dar permite și programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu de garbage collection. Alt avantaj al limbajului este existența unei ample biblioteci standard de metode.

5.2.2 Flask

Flask este un micro framework web scris în Python. Este clasificat ca microframe, deoarece nu necesită anumite instrumente sau biblioteci. Nu are un strat de abstractizare a bazei de date, validarea formularelor sau alte componente în care bibliotecile terțe preexistente oferă funcții comune. Cu toate acestea, Flask acceptă extensii care pot adăuga caracteristici aplicației ca și cum ar fi implementate în Flask. Există extensii pentru validarea formularelor, gestionarea încărcărilor, diverse tehnologii de autentificare și mai multe instrumente comune legate de cadru. Printre aplicațiile care utilizează Flask se numără LinkedIn și Pinterest.

În continuare, vom aborda următoarele categorii din punct de vedere al implementării atât pentru client cât și pentru server:

- Instalare & Configurare
- Detecție & Carantinare fișiere malițioase
- Raportare & Centralizare alerte

5.3 Instalare & Configurare

5.3.1 Server

Primul pas pentru instalarea completa a soluției este cea de introducere a fișierelor necesare pe server. Acest lucru se poate realiza prin intermediul unei descărcări directe de pe internet cât și prin inserarea unui dispozitiv periferic precum USB, CD, DVD, sau Hard Disk. Totuși, abordarea aleasă, în scop demonstrativ, a fost hostarea fișierelor triviale instalării pe un server în cloud-ul **Linode**, platforma ce oferă soluții rapide pentru hostarea unei mașini virtuale online. Astfel, pentru a instala soluția pe server, este nevoie doar de executarea scriptului „run.sh” ce poate fi descărcat de pe serverul de Linode cu IP-ul **139.162.230.80**. Pașii pe care utilizatorul trebuie sa îi urmeze cât și cei pe care scriptul îi executa sunt următorii:

1. Utilizatorul instalează dependințele necesare pentru a putea rula serviciul web de centralizare a alertelor (aplicație de Flask) cât și pentru hostarea fișierelor necesare clienților (serviciu Apache).

```
apt-get install apache2
apt-get install python
apt-get install python3-pip
apt-get install curl
pip3 install Flask
```

2. Utilizatorul descarcă scriptul „run.sh”, îl marchează ca executabil (+x) si îl execută.

```
wget -q http://139.162.230.80/licenta/run.sh -O run.sh
chmod +x run.sh
./run.sh
```

3. Scriptul instalează serviciul **apache2** pentru hostarea fișierelor necesare clienților.

```
# Install and run apache2 in order to host the webserver
check_status apt-get install -qq apache2 $RED"[!] Apache2 could not
be installed!"$ANSII_END
check_status service apache2 start $RED"[!] Apache2 could not be
started!"$ANSII_END
echo -e $GREEN"[+] Apache2 up and running."$ANSII_END
```

4. Scriptul descarcă toate fișierele necesare instalării soluției atât pentru server cât și pentru client și le plasează ori in directorul specific serviciului Apache (/var/www/html) ori în folderul de root.

```
# Download all the necessary files from the Github repository.
mkdir -p /root/static/image
mkdir -p /root/template/
check_status wget -q http://139.162.230.80/licenta/server.sh -O
/root/server.sh $RED"[!] server.sh script failed while
downloading!"$ANSII_END
check_status wget -q http://139.162.230.80/licenta/blacklister.sh -O
/var/www/html/blacklister.sh $RED"[!] blacklister.sh script failed
while downloading!"$ANSII_END
```

```
check_status wget -q http://139.162.230.80/licenta/reporter.sh -O
/var/www/html/reporter.sh $RED"[!] reporter.sh script failed while
downloading!"$ANSII_END
check_status wget -q http://139.162.230.80/licenta/install.sh -O
/var/www/html/install.sh $RED"[!] reporter.sh script failed while
downloading!"$ANSII_END
check_status wget -q http://139.162.230.80/licenta/logger.py -O
/root/logger.py $RED"[!] logger.py script failed while
downloading!"$ANSII_END
check_status wget -q
http://139.162.230.80/licenta/static/image/refresh.png -O
/root/static/image/refresh.png $RED"[!] logger.py script failed while
downloading!"$ANSII_END
check_status wget -q
http://139.162.230.80/licenta/template/index.html -O
/root/template/index.html $RED"[!] logger.py script failed while
downloading!"$ANSII_END
```

5. În final, se rulează 2 scripturi auxiliare, unul ce pornește aplicația web pe portul 8080 și altul ce ascultă pe portul 1337 pentru a primi alerte de la clienții din retea.

```
echo -e $GREEN"[+] Running server and web application"$ANSII_END
python /root/logger.py > /dev/null 2>&1 &
bash /root/server.sh
```

6. Daca toți pașii s-au executat cu succes, pe server ar trebui sa avem următorul output în urma rulării comenzilor de descărcare și rulare a celor 2 scripturi auxiliare.

```
root@server:~# wget -q http://139.162.230.80/licenta/run.sh -O run.sh
root@server:~# chmod +x run.sh
root@server:~# ./run.sh
[+] Apache2 up and running.
[+] Files successfully downloaded.
[+] Running server and web application
```

7. Verificam ca netcat ruleaza pe portul 1337, python pe 8080 și Apache pe 80.

```
root@server:~# ss -tunlp
Netid  State   Recv-Q  Send-Q  Local Address:Port   Peer Address:Port
udp    UNCONN  0       0       127.0.0.53%lo:53     0.0.0.0:*      users:(("systemd-resolve",pid=72,fd=12))
udp    UNCONN  0       0       10.0.3.146%eth0:68   0.0.0.0:*      users:(("systemd-network",pid=49,fd=19))
tcp    LISTEN  0       128     0.0.0.0:8080         0.0.0.0:*      users:(("python",pid=752763,fd=3))
tcp    LISTEN  0       128     127.0.0.53%lo:53     0.0.0.0:*      users:(("systemd-resolve",pid=72,fd=13))
tcp    LISTEN  0       1       0.0.0.0:1337         0.0.0.0:*      users:(("nc",pid=752766,fd=3))
tcp    LISTEN  0       511     *:80                 *:*            users:(("apache2",pid=94,fd=3),("apache2",pid=9
3,fd=3),("apache2",pid=92,fd=3))
```

5.3.2 Client

Având partea de server deja configurată corespunzător, pe partea de client trebuie să descărcam, de pe server (având IP-ul 10.0.3.146), scriptul „install.sh” pe care îl vom executa dându-i ca unic parametru adresa IP a server-ului. Astfel, pașii pe care utilizatorul trebuie să îi urmeze cât și cei pe care scriptul îi executa sunt următorii:

1. Utilizatorul descarcă scriptul „install.sh” de pe server și îl executa cu adresa IP a serverului.

```
wget -q http://10.0.3.146/install.sh -O install.sh
chmod +x install.sh
./install.sh 10.0.3.146
```

2. Scriptul descarcă și instalează serviciul responsabil cu detectarea fișierelor malițioase.

```
# Setting up the malware detection service
create_service $URL_MALWARE blacklister
```

3. Scriptul descarcă și instalează serviciul responsabil de raportarea alertelor de pe sistem.

```
# Setting up the log reporter service
create_service $URL_REPORTER reporter $1
```

4. Daca toți pașii s-au executat cu succes, pe client ar trebui sa avem următorul output

```
=================== Setting up the blacklister service ===================
[+] Script downloaded successfully.
[+] Service created and successfully started.
=================== Setting up the reporter service ===================
[+] Script downloaded successfully.
[+] Service created and successfully started.
==========================================================================
```

5. De asemenea, pentru a verifica faptul ca serviciile au pornit cu succes, putem folosi utilitarul service pentru a interoga starea acestora și a confirma că funcționează.

```
root@client1:~# service blacklister status
● blacklister.service
   Loaded: loaded (/etc/systemd/system/blacklister.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-06-20 21:50:55 UTC; 2min 10s ago
 Main PID: 1984957 (blacklister.sh)
    Tasks: 2 (limit: 19112)
   CGroup: /system.slice/blacklister.service
           ├─1984957 /bin/bash /usr/local/bin/blacklister.sh
           └─1985247 sleep 5

Jun 20 21:50:55 client1 systemd[1]: Started blacklister.service.
Jun 20 21:50:55 client1 systemd[1]: blacklister.service: Failed to reset devices.list: Operation not permitted
Jun 20 21:50:55 client1 systemd[1]: blacklister.service: Failed to reset devices.list: Operation not permitted
root@client1:~# service reporter status
● reporter.service
   Loaded: loaded (/etc/systemd/system/reporter.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-06-20 21:52:09 UTC; 58s ago
 Main PID: 1985058 (reporter.sh)
    Tasks: 2 (limit: 19112)
   CGroup: /system.slice/reporter.service
           ├─1985058 /bin/bash /usr/local/bin/reporter.sh 10.0.3.146
           └─1985282 sleep 1

Jun 20 21:52:09 client1 systemd[1]: Started reporter.service.
root@client1:~#
```

Pentru instalarea acestor servicii, s-a folosit următoarea funcție implementată în Bash ce generează un serviciu prin adăugarea unei intrări noi in folderul /etc/systemd. Această funcție primește 2 parametri prin care se specifică numele noului serviciu cât și URL-ul de unde poate descarcă scriptul ce trebuie rulat de serviciu.

```
# Function that can set up a service given the script URL and service name
function create_service {
    echo -e $YELLOW"==== Setting up the $2 service ===="$ANSII_END

    # Download the blacklister script and set it properly
    check_status wget $1 -q -O $2.sh --timeout=2  --tries=2 $RED"[!] The
resource is unavailable. Please check your internet connection!"$ANSII_END
```

```
    echo -e $GREEN'[+] Script downloaded successfully.'$ANSII_END

    mv $2.sh /usr/local/bin/
    chmod 755 /usr/local/bin/$2.sh


    # Create a new service on the host and set it properly
    cat <<- EOF > /etc/systemd/system/$2.service
    [Unit]
    Description=$2 service
    After=network.target

    [Service]
    ExecStart=/bin/bash /usr/local/bin/$2.sh $3
    Restart=always
    RestartSec=3

    [Install]
    WantedBy=default.target
    EOF

    check_status chmod 664 /etc/systemd/system/$2.service $RED"[!] Something bad happened while setting up the service."$ANSII_END
    check_status systemctl daemon-reload $RED"[!] Something bad happened while setting up the service."$ANSII_END
    check_status systemctl enable $2.service 2> /dev/null $RED"[!] Something bad happened while setting up the service."$ANSII_END
    check_status systemctl start $2.service $RED"[!] Something bad happened while setting up the service."$ANSII_END

    echo -e $GREEN"[+] Service created and successfully started."$ANSII_END
}
```

De asemenea, s-au mai folosit si alte 2 funcții auxiliare. Una dintre ele verifică dacă o comandă a generat o eroare, moment în care scriptul se va termina imediat. Cealaltă funcție a fost folosită pentru a testa dacă două variabile au aceeași valoare.

```
# Create a functions that check if a specific command ended with an error or not. If any error occured, display a message and exit
function check_status {
    "${@:1:($#-1)}"
    local status=$?
    local last_param=${@: -1}
    if [ $status -gt 0 ]
    then
        echo -e $last_param
        exit 1
    fi
}


# Function that tests if 2 parameters are equals. If they differ, prints a message and exists the script
function test_statement {
    if [[ "$1" != $2 ]]
    then
        echo -e $3
        exit 1
    fi
}
```

5.4 Detecție & Carantinare fișiere malițioase

În această secțiune urmărim îndeaproape detaliile de implementare ce realizează atât detecția fișierelor malițioase nou apărute pe sistem cât și carantinarea acestora pentru a împiedica user-ul din a le deschide sau executa. Acest proces se execută doar la nivelul clientului și constă din serviciul „blacklister” ce executa scriptul „blacklist.sh” aflat in folderul /usr/local/bin. Pașii pe care scriptul ii urmează metodic in detecția fișierelor malițioase sunt următorii:

1. Se caută folosind comanda **find** toate fișierele care au fost modificate sau accesate în ultimele 6 secunde (0.1 minute).

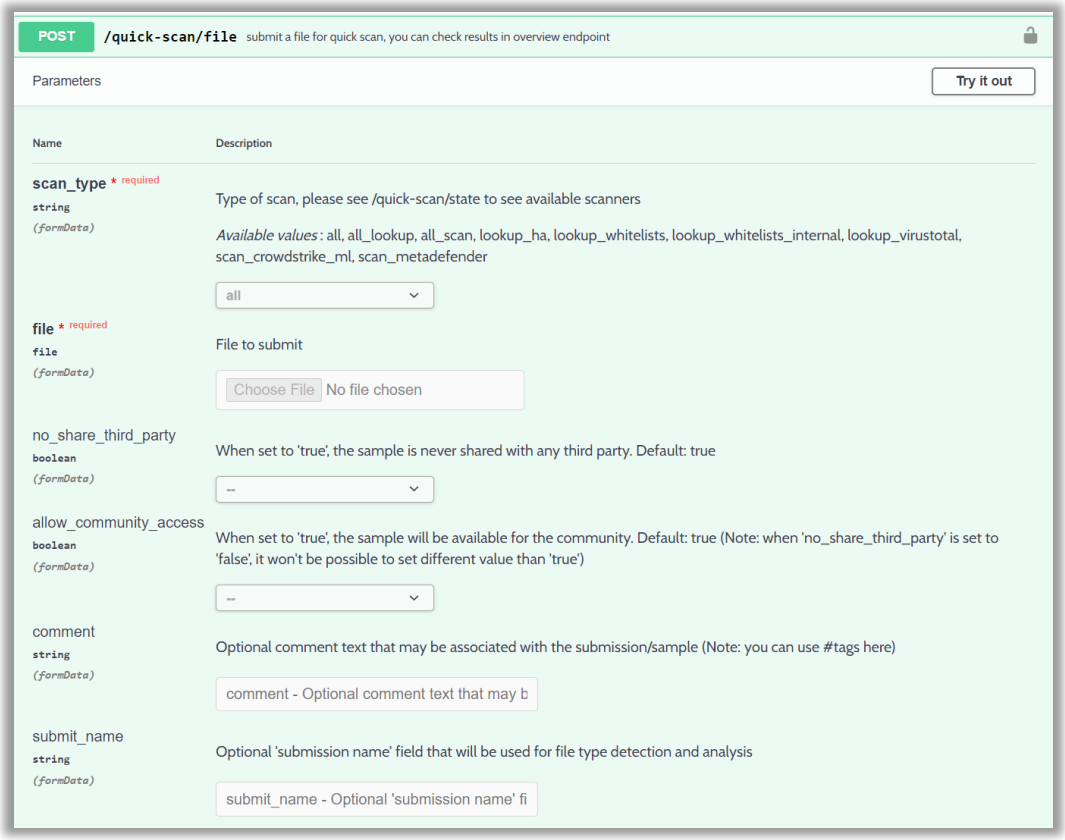
```
# Check every 6 seconds for new files on the system. As users should
only have write access to their home directory ONLY, monitor that
directory only.
# This can be easily changed to match multiple directories or match
all except some (-not -path)
new_files=`find /home -ignore_readdir_race -type f -mmin -0.1 2>
/dev/null`
```

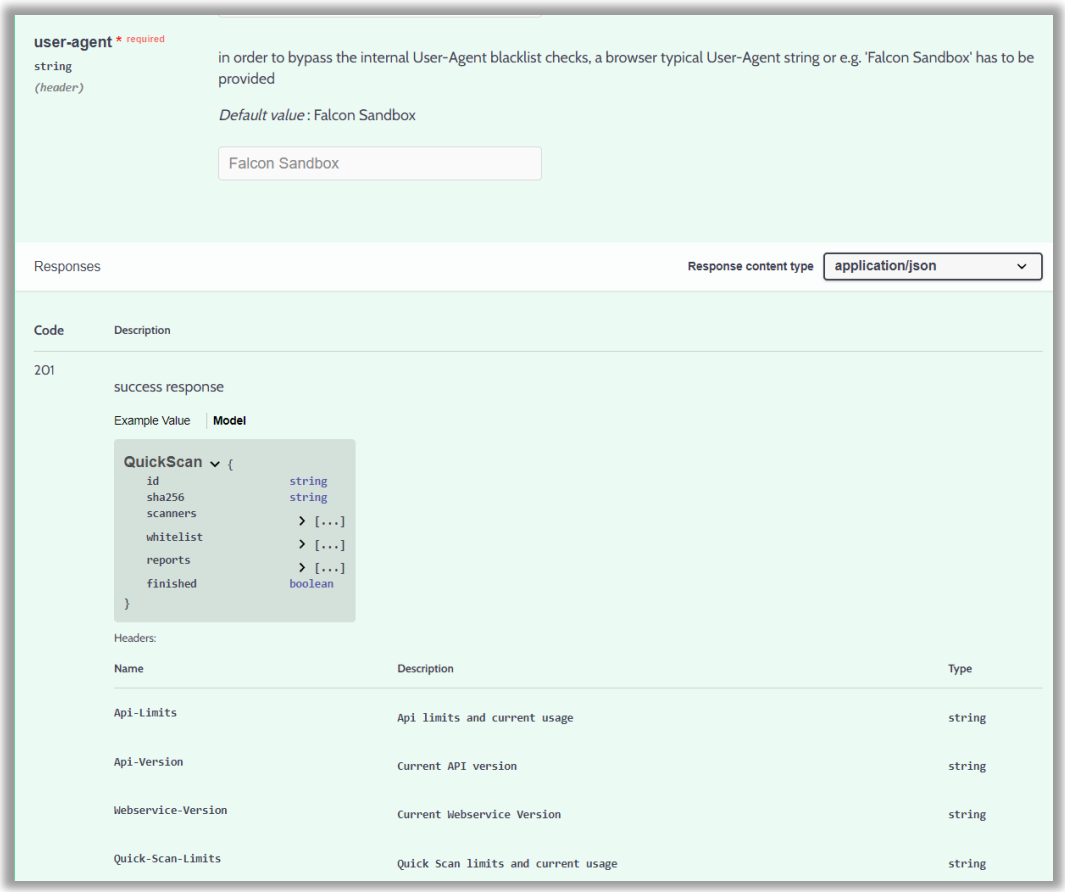
2. Pentru fiecare fișier obținut la subpunctul 1, se apelează funcția **process_file** având ca parametru calea absolută către fișier. La finalul instrucțiunii „for”, se așteaptă 5 secunde până la următoarea verficiare pentru a evita detectarea aceluiași fișier.

```
# Submit them to HybridAnalysis by spawning the function in the
background.
for i in $new_files
do
    process_file $i &
done
```

3. Funcția „process_file” are ca scop încărcarea fișierului în platforma HybridAnalysis folosind API-ul expus de aceasta și vizualizarea rezultatului ce conține informații despre tipul de fișier încărcat (**empty, clean, suspicious, malicious**). Apelul către endpoint-ul de API este public și necesită următoarele pentru a uploada un fișier:

```
function process_file(){
    # Construct the request to the API using the official schema.
    submission_name="`hostname`_`hostname -I | sed 's/ //g'`_`date
+'%s'`_`echo $1 | awk -F '/' '{print $NF}'`"
    resp=`curl -X POST --silent \
                    -H "User-Agent: Falcon Sandbox" \
                    -H "Accept: application/json" \
                    -H "Content-Type: multipart/form-data" \
                    -H "Api-Key: $API_KEY" \
                    -F "scan_type=all" \
                    -F "no_share_third_party=false" \
                    -F "allow_community_access=true" \
                    -F "submit_name=$submission_name" \
                    -F "file=@$1" \
                    "https://www.hybrid-analysis.com/api/v2/quick-
scan/file"`
```


POST
/quick-scan/file
submit a file for quick scan, you can check results in overview endpoint
Parameters
Try it out
Name
Description
scan_type * required
string
(formData)
Type of scan, please see /quick-scan/state to see available scanners
Available values : all, all_lookup, all_scan, lookup_ha, lookup_whitelists, lookup_whitelists_internal, lookup_virustotal, scan_crowdstrike_ml, scan_metadefender
all
file * required
file
(formData)
File to submit
Choose File
No file chosen
no_share_third_party
boolean
(formData)
When set to 'true', the sample is never shared with any third party. Default: true
--
allow_community_access
boolean
(formData)
When set to 'true', the sample will be available for the community. Default: true (Note: when 'no_share_third_party' is set to 'false', it won't be possible to set different value than 'true')
--
comment
string
(formData)
Optional comment text that may be associated with the submission/sample (Note: you can use #tags here)
comment - Optional comment text that may b
submit_name
string
(formData)
Optional 'submission name' field that will be used for file type detection and analysis
submit_name - Optional 'submission name' fi

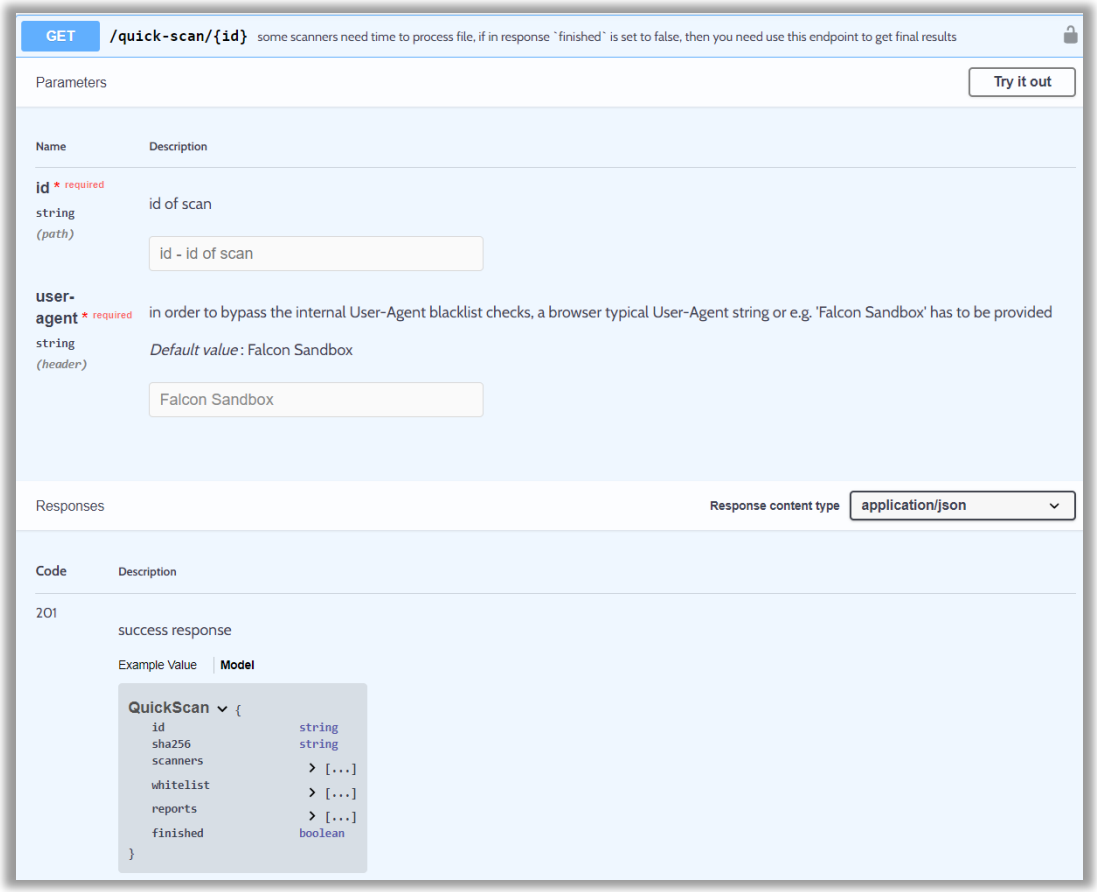

user-agent * required
string
(header)
in order to bypass the internal User-Agent blacklist checks, a browser typical User-Agent string or e.g. 'Falcon Sandbox' has to be provided
Default value : Falcon Sandbox
Falcon Sandbox
Responses
Response content type
application/json
Code
Description
201
success response
Example Value
Model
QuickScan {
id string
sha256 string
scanners [...]
whitelist [...]
reports [...]
finished boolean
}
Headers:
Name
Description
Type
Api-Limits
Api limits and current usage
string
Api-Version
Current API version
string
Webservice-Version
Current Webservice Version
string
Quick-Scan-Limits
Quick Scan limits and current usage
string

4. Daca fișierul încărcat nu este gol, parsăm răspunsul de la API și extragem id-ul submisiei, starea scanării („finished”) dar și valoarea SHA256 aferentă submisiei.

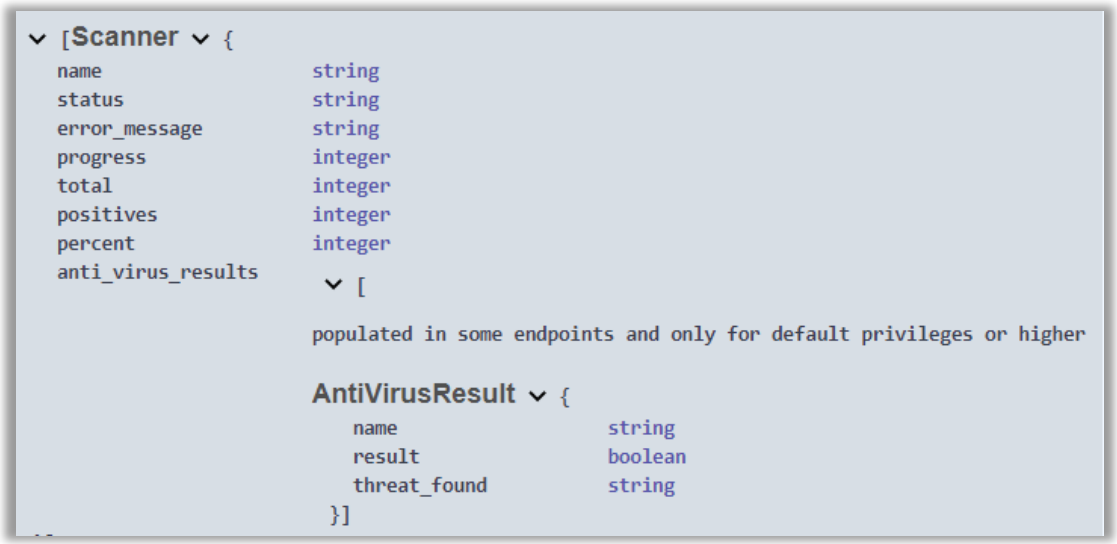
```
# If file is not empty.
if [[ $resp != *"An empty file is not allowed"* ]]
then
    id=`echo $resp | jq .id | sed 's/\"//g'`
    finished=`echo $resp | jq .finished`
    sha256=`echo $resp | jq .sha256 | sed 's/\"//g'`
```

5. Întrucât fișierele de dimensiuni mari necesită un timp de analiza mai mare, prin parametrul „finished” putem afla dacă analiza s-a terminat sau nu. În cazul în care analiza este încă în proces, trebuie să interogăm o altă rută de API pentru a verifica dacă s-a terminat submisia. (dacă da, obținem datele necesare clasificării fișierului)

```
# If file has not been fully analyzed yet, retry until it's done.
while [[ "$finished" == "false" ]]
do
    sleep 0.1
    resp=`curl -X GET --silent \
            -H "User-Agent: Falcon Sandbox" \
            -H "Accept: application/json" \
            -H "Api-Key: $API_KEY" \
            "https://www.hybrid-analysis.com/api/v2/quick-scan/$id"`
    finished=`echo $resp | jq .finished`
done
```

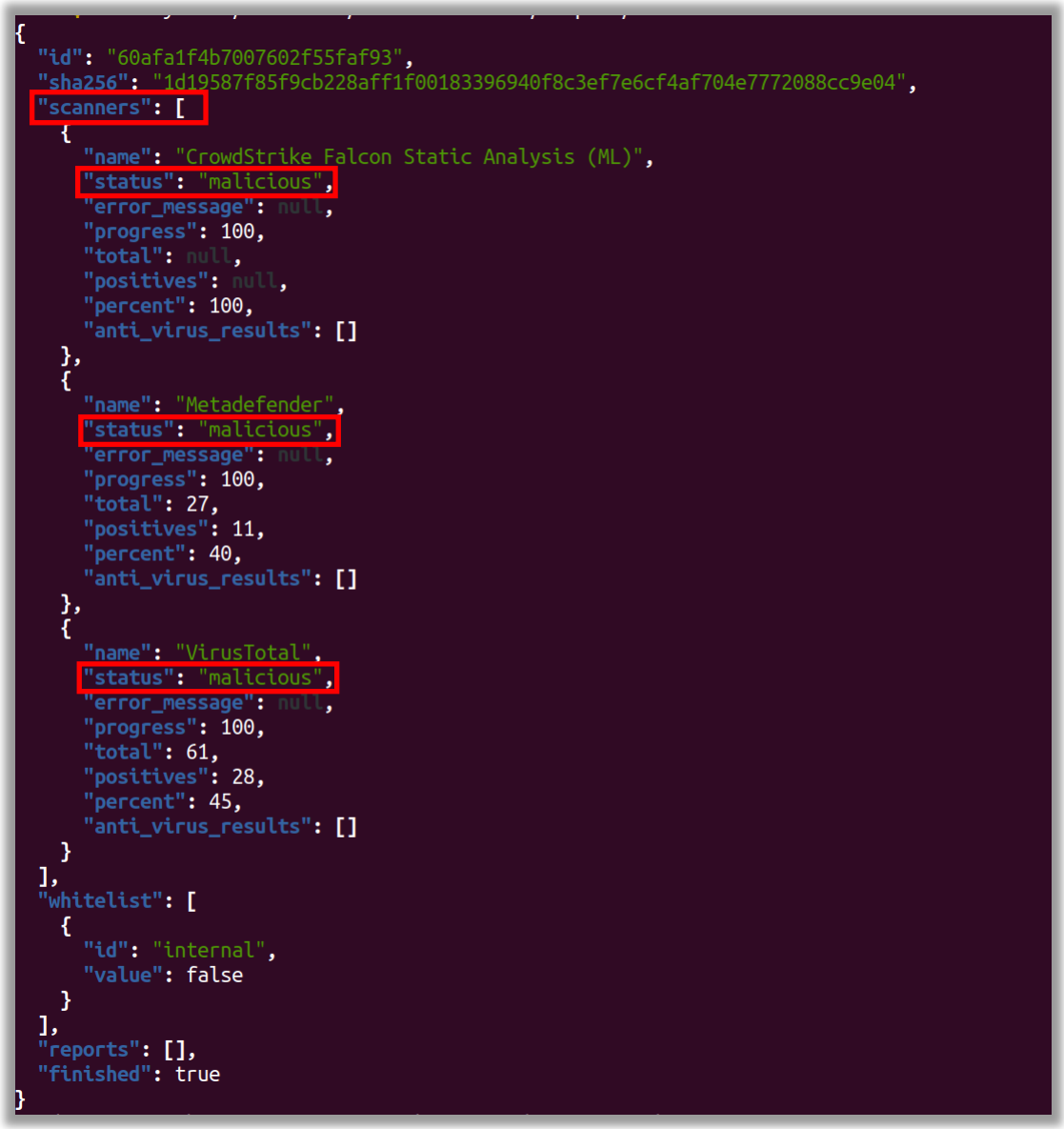


6. În acest punct, în variabila „resp” avem detaliile necesare pentru a putea concluziona dacă fișierul este malițios sau nu. Mai concret, în obiectul **JSON** putem verifica conținutul cheii „scanners” ce oferă o listă a vendorilor interogați cât și detalii despre verdictul oferit de aceștia. Pentru a lua decizia carantinării fișierului, am verificat dacă statusul de la cel puțin un vendor a fost „malicious” sau „suspicious”. Desigur că aceste criterii se pot customiza ușor în funcție de preferințele administratorului de sistem.




```
# See if any of the scanners responded with 'suspicious' or
'malicious' and isolate the file is so.
# To append to logfile, delete immutable attribute first, append then
add it back.
malicious=`echo $resp | jq '.scanners[].status |
contains("malicious")'`
suspicious=`echo $resp | jq '.scanners[].status |
contains("suspicious")'`
if [[ $malicious == *"true"* ]] || [[ $suspicious == *"true"* ]]
then

    fn="$QUARANTINE/$submission_name"
    chattr -i $LOGFILE
    echo -e "[+] New suspicious/malicious file: $1\n    Scan URL:
https://www.hybrid-analysis.com/sample/$sha256\n    Quarantined:
$fn\n    Timestamp: `date +'%R %d/%m/%Y'`" >> $LOGFILE
    chattr +i $LOGFILE
    isolate_file $1 "$fn"
fi
```



7. În imaginea de mai sus, carantinarea fișierului se realizează prin funcția „isolate_file” ce primește ca parametru calea absolută către fișierul ce trebuie carantinat cât și locația unde acesta va fi mutat (locația de carantinare). Înainte de a apela această funcție, am generat o alertă aferenta acestui fișier ce a fost detectat malițios și am salvat datele legate de numele fișierului, ora, locația fișierului carantinat cât și URL-ul expus de API pentru a vizualiza raportul de scanare. Procesul de carantinare constă în mutarea fișierului într-o locație accesibilă doar de administratorul sistemului cât și modificarea permisiunilor fișierului dar și setarea sa ca imutabil (nu poate fi alterat).

```
function isolate_file(){
    # If file is malicious or suspicious, move them to a temp folder, change owner to root and change permissions. Also, make it immutable.
    mv $1 $2
    chown root:root $2
    chmod 600 $2
    chattr +i $2
}
```

8. Singurul lucru care mai trebuie definit, pentru o bună funcționare a scriptului, sunt variabilele globale cu dețin valorile pentru cheia de API, hostname-ul sistemului, fișierul în care se vor salva alertele dar și folderul în care se vor carantina toate alertele.

```
# Define API Key to be used for HybridAnalysis, Logfile, Quarantine
directory as well as system hostname.
API_KEY="y8h1wkc8fd8afc85d3shu7foa91f4d63mu2hkeji0aa425dbdrevqgfzc878
cafd"
HOSTNAME=`hostname`
LOGFILE="/root/log.txt"
QUARANTINE="/root/quarantine"
```

5.5 Raportare si centralizare alerte

Prin raportare ne referim la procesul de transmitere a datelor de la client la server. În cazul nostru, pentru a transmite alertele de la toți clienții către server, s-a instalat pe client, prin scriptul „run.sh”, un serviciu care preia datele din fișierul de log și le transmite către server. Acest lucru implică și existența unui serviciu similar pe server care introduce datele primite în alt fișier de log. În final, aplicația web va parsa și centraliza datele din acest fișier.

Astfel, această secțiune este compusă din 3 sub-secțiuni, fiecare referindu-se la un mecanism distinct prin care se realizează raportarea si centralizarea alertelor:

- Transmitere client –> server
- Recepționare server
- Centralizare server

5.5.1 Transmitere client -> server

Acest lucru se realizează printr-un serviciu numit **reporter** care execută următorii pași:

1. Verifică dacă exista măcar o linie în fișierul de log. În realitate, o intrare în fișierul de log va avea mereu exact 4 linii întrucât acesta este formatul definit al alertelor. Dacă aceasta condiție este îndeplinită, extragem primele 4 linii (o alertă) din fișier și le formatăm corespunzător pentru a putea fi trimise ulterior către server folosind netcat.

```
# If the logfile is not empty, extract one alert entry (4 lines) and
consume them by sending to the server
if [[ `wc -l < $LOGFILE` -ne "0" ]]
then
    lines="`head -n4 $LOGFILE | sed 's/\[+\]/   /g' | sed 's/^/
/g'`"
    echo -e "[+] New alert from `hostname`@`hostname -I | sed 's/
//g'`:\n$lines" > $TMP_FILE
```

2. În continuare, (condiția este îndeplinita), se șterg primele 4 linii din fișier întrucât acestea au fost deja consumate (copiate în alt fișier). În final, se trimite fișierul temporar generat către server folosind utilitarul netcat după care se așteaptă o secunda până la următoarea verificare a fișierului de log pentru alte alerte. Ce se poate observa este că, pentru orice operație de modificare a conținutului fișierului de log, am folosit utilitarul „chattr” pentru a șterge și adăuga atributul de imutabilitate fișierului.

```
# Make sure to make the logfile not immutable before removing lines and
add the attribute after
    chattr -i $LOGFILE
    sed -i '1,4d' $LOGFILE
    chattr +i $LOGFILE

    # Send the tmp file to the server and cause a small delay.
    nc -w 3 $SERVER_IP $SERVER_PORT < $TMP_FILE
fi
sleep 1
```

5.5.2 Recepționare server

Recepționarea alertelor la nivel de server se face prin scriptul „server.sh” care preia toate datele primite pe portul 1337 și le plasează în fișierul de log. Acest lucru se realizează prin utilizarea comenzii nc ce deschide local portul 1337 și redirectează orice date primite pe acest port către fișierul de log. Se verifică, într-o bucla infinită, daca fișierul de log conține mai multe linii decât la ultima verificare (exista o alerta nouă), caz în care le afișează.

```
# Listen for any incoming message and place it in the log file
i=0
nc -klp $PORT >> $LOGFILE &


# Check if there is any new line in logfile and print it to stdout
while true
do
    new_i=`wc -l < $LOGFILE`
    if [[ $i -ne $new_i ]]
    then
        awk "NR<="$new_i"&&NR>"$i $LOGFILE
        i=$new_i
    fi
done
```

5.5.3 Centralizare server

Acest lucru se realizează printr-o aplicație web care are următorii pași de execuție:

1. Se pornește un server de Flask, pe mai multe thread-uri, pe portul 80 (adresa IP 0.0.0.0).

```
port = int(os.environ.get('PORT', 80))
if __name__ == '__main__':
    app.run(threaded=True, host='0.0.0.0', port=port)
```

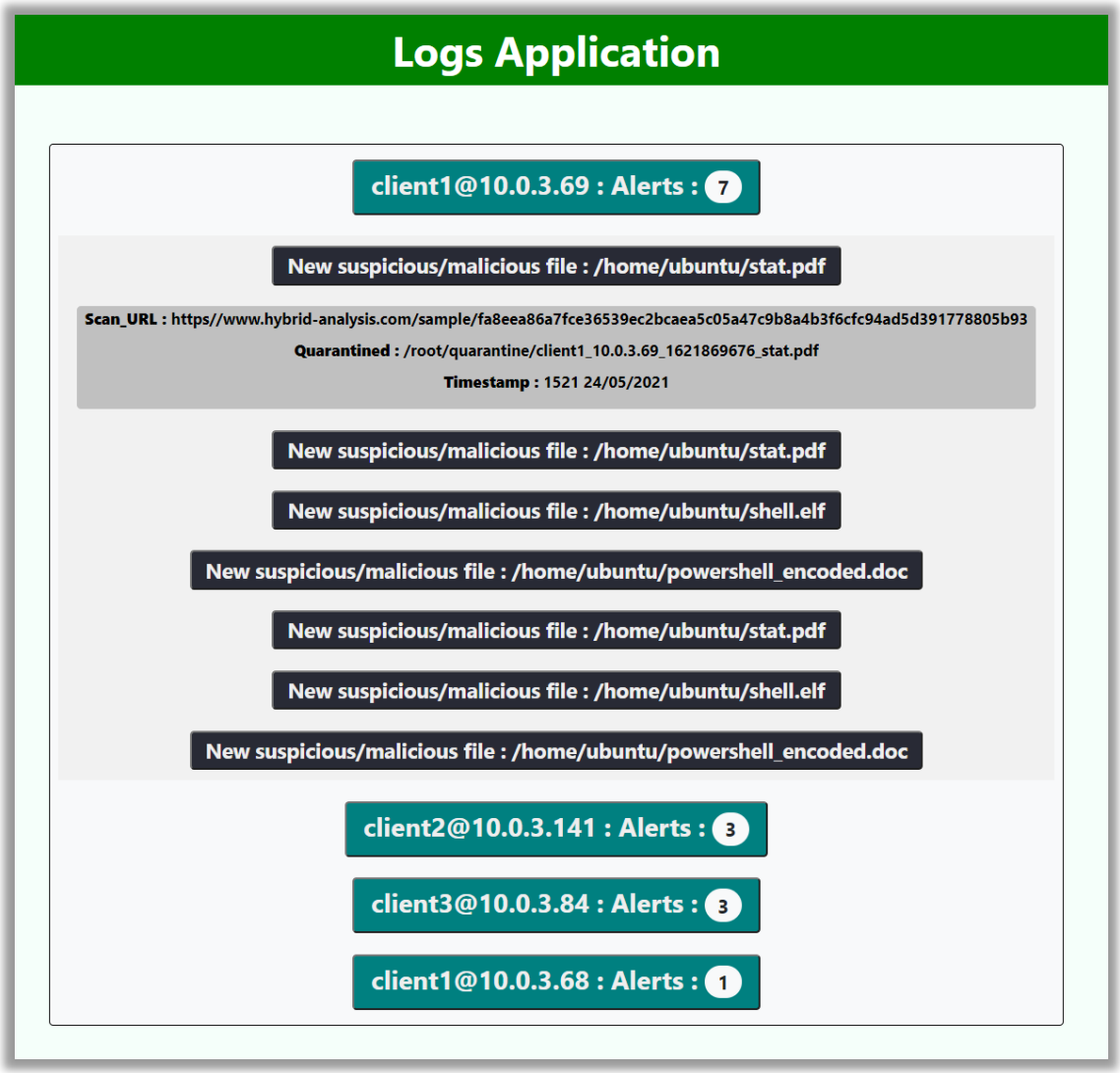
2. Serverul expune endpoint-ul **/data** ce întoarce conținutul fișierului de log in format text.

```
@app.route('/data',methods = ['GET'])
def readData():
    f = open("static/log.txt", "r")
    data = str(f.read())
    res = Response(data,mimetype="text/xml")
    return res
```

3. Se randează in pagina web fișierul **index.html** din folderul **static**

```
@app.route('/')
def create():
    return render_template("index.html")
```

4. Prin intermediul limbajului JavaScript, ce este interpretat de browser, se apelează endpoint-ul /data și se parsează datele întoarse pentru a fi expuse în pagina web sub formă de listă. Lista finala conține o intrare pentru fiecare pereche hostname + IP unde se specifică câte alerte s-au detectat pe acel host. Fiecare element din această listă poate fi expandat, rezultând în o listă cu alertele propriu-zise. Fiecare alertă conține informații despre fișierul care a fost detectat ca posibil malițios. Fiecare alerta, la rândul său, poate fi expandată într-un câmp text unde sunt specificate detaliile detecției: ora detecției, scan URL-ul unde se poate vizualiza scanarea (din cadrul platformei HybridAnalysis) și locația unde a fost carantinat fișierul. Datele din pagină sunt actualizate la fiecare 5 minute reinterogând endpoint-ul oferit de aplicație. Re-actualizarea se poate face și manual prin click-ul în zona neagră din partea superioară a paginii.



```
 <body>
    <h1
      id="title"
      onClick="reload()"
      class="badge"
      style="font-size: 40px; background-color: green; text-align: center;"
    >
      Logs Application
    </h1>
    <br/><br/>

    <div id="output" class="container" style="align-content: center"></div>
    <script>
      var fileDisplayArea = document.getElementById("output");
      var prevData = "";
      var logsArr = [];
      const setTemplate = (data) => {
        logsArr = [];
        if (prevData == "") {
          transformData(data);
        } else if (data == prevData) {
        } else if (data != prevData) {
          transformData(data);
        }
      };

      const transformData = (data) => {
        var logs = data.split("[+]");
        if (logs != null || logs != [] || logs != "") {
          for (let i = 1; i < logs.length; i++) {
            let cd = {
              client: "",
              noOfAlerts: 1,
              suspicious_malicious_file: [],
              Scan_URL: [],
              Quarantined: [],
              Timestamp: [],
            };
            let dividedLogs = logs[i].split(" New alert from
")[1].split("\n");
            let alreadYAlerted = false;

            for (let y = 0; y < logsArr.length; y++) {
              if (logsArr[y]["client"] == dividedLogs[0].split(":")[0]) {
                logsArr[y]["noOfAlerts"] = logsArr[y]["noOfAlerts"] + 1;
                logsArr[y]["suspicious_malicious_file"].push(
                  dividedLogs[1].split(":")[1]
                );
                logsArr[y]["Scan_URL"].push(
                  dividedLogs[2]
                    .split(":")[1]
                    .concat(dividedLogs[2].split(":")[2])
                );

logsArr[y]["Quarantined"].push(dividedLogs[3].split(":")[1]);
                logsArr[y]["Timestamp"].push(
                  dividedLogs[4]
                    .split(":")[1]
                    .concat(dividedLogs[4].split(":")[2])
```

```
                );
                alreadYAlerted = true;
                break;
              }
            }

            if (alreadYAlerted == false) {
              cd["client"] = dividedLogs[0].split(":")[0];
              cd["suspicious_malicious_file"].push(
                dividedLogs[1].split(":")[1]
              );
              cd["Scan_URL"].push(
                dividedLogs[2]
                  .split(":")[1]
                  .concat(dividedLogs[2].split(":")[2])
              );
              cd["Quarantined"].push(dividedLogs[3].split(":")[1]);
              cd["Timestamp"].push(
                dividedLogs[4]
                  .split(":")[1]
                  .concat(dividedLogs[4].split(":")[2])
              );
              logsArr.push(cd);
            }
          }
        }
        var clientData = "";
        for (let i = 0; i < logsArr.length; i++) {
          clientData += `<button type="button" id="btn" style="background-
color: teal; color:#eee;"  class="badge">${logsArr[i]["client"]} : Alerts :
<span class="badge rounded-pill bg-light text-
dark">${logsArr[i]["noOfAlerts"]}</span></button>
                        <br/> <div id="outerDiv" class="content">`;
          for (
            let j = 0;
            j < logsArr[i]["suspicious_malicious_file"].length;
            j++
          ) {
            clientData += `<button type="button" id='btn1' style="font-
size:20px; background-color:#282A35; color:#eee;" class="badge">New
suspicious/malicious file :
${logsArr[i]["suspicious_malicious_file"][j]}</button>
                                                    <br/><div
id='innerDiv' class="badge" style="display: none; color:#000000;
background-color:silver">
                                                    <p style="font-
size:15px;"><b>Scan_URL : </b> ${logsArr[i]["Scan_URL"][j]}</p>
                                                    <p style="font-
size:15px;"><b>Quarantined : </b>  ${logsArr[i]["Quarantined"][j]}</p>
                                                    <p style="font-
size:15px;"><b>Timestamp : </b>  ${logsArr[i]["Timestamp"][j]}</p>
                                                    </div>
                                                    `;
          }
          clientData += `</div>`;
        }
        fileDisplayArea.innerHTML =
          '<div class="badge bg-light text-dark" style="border:1px solid
black;">' +
          clientData +
          "</div>";
```

```
      var coll = document.getElementsByClassName("badge");
      for (var i = 0; i < coll.length; i++) {
        coll[i].addEventListener("click", function () {
          this.classList.toggle("active");
          var content = this.nextElementSibling;
          content = content.nextElementSibling;
          // content = content.nextElementSibling;
          if (content.style.display === "block") {
            content.style.display = "none";
          } else {
            content.style.display = "block";
          }
        });
      }
      prevData = data;
    };

    function reload() {
      window.location.reload();
    }

    function getLogData() {
      setTimeout(function () {
        fetch("/data")
          .then((response) => response.text())
          .then((data) => {
            setTemplate(data);
          });
        getLogData();
      }, 300000);
    }

    fetch("/data")
      .then((response) => response.text())
      .then((data) => {
        setTemplate(data);
        getLogData();
      });
  </script>
 </body>
</html>
```

6. STUDIU DE CAZ / EVALUAREA REZULTATELOR

Pentru a putea studia și evalua performanțele soluției implementate, am simulat un mediu IoT cu 4 clienți și 1 server. Fiecare din aceste componente reprezintă un container de Linux construit prin intermediul tehnologiei **LinuX Containers** (LXC), ce este un sistem de operare la nivel de virtualizare pentru rularea mai multor izolate sistemele Linux (containere) pe un singur control gazdă (LXC gazdă).

Astfel, primul pas este de obținere a unei imagini virtuale de Linux, in cazul nostru am ales Ubuntu 20.04, pe care sa o instalam folosind un software precum Virtual Box sau VMware. Odată ce avem o mașina de Linux la îndemâna, am instalat pachetul lxc folosind următoarea comanda:

```
sudo apt-get install lxc
```

Pentru a crea 5 containere de Linux, putem folosi următoarea comanda:

```
sudo lxc-create -t download -n nume_container
```

Aceasta comanda ne va cere sa alegem distributia de Linux care sa ruleze pe container. Pentru mediul nostru de test am mers pe *ubuntu xenial amd64*.

Pentru a porni un container si a ne atașa la el trebuie sa rulam următoarele comenzi:

```
sudo lxc-start nume_container
sudo lxc-attach nume_container
```

7 CONCLUZII

De completat

8 BIBLIOGRAFIE

De completat