UNIVERSITATEA POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMĂ

Embedded Devices Malware Detection, Prevention & Centralization

Andrei Grigoras

**Coordonator științific:**

Prof. dr. ing. Cornel Popescu

BUCUREŞTI

2021

**CUPRINS**

[Sinopsis 1](#_Toc73017450)

[Abstract 1](#_Toc73017451)

[1 Introducere 2](#_Toc73017452)

[1.1 Context 2](#_Toc73017453)

[1.2 Problema 2](#_Toc73017454)

[1.3 Obiective 2](#_Toc73017455)

[1.4 Structura lucrării 2](#_Toc73017456)

[2 Analiza și specificarea cerințelor 3](#_Toc73017457)

[3 Studiu de piață / Abordări existente 4](#_Toc73017458)

[4 Soluția propusă 5](#_Toc73017459)

[5 Detalii de implementare 6](#_Toc73017460)

[5.1 Tehnologii client 6](#_Toc73017461)

[5.1.1 Bash – limbaj de programare 6](#_Toc73017462)

[5.1.2 Netcat – utilitar pentru transmiterea datelor către client 6](#_Toc73017463)

[5.1.3 Systemd – manager pentru crearea si managementul serviciilor pe sistem 6](#_Toc73017464)

[5.1.4 HybridAnalysis 6](#_Toc73017465)

[5.2 Tehnologii server 7](#_Toc73017466)

[5.2.1 Python 7](#_Toc73017467)

[5.2.2 Flask/Django 7](#_Toc73017468)

[5.3 Instalare & Configurare 8](#_Toc73017469)

[5.3.1 Server 8](#_Toc73017470)

[5.3.2 Client 9](#_Toc73017471)

[5.4 Detecție & Carantinare fișiere malițioase 10](#_Toc73017472)

[6 Studiu de caz / Evaluarea rezultatelor 11](#_Toc73017473)

[7 Concluzii 12](#_Toc73017474)

[8 Bibliografie 13](#_Toc73017475)

[9 Anexe 14](#_Toc73017476)

# Sinopsis

Scopul acestui proiect a fost de a implementa un mecanism rapid, portabil, ușor de instalat, si 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).

In urma unei investigări pentru a găsi software echivalent (gratis si disponibil pentru orice sistem Unix), am observat absenta unei astfel de soluții. Din acest motiv, proiectul se adresează dispozitivelor embedded din categoria „Internet of Things”, si 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 enters a **Unix** system via Internet (download) or external devices (USB, CD). Therefore, I looked for creating a solution using **Bash** scripting language, which is the native language used my **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 of 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 a 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**.

# Introducere

## 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, determina daca fișierul este sau nu malițios. Pe de alta parte, proiectul implementat diferă de o soluție clasica de antivirus prin piața ce o urmărește, aceea a dispozitivelor embedded (precum cele din IoT) ce oferă resurse limitate la nivelul sistemului de operare, făcând astfel imposibila instalarea unei soluții precum Avira/Bitdefender/Avast. (ce sunt specifice sistemului de operare Windows).

Deși exista 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.

## 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 piața exista astfel de soluții (mult mai complexe) denumite „Host based Intrusion Detection System” sau „Endpoint detection and response” dar care necesita achiziționarea unui abonament/licențe.

## Obiective

Obiectul principal al proiectului este de a reuși cu succes să detectăm și eradicăm fișierele malițioase noi apărute pe mai multe sistem Unix cat si centralizarea grafica 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 daca un fișier este malițios sau nu.

## Structura lucrării

In secțiunea următoarea, vom analiza si specifica cerințele ce au fost luate in considerare in implementarea proiectului cat si modalitățile/abordarea care a garanta îndeplinirea lor.

După, vom face o analiza a pieței, soluțiile existente, cat si factorii prin care soluția propusa se evidențiază in acest domeniu. Vom porni de la contextul atacurilor de tip malware si vom sfârși prin a înțelege utilitatea soluției noastre in industria IoT.

In secțiunile 4 si 5 vom discuta despre soluția propusa (arhitectura, diagrama, cazuri de utilizare) cat si detalii specifice precum bucăți de cod, funcționalități si limitări.

In secțiune 6, vom evalua performantele prin crearea unui mediu de test si simulând un utilizator. Astfel vom putea deduce timpul de răspuns, eficienta dar si corectitudinea datelor.

Ultimele secțiuni sunt adresate unor concluzii, bibliografii folosite cat si anexe utile in înțelegerea soluției.

# Analiza și specificarea cerințelor

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

* scalabilitate
* disponibilitate
* viteza cat mai ridicata a procesării
* costuri cat mai reduse
* resurse de calcul cat mai scăzute (evitare overhead)
* ușor de instalat si configurat

Abordarea propusa se pliază ușor pe aceste criterii întrucât, folosind un limbaj nativ sistemului de operare ce suporta si multi-threading (Bash), atât criteriul de scalabilitate cat si cel de viteza sunt ușor de îndeplinit. Pentru a facilitata instalarea ușoara a produsului, trebuie realizat un script care, in urma execuției, va realiza toți pașii necesari pregătirii si instalării de software. De asemenea, întrucât produsul final este open-source, am adăugat comentarii pentru fiecare bloc de cod, dar si indicații in cazul in 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 si folosirii este zero. De asemenea, deoarece procesarea se face prin intermediul soluțiilor open-source ce rulează in **cloud**, resursele necesare rulării sunt minime si sunt strict dependente de calitatea conexiunii la Internet la rețea. (pentru interacțiunea cu API-ul si transmiterea alertelor din rețea către server)

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

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

# Studiu de piață / Abordări existente

Pentru a putea implementa o soluție cat mai buna si care sa poată fi folosita înafara unui „Proof of Concept”, trebuie sa analizam piața aferenta zonei pe care produsul o adresează, in cazul nostru fiind cea alcătuita din software ce ajuta in detectarea de malware.

In analiza pieței, am consultat tipurile de soluții ce pot concura cu proiectul nostru, acestea împărțind-se in mai multe categorii:

* HIDS (Host Based Intrusion Prevention Systems)
* EDR (Endpoint Detection and Response)
* Tool-uri open-source

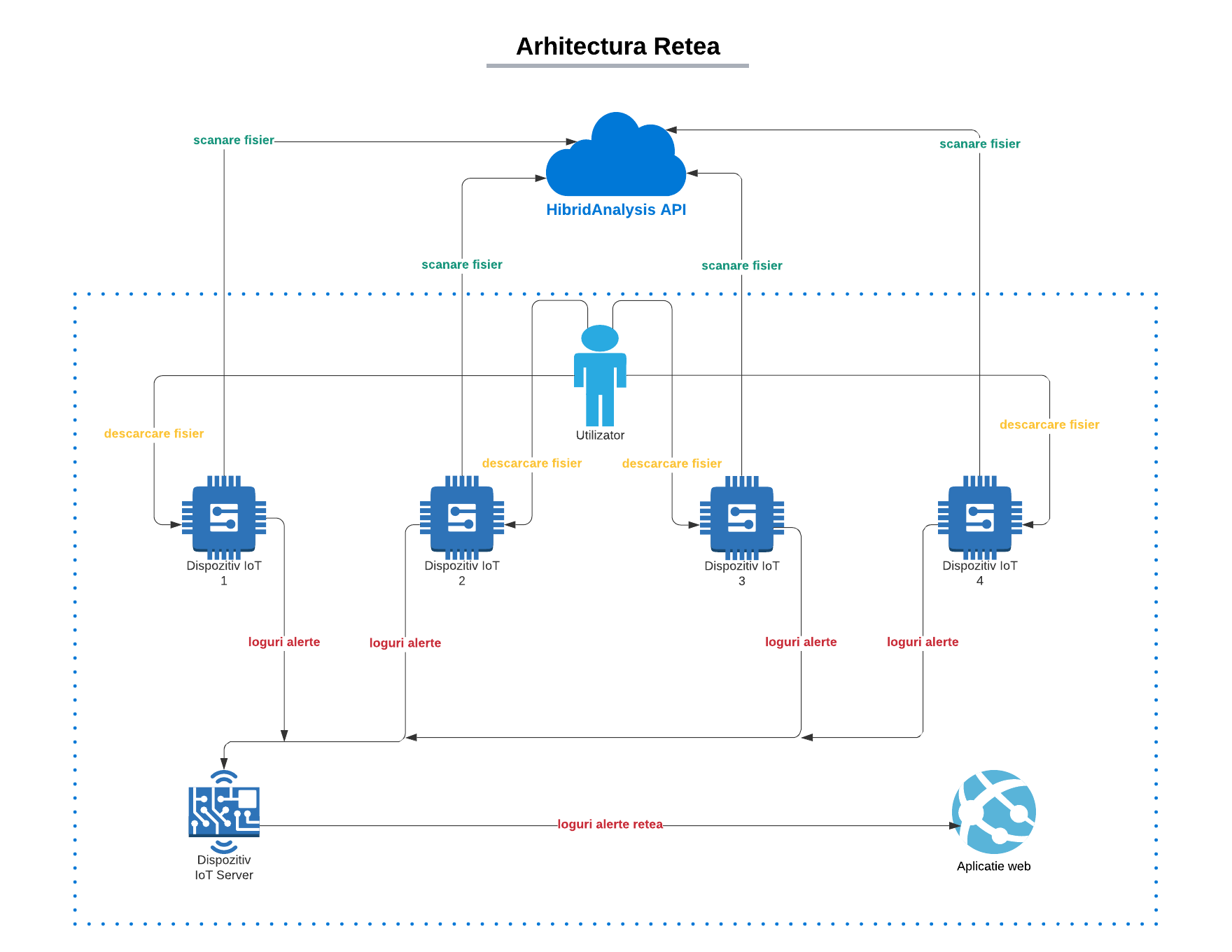
Categoriile de HIDS si EDR sunt adresate mediilor corporate întrucât sunt construite pentru a scala pentru mii de sisteme. De asemenea, acestea încorporează o sumedenie de funcționalități de prevenire si detecție a incidentelor de securitate pe un host (nu doar detecție de malware). Problema acestor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

Pe de alta parte, exista câteva soluții gratis, open-source care pot scana un sistem si detecta diverse tipuri de malware dar si alte tipuri de probleme precum configurații greșite, software cu versiuni ne-update etc. Una din cele mai cunoscute astfel de soluții este Wazuh/OSSEC cu care, personal, am interacționat si am observat funcționalitățile/limitările. Deși aceasta soluție este net superioara celei propuse de noi, ea nu se adresează tuturor sistemelor Unix si depinde de existenta unui „package manager” sau a unei soluții precum **Docker** la nivelul sistemului de operare (pentru a seta mediul de rulare al software-ului). In cazul dispozitivelor embedded ce au un kernel personalizat, aceste dependențe nu pot fi obținute.

Ținând cont de aceste aspecte ale pieței, am considerat ca proiectul va acoperi un mare gap in industrie, adresându-se unei industrii nișate dispozitivelor embedded. Acestea, in multe circumstanțe, doresc sa fie construite cu resurse cat mai puține dar sa prezinte anumite masuri de siguranța la nivelul securității sistemului. Putem concluzia astfel ca, piața curenta 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.

# Soluția propusă

Soluția propusa se adresează unei arhitecturi cu mai mulți clienți ce sunt reprezentate de dispozitive IoT si de un server care este tot un dispozitiv IoT (cel mai practic ar fi o plăcută Raspberry Pi). Software-ul ce rulează pe clienți va încarcă periodic fișierele noi apărute in sistem pe platforma HybridAnalysis folosind API-ul expus de acesta. Daca fișierul este detectat ca fiind malițios, se adaugă intr-un fișier de log care, la rândul sau, este transmis către server la orice noua intrare. Acesta, serverul, are rol in colectarea log-urilor de la toți clienții (alerte de fișiere potențial malițioase) si de centralizarea lor printr-o aplicație web ce va afișa informații precum: IP si hostname client, număr alerte in ultimele 24 ore cat si un sumar al acestor fișiere. Astfel, arhitectura poate fi reprezentata sub forma următoarei scheme:



Soluția propusa nu are nicio dependință sau componenta hardware întrucât este reprezentata de o multitudine de servicii (software) care trebuie instalate atât pe clienți cat si pe un dispozitiv ce va servi ca si server. Corectitudinea soluției este strâns data de 2 factori:

* corectitudinea detecției soluției oferite de HybridAnalysis prin API-ul expus
* transmiterea corecta si in timp a datelor de la client la API si la server (sa nu fie corupte sau sa nu ajungă deloc)

# Detalii de implementare

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

## Tehnologii, limbaje si utilitare client

In implementarea clientului, am folosit următoarele utilitare, limbaje si tehnologii:

### 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.

### 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.

### 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 paralela, utiliza socket-ul 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.

### 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.

### chattr – utilitar folosit pentru schimbarea atributelor unui fișier.

### curl – utilitar folosit pentru transferarea datelor de la server folosind protocoale precum HTTP, HTTPS, FTP etc**.**

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

### date – utilitar folosind pentru afișarea si setarea datei sistemului

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

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

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

### mv – utilitar folosit pentru mutarea sau redenumirea fișierelor.

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

### chmod – utilitar folosit pentru schimbarea permisiunilor unui fișier

### mkdir – utilitar folosit pentru crearea de noi directoare

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

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

### sleep – utilitar folosit pentru întârzierea sistemului

### echo - utilitar folosit pentru afișarea unei linii de text

### find – utilitar folosit in căutarea fișierelor într-o ierarhie de directoare

### rm – utilitar folosit pentru ștergerea de fișiere sau directoare

### cat – utilitar folosit in concatenarea si afișarea fișierelor

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

### head – utilitar folosit in afișarea primei partid intr-un fișier

### exit – utilitar folosit pentru terminarea unui proces

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

### apt – utilitar folosit pentru managementul pachetelor de pe sistem

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

### bash – interpretor pentru limbajul de programare Bash

## 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 construita folosind următoarele tehnologii:

### **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.

### **Flask/Django**

..

..

...

. ...

...

...

...

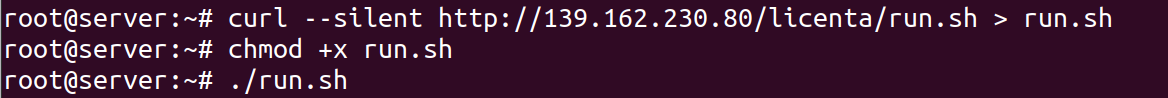
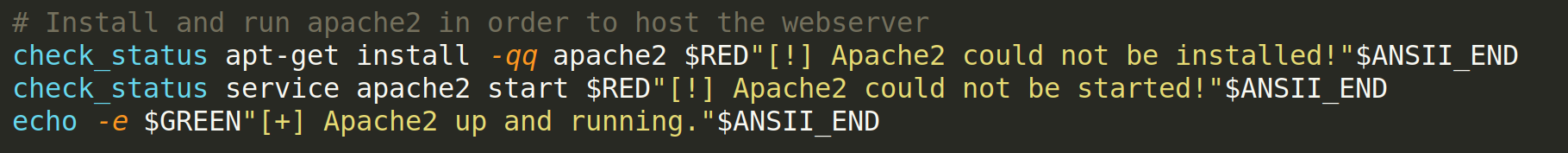
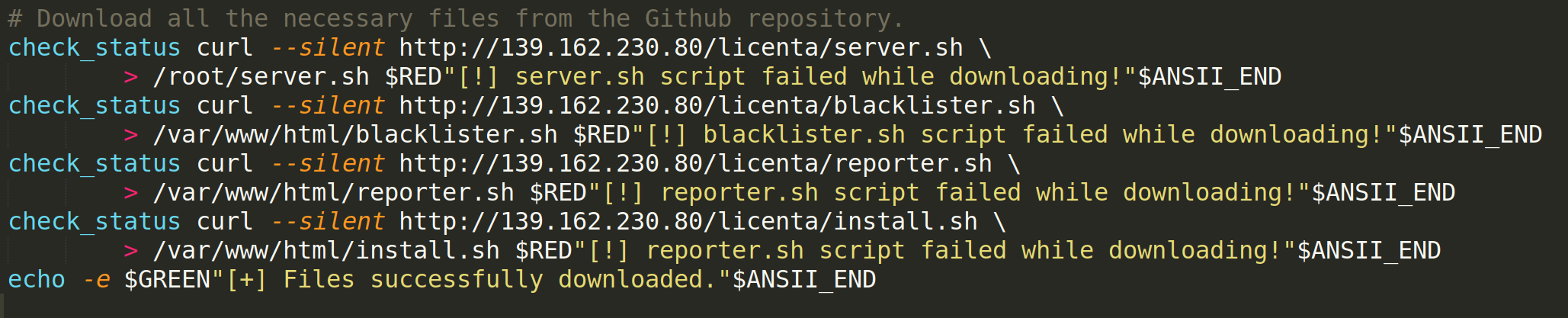
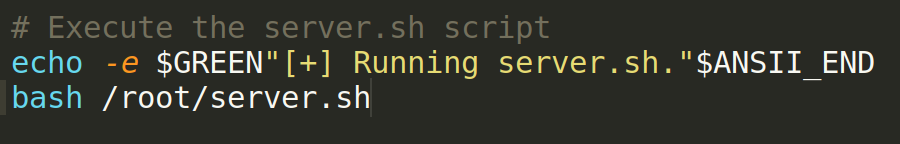
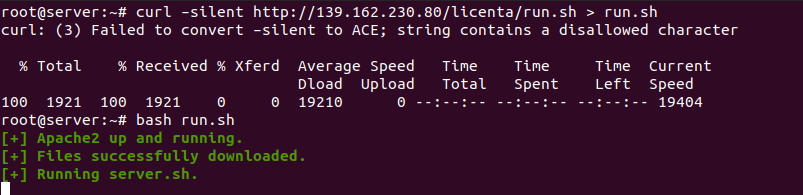
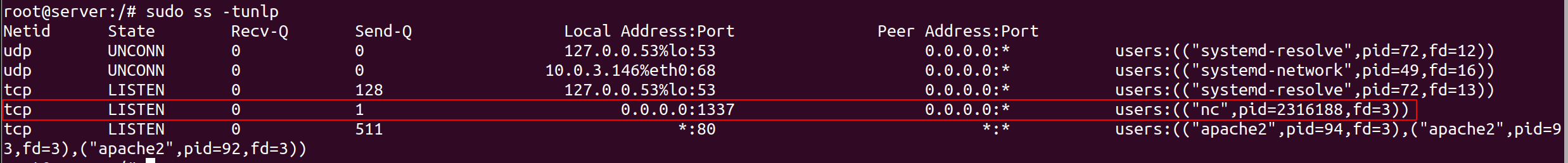
In continuare, vom aborda următoarele categorii din punct de vedere al implementării atât pentru client cat si pentru server:

* Instalare & Configurare
* Detecție & Carantinare fișiere malițioase
* Raportarea & Centralizarea alertelor

## Instalare & Configurare

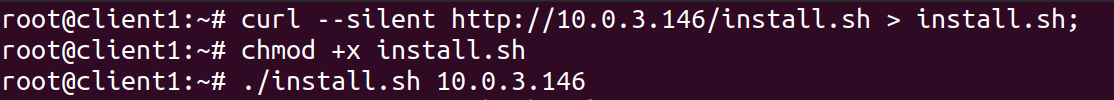
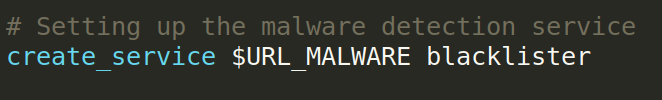
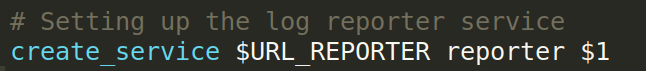
### **Server**

Primul pas pentru instalarea completa a soluției este cea introducere a fișierelor necesare pe server. Acest lucru se poate realiza prin intermediul unei descărcări directe de pe internet cat si prin inserarea unui dispozitiv periferic precum USB, CD, DVD, sau Hard Disk. Totuși, abordarea aleasa ,in scop demonstrativ, a fost hostarea fișierelor triviale instalării pe un server in cloud-ul **Linode** 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 ii urmeze cat si cei pe care scriptul ii executa sunt următorii:

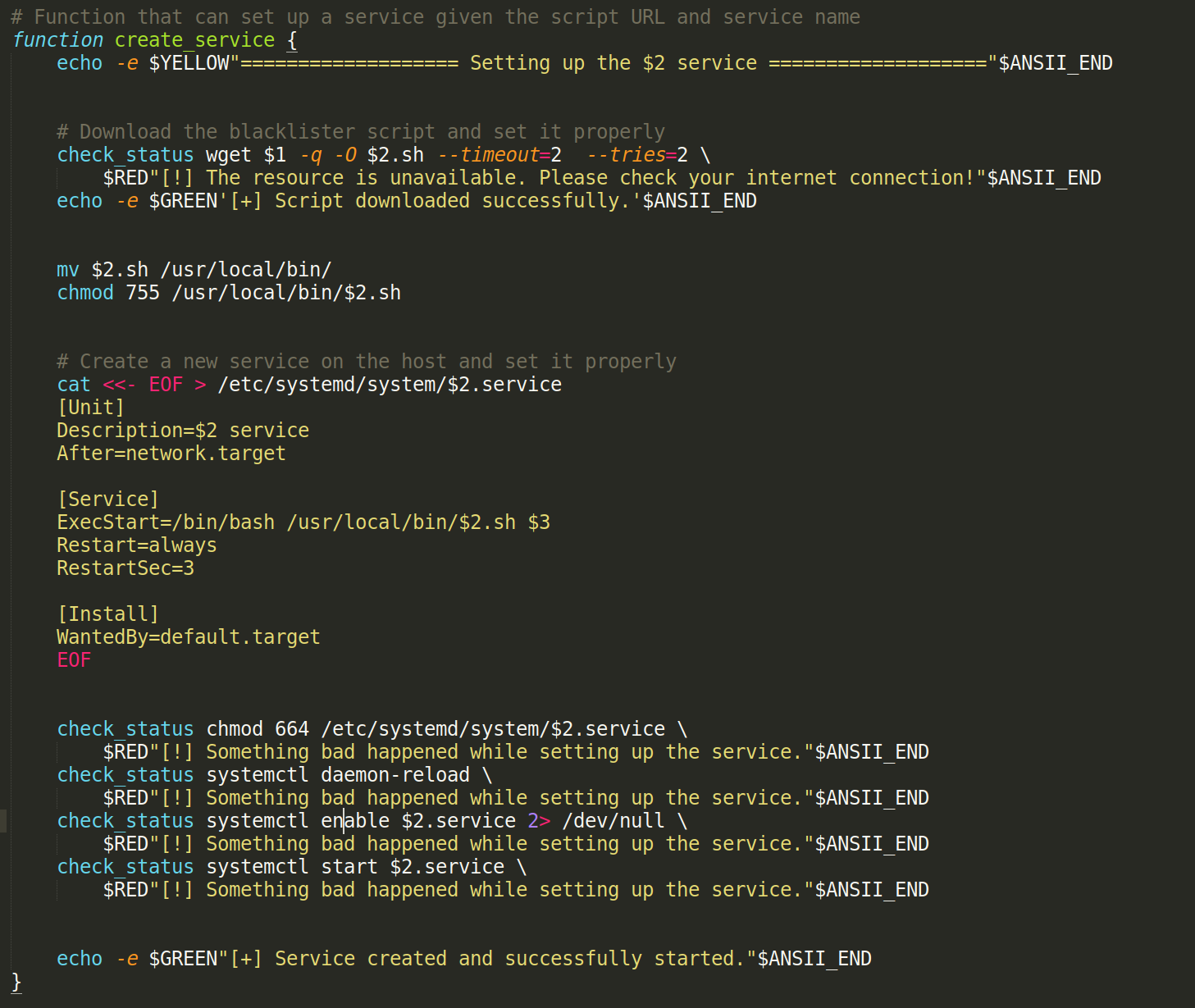
1. Utilizatorul descarcă scriptul „run.sh”, îl marchează ca executabil (+x) si il executa.
2. Scriptul instalează serviciul **apache2** pentru hostarea fișierelor necesare clienților.
3. Scriptul descarcă toate fișierele necesare instalării soluției atât pentru server cat si pentru client si le plasează in directorul specific serviciului Apache (/var/www/html).
4. Scriptul rulează un alt script numit „server.sh” care deschide un port local prin care se vor primi alertele de la clienții din rețea prin intermediul mesajelor text transmise cu netcat.
5. Daca toți pașii s-au executat cu succes, pe server, ar trebui sa avem următorul output in urma rulării comenzilor de descărcare si rulare a scriptului „run.sh”.
6. Verificam ca pe server rulează netcat pe portul 1337 folosind utilitarul „ss” sau ”netstat”.

### **Client**

Având partea de server deja configurata corespunzător, pe partea de client trebuie sa descărcam, de pe server, scriptul „install.sh” pe care îl vom executa dându-i ca unic parametru adresa IP a server-ului. Astfel, pașii pe care utilizatorul trebuie sa ii urmeze cat si cei pe care scriptul ii executa sunt următorii:

1. Utilizatorul descarcă scriptul „install.sh” de pe server si îl executa cu adresa IP a serverului.
2. Scriptul descarcă si instalează serviciul responsabil de detecția fișierelor malițioase pe sistem.
3. Scriptul descarcă si instalează serviciul responsabil de raportarea alertelor de pe sistem.

Pentru instalarea acestor servicii, s-a folosit următoarea funcție implementata in Bash ce generează un nou serviciu prin adăugarea unei intrări noi in folderul /etc/systemd. Aceasta funcție primește 2 parametri prin care se specifica numele noului serviciu cat si URL-ul de unde poate descarcă scriptul ce trebuie rulat de serviciu.

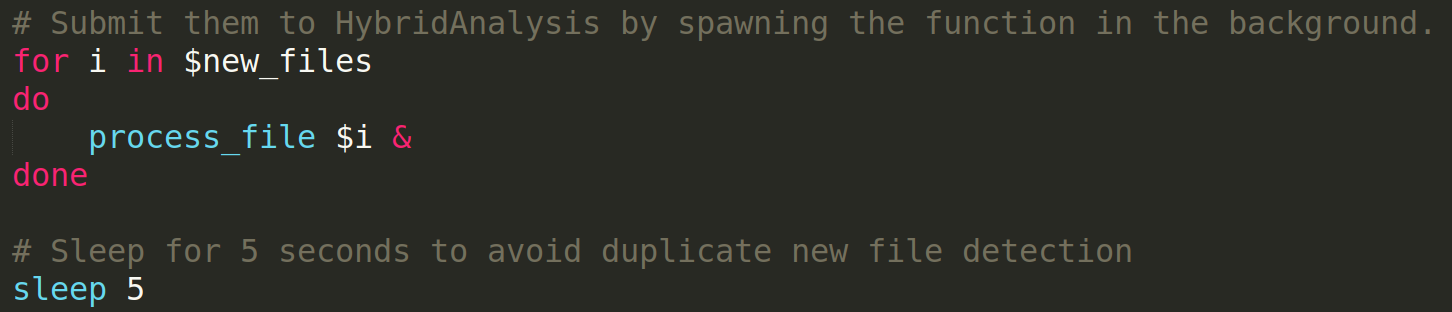
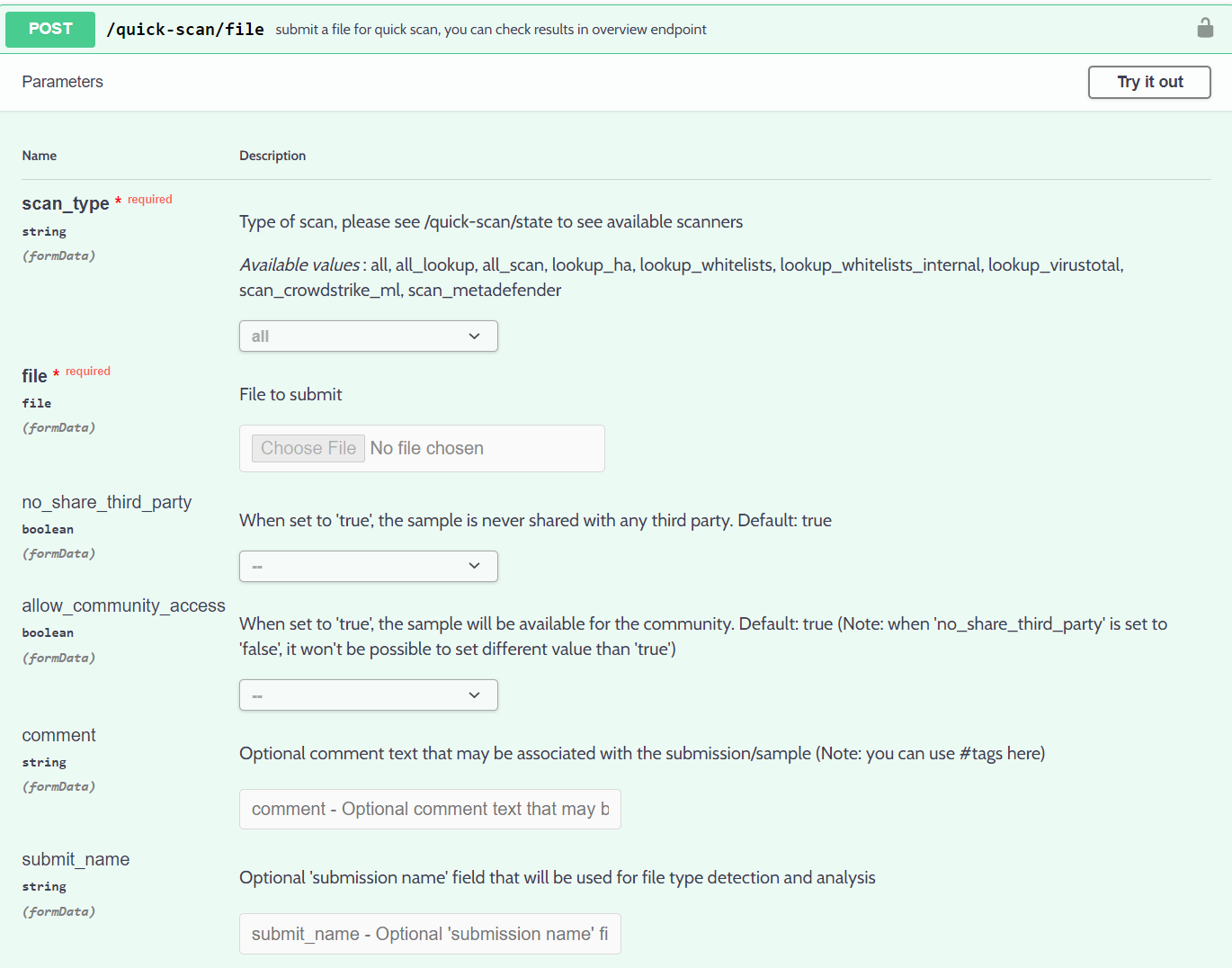


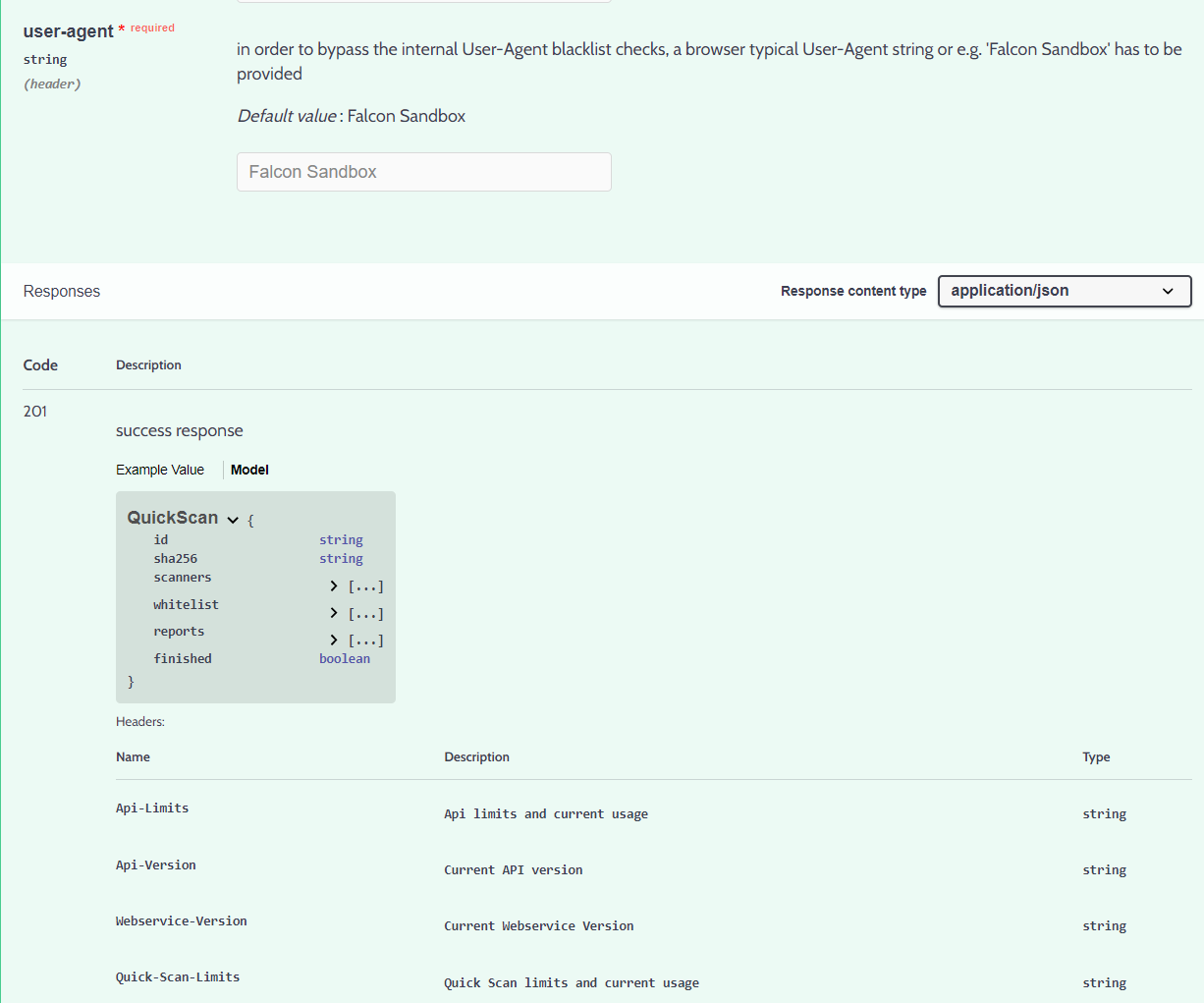
## Detecție & Carantinare fișiere malițioase

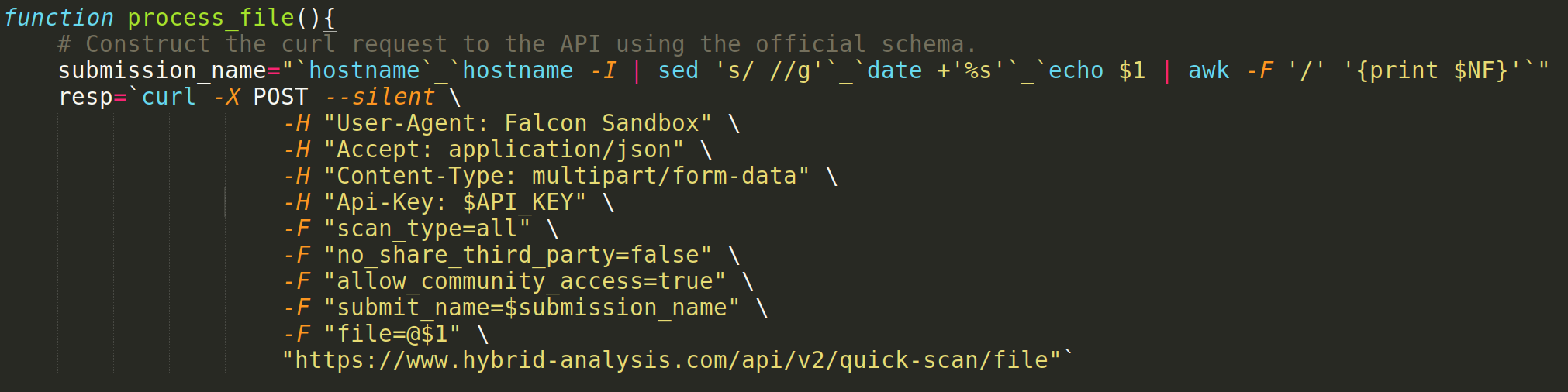
In aceasta secțiune vom urmări îndeaproape detaliile de implementare ce realizează atât detecția fișierelor malițioase noi apărute pe sistem cat si carantinarea acestora pentru a împiedica user-ul din a le deschide sau executa. Acest proces se executa doar la nivelul clientului si consta din serviciul „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:

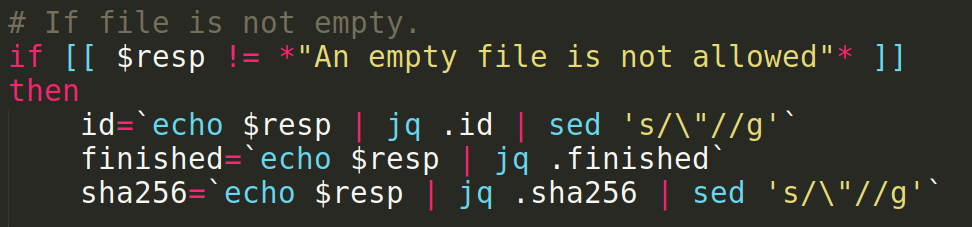
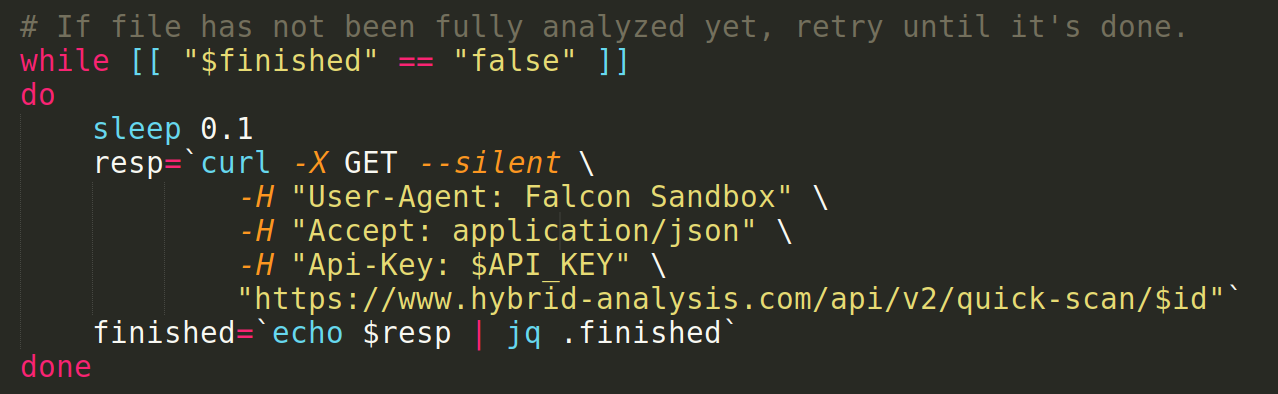
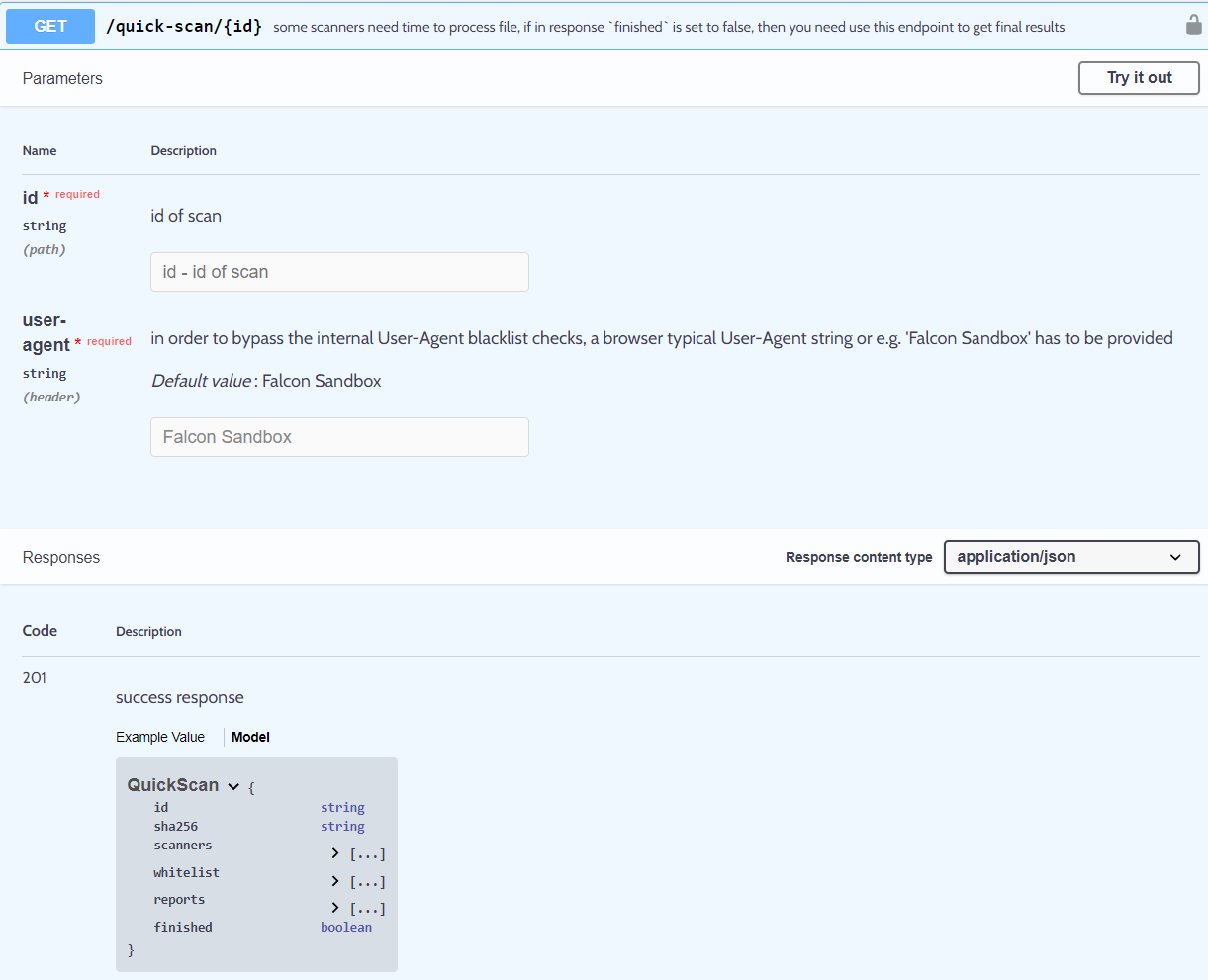
1. Se caută folosind comanda **find** toate fișierele care au fost modificate sau accesate in ultimele 6 secunde (0.1 minute).

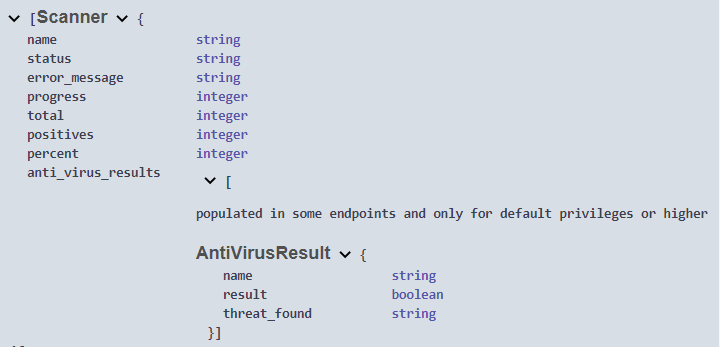


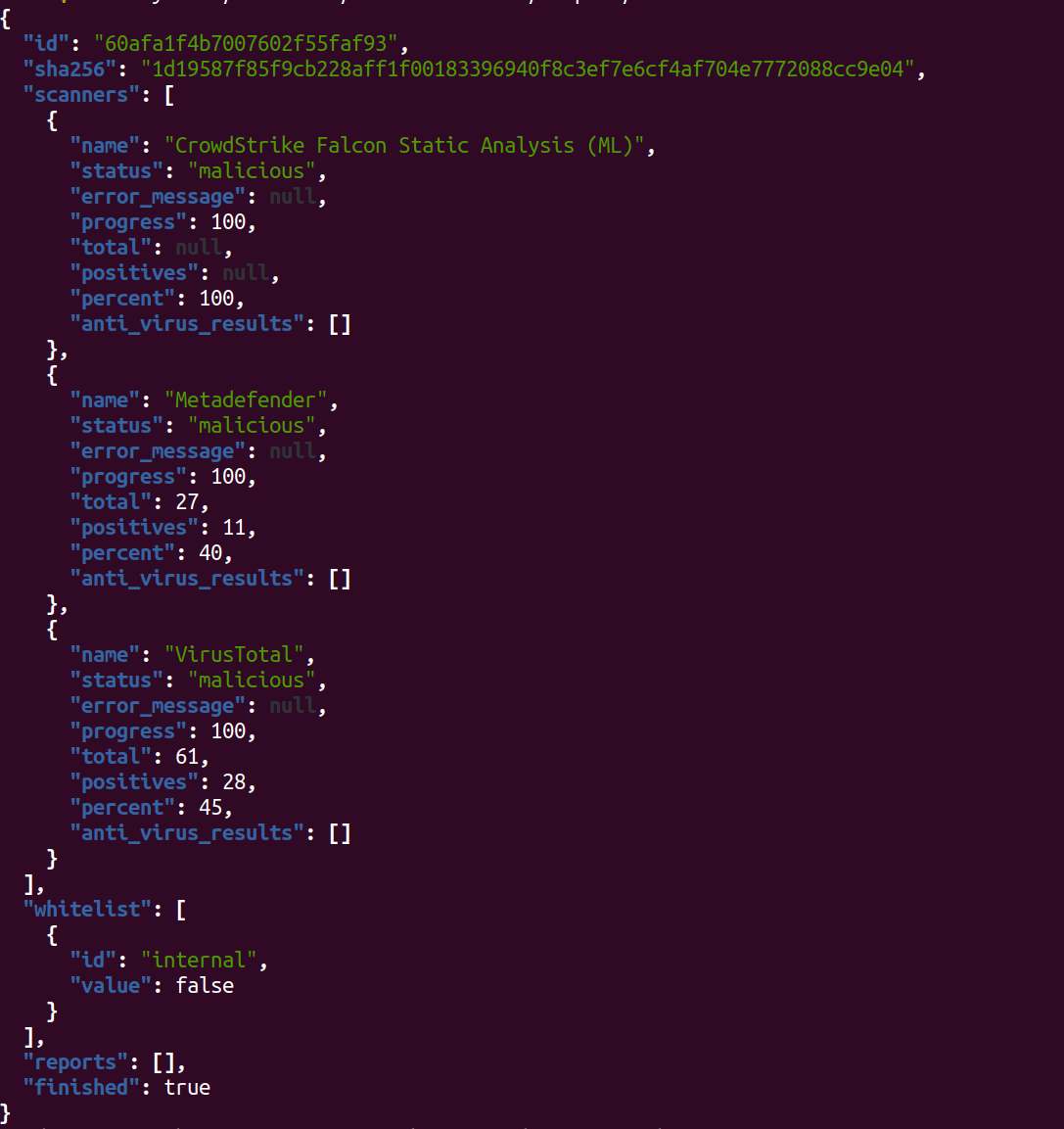
1. Pentru fiecare fișier obținut la subpunctul 1, se apelează funcția **process\_file** având ca parametru calea absoluta către fișier. La finalul instrucțiunii „for”, se așteaptă 5 secunde pana la următoarea comanda „find” pentru a evita detectarea aceluiași fișier.
2. Funcția „process\_file” are ca scop încărcarea fișierului in platforma HybridAnalysis folosind API-ul expus de aceasta si vizualizarea rezultatului ce conține informații despre tipul de fișier încarcă (**empty, clean, suspicious, malicious**). Structura necesara încărcării fișierului pe API este publica si expusa printr-o documentație:

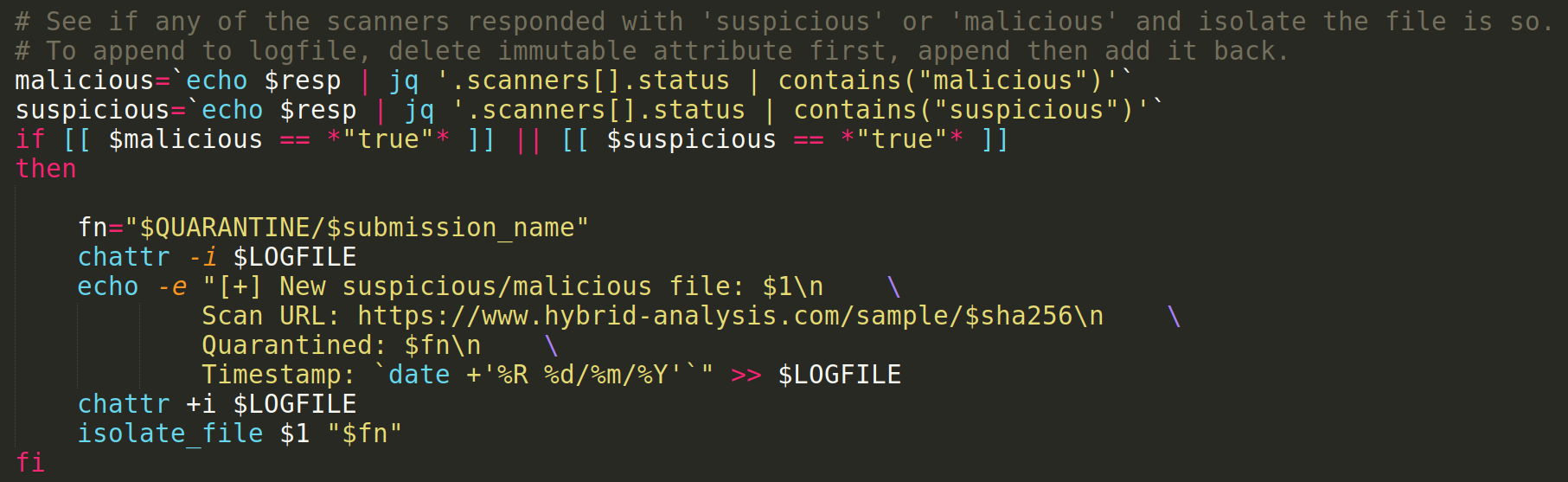


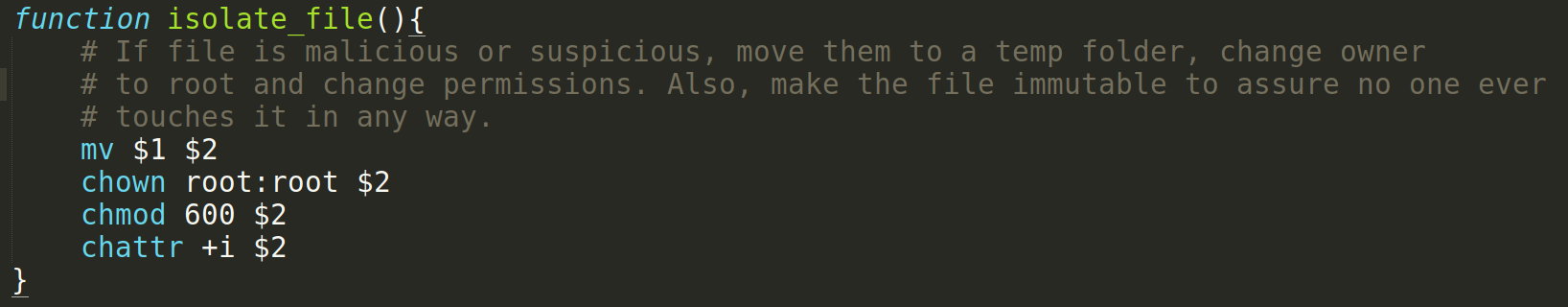
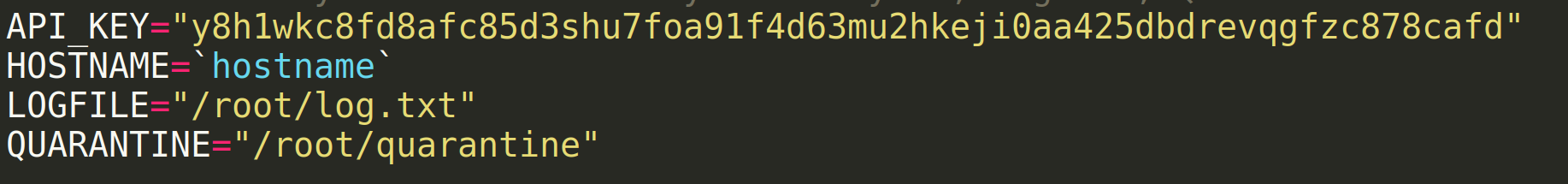


1. Daca fișierul încărcat nu este gol, parsăm răspunsul de la API si extragem id-ul submisiei, starea scanării („finished”) dar si valoarea SHA256 aferenta submisiei.
2. Întrucât fișiere de dimensiuni mari necesita un timp de analiza mai mare, prin parametrul „finished” putem afla daca analiza s-a terminat sau nu. In cazul in care analiza este încă in proces, trebuie sa interogam o alta ruta de API pentru a afla când submisia s-a terminat, obținând astfel datele necesare clasificării fișierului.
3. In acest punct, in variabila „resp” avem detaliile necesare pentru a putea concluziona daca fișierul este malițios sau nu. Mai concret, in obiectul **JSON** putem verifica conținutul cheii „scanners” ce oferă o lista a vendorilor interogați cat si detalii despre verdictul oferit de aceștia. Pentru a lua decizia carantinarii fișierului, am verificat daca statusul de la cel puțin un vendor a fost „malicious” sau „suspicious”. Desigur ca aceste criterii se pot customiza ușor in funcție de preferințele administratorului de sistem.







1. In imaginea de mai sus, carantinarea fișierului se realizează prin funcția „isolate\_file” ce primește ca parametru calea absoluta către fișierul ce trebuie carantinat cat si locația unde acesta va fi mutat (locația de carantinare). Înainte de a apela aceasta funcție, am generat o alerta aferenta acestui fișier ce a fost detectat malițios si am salvat datele legate de numele fișierului, ora, locația fișierului carantinat cat si URL-ul expus de API pentru a vizualiza raportul de scanare. Procesul de carantinare consta in mutarea fișierului într-o locație accesibila doar de administratorul sistemului cat si modificarea permisiunilor fișierului dar si setarea sa ca imutabil (nu poate fi alterat).
2. Singurul lucru care mai trebuie definit pentru o buna funcționare a scriptului sunt variabilele globale cu dețin valorile pentru cheia de API, hostname-ul sistemului, fișierul in care se vor salva alertele dar si folderul in care se vor carantina toate alertele.

## Raportarea si centralizarea datelor

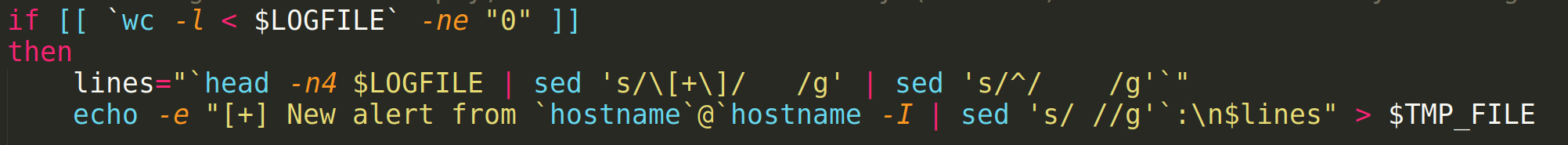
Prin raportare ne referim la procesul de transmitere a datelor de la client la server. In cazul nostru, pentru a transmite alertele de la toți clienții către server, s-a instalat pe client, prin scriptul „run.sh”, un serviciu care preia datele din fișierul de log si le transmite către server. Acest lucru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

Astfel, aceasta secțiune este compusa din 3 sub-secțiuni, fiecare referindu-se la un mecanism distinct prin care se realizează raportarea si centralizarea datelor:

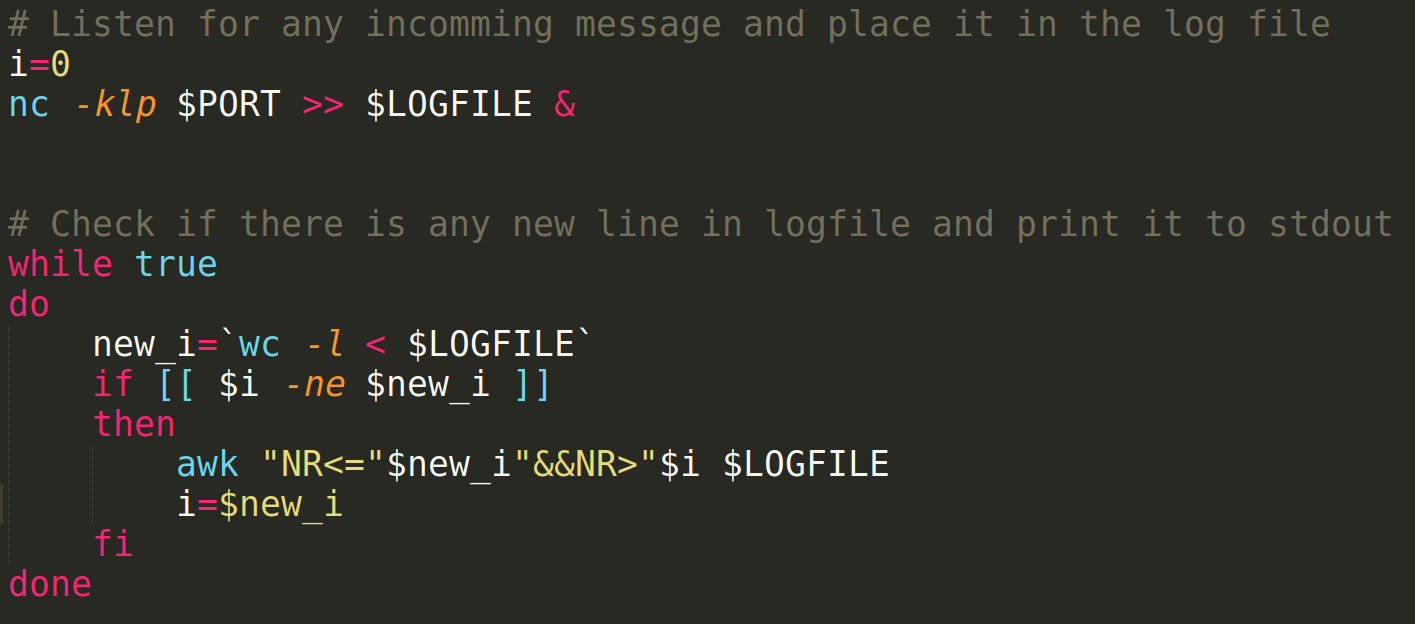
* Transmitere client –> server
* Recepționare server
* Centralizare server

### Transmitere client -> server

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

1. Verifica daca exista măcar o linie in fișierul de log. In realitate, o intrare in fișierul de log va avea mereu exact 4 linii întrucât acesta este formatul definit al alertelor. Daca aceasta condiție este îndeplinita, atunci extragem primele 4 linii (o alerta) din fișier, le formatăm corespunzător pentru a putea fi trimise ulterior către server folosind netcat.
2. In continuare, (condiția este îndeplinita), se șterg primele 4 linii din fișier întrucât acestea au fost deja consumate (copiate in alt fișier). In final, se trimite fișierul temporar generat către server folosind utilitarul netcat după care se așteaptă o secunda pana la următoarea verificare a fișierului de log pentru alte intrări/alerte. Ce se poate observa este ca pentru orice operație de modificare a conținutului fișierului de log, am folosit utilitarul „chattr” pentru a șterge si adăuga atributul de imutabilitate fișierului.

### Recepționare server

Recepționarea alertelor la nivel de server se face prin scriptul „server.sh” care preia toate datele primite pe portul 1337 si le plasează in fișierul de log. Acest lucru se realizează prin utilizarea comenzii nc ce deschide local portul 1337 si redirectează orice primit date primite pe acest port către fișierul de log. Intr-o bucla infinita se verifica constant daca fișierul de log conține mai multe linii decât la ultima verificare, caz in care le afișează (stdout). 

### Centralizare server

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

# 6. Studiu de caz / Evaluarea rezultatelor

De completat

# Concluzii

De completat

# Bibliografie

De completat

# Anexe

De completat