UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Sistem de Detecție de Anomalii Folosind Tehnologia eBPF

propusă de

Alexandru Grosu

Sesiunea: *iulie, 2019*

Coordonator științific
Lect. Dr. Dragoș Teodor Gavriluț

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

Sistem de Detecție de Anomalii Folosind Tehnologia eBPF

Alexandru Grosu

Sesiunea: *iulie, 2019*

Coordonator științific
Lect. Dr. Dragoș Teodor Gavriluț

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemntatul(a) ……

domiciliul în ……..

născut(ă) la data de ……., identificat prin CNP ……., absolvent(a) al(a) Universității „Alexandru Ioan Cuza" din Iași, Facultatea de …….. specializarea ……., promoția ……., declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul: _____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

Anexa 5

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Titlul complet al lucrării*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Prenume Nume*

_____

(semnătura în original)

Cuprins

Introducere

O dată cu creșterea complexității sistemelor informatice, cauzată de nevoia de a crește resursele și funcționalitatea sistemelor, a început să crească șansa apariției unei anomalii în sistem. O anomalie într-un sistem informatic reprezintă un comportament diferit față de cel așteptat, considerat normal. Detecția de anomalii este o problemă de actualitate în domenii variate, cum ar fi:

- detectarea intruziunilor
- detectarea fraudelor
- detectarea daunelor industriale
- medicină
- procesare de imagini
- detecția anomaliilor din texte
- sisteme de senzori
- alte domenii

Serviciul de detecție al anomaliilor prezentat în această lucrare are rolul de a detecta evenimente petrecute în cadrul unui sistem informatic ce sunt considerabil diferite față de evenimentele uzuale. Această sarcină are o importanță foarte mare deoarece în cazul în care o anomalie nu este descoperită și tratată la timp poate aduce daune imense.

Motivație

Motivul alegerii temei „Sistem de Detecție de Anomalii Folosind Tehnologia eBPF” este reprezentat de interesul personal pentru acest domeniu. Faptul că în ultimii ani atacatorii au reușit să extragă o cantitate imensă de date din marile companii, provocând daune de milioane de dolari, în unele cazuri de miliarde de dolari, mi-a sporit interesul în domeniu și mi-a dat dorința de a găsi metode optime pentru detecta aceste atacuri. De asemenea, urmărind noile tehnologii apărute mi-a fost atrasă atenția de către eBPF, o tehnologie ce a apărut în kernelul Linux la sfârșitul anului 2014 dar ce a luat amploare începând cu anul 2017 din cauza uzabilității și a noilor caracteristici adăugate tehnologiei. Având în vedere că atacurile informatice au crescut și că aceasta tehnologie promițătoare mi-a captat interesul am fost motivat să dezvolt acest proiect.

Gradul de Noutate

Deși tema „Detecții de Anomalii” nu este nouă, abordarea personală a temei diferă prin tehnologiile folosite și prin modul de abordare a problemei, ce se bazează pe învățarea comportamentului fiecărui sistem în parte nu pe detecția unor modele prestabilite de către dezvoltatori. Proiectul oferă posibilitatea utilizatorilor experimentați de a configura și de a

adăuga extensii pentru ca proiectul să ofere o detecție cât mai precisă urmărind doar evenimentele necesare utilizatorului.

Obiectivele Lucrării

Principalele obiective ale lucrării sunt:

- detecția unui comportament anormal al sistemului ce depășește limitele învățate
- analiza comportamentului anormal pentru a ajuta utilizatorul să investigheze într-un timp cât mai scurt
- avertizarea acestuia într-un mod care îl va informa rapid si îi va oferi datele necesare investigării anomaliei apărute
- punerea la dispoziție a unor decizii rapide pentru utilizator în cazul unei anomalii minore

Descrierea Sumară a Soluției

Pentru a asigura securitate sistemului, o bună abordare pentru a putea detecta anomaliile unui sistem informatic este bazată pe urmarea pașilor de mai jos:

- Scanarea evenimentelor dorite din sistemul de operare
- Clusterizarea și clasificarea evenimentelor noi pentru a putea detecta valorile aberante
- În cazul apariției unei anomalii se face corelarea evenimentelor petrecute în sistem pentru a investiga cauza anomaliei
- Notificarea administratorului de sistem în cazul apariției unei anomalii

Soluția propusă folosește o nouă tehnologie implementată în kernelul Linux , numită eBPF, ce are rolul de a monitoriza o serie de evenimente petrecute în sistemul de operare fără a afecta semnificativ performanța sistemului. Această tehnologie va garanta performanța în componentele proiectului ce au rolul de monitorizare a sistemului.

Pentru a folosi acest serviciu este nevoie de mașină informatică ce are instalată o distribuție de Linux cu o versiune de kernel mai nouă de 4.4 (recomandat este ca versiunea de kernel să depășească 4.9 din cauza posibilelor probleme ce pot apărea din cauza interfeței eBPF pe distribuțiile Ubuntu).

1. Contribuții

Proiectul realizat pentru lucrarea de licență are tematica „Detecții de Anomalii” din nevoia de a avea un sistem care poate să detecteze un comportament diferit într-un sistem pentru a putea controla performanța și pentru a detecta comportamente malițioase. Cu ajutorul profesorului coordonator am reușit să decid cele mai bune practici pe care le pot pune în aplicare în acest proiect pentru a oferi o detecție cât mai bună de anomalii având cât mai puține alarme false.

Principalele contribuții aduse de mine sunt:

- Modelarea structurii proiectului
- Implementarea Proiectului
- Alegerea evenimentelor ce trebuie monitorizate
- Extragerea de evenimente necesare din kernel folosind interfața eBPF
- Abstractizarea evenimentelor pentru clusterizare
- Decizia metricului de distantă folosit între 2 stări de sistem
- Decizia folosirii algoritmului de clusterizare DBSCAN
- Aplicația de android ce primește notificări legate de anomalie și din care se poate decide cum se va proceda în cazul unei anomalii pe baza unei investigații rapide

Cunoștințele pe bază cu ajutorul cărora am putut realiza aplicația se bazează în mare parte pe informațiile dobândite la cursurile:

- Sisteme de Operare
- Programare Orientata pe Obiecte
- Programare Avansată
- Ingineria Programării
- Rețele de Calculatoare
- Practica Python
- Tehnici de Programare pe Platforma Android

2. Analiza Soluției

2.1 Detectarea Anomaliilor

Detectarea anomaliilor într-un sistem este o sarcină foarte grea din cauză că trebuie menținut foarte bine echilibrul dintre alarme false și o detecție bună, altfel vor apărea situații în care o persoană este alarmată pentru un caz de anomalie inexistent (ar induce panica în situații cum ar fi detecția unei anomalii în unul din sistemele unei centrale nucleare) sau s-ar ajunge în momentul în care o anomalie de o importanță ridicată nu ar fi detectată. Aceste evenimente anormale pot apărea începând cu un defect apărut în contextul unui proces ajungând și la prezența unui intrus în sistem. În toate cazurile blocarea cauzei anomaliei trebuie făcută imediat pentru a nu compromite sistemul.

Pentru a fi sigur că balanța este păstrata în punctul în care detecția deține un procent cât mai mare iar alarmele false tind spre zero am început să cercetez însemnătatea anomaliilor și am concluzionat că anomaliile sunt de fapt valorile aberante dintr-un set de date.

Pentru a detecta valorile aberante exista trei tehnici fundamentale:

- Învățare Nesupervizată
- Învățare Supervizată
- Novelty Detection (Detectarea Noutăților)

2.1.1 Învățare Nesupervizată

Verificarea valorilor aberante fără a avea cunoștințe anterioare despre comportamentului sistemului. În acest caz se folosește învățarea nesupervizată. Această abordare constă în clusterizarea setului de date apoi cele mai îndepărtate elemente sunt considerate, fie un procent dintre ele, fie doar cele ce depășesc un prag, ca fiind valorile aberante ale setului de date.

În cazul învățării nesupervizate există 2 sub tehnici:
- diagnosticarea valorilor aberante și tratarea lor
- acomodarea valorilor aberante în setul de date pentru a oferi o metoda robusta de clasificare

În general este preferată metoda diagnosticării valorilor aberante pentru a menține setul de date cât mai mic ca să nu afecteze performanța sistemului. În cazul în care se dorește și o clasificare specifică a tipului de anomalie va fi folosită metoda acomodării.

2.1.2 Învățare Supervizată

Învățarea Detectorului de Anomalii care sunt valorile normale și care sunt cele aberante. În acest caz se folosește învățarea supervizata de aceea va fi nevoie de un set semnificativ de date care au deja asignată o clasă (ori valoare normală, ori valoare aberantă) sau dacă este nevoie de segmentarea în mai multe tipuri de comportamente se vor asigna și subclase valorilor (în cazul anomaliilor se poate ști mai exact ce tip de anomalie a apărut în sistem).

Una din problemele acestei tehnici este faptul ca setul de date trebuie să fie mare, să conțină un număr aproximativ egal de evenimente normale și evenimente anormale. O altă problemă ar putea fi faptul că este limitată la o serie de clase cunoscute. Spre exemplu, dacă clasificatorul a primit ca date de intrare doar 2 tipuri de comportamente anormale (ex: prea multe fișiere scrise pe disc într-un interval scurt de timp și prea puține conexiuni la un anumit server cu care comunica de obicei) când va descoperi un al 3-lea tip de comportament anormal sunt șanse să nu îl clasifice în una din clasele evenimentelor anormale ci în una din clasele evenimentelor normale.

2.1.3 Novelty Detection (Detectarea Noutăților)

Învățarea Detectorului de Anomalii doar care sunt valorile normale și în cazul în care seamănă cu un eveniment normal învățarea sistemului și unele valori anormale. Autorii ce au venit cu acest model de învățare au numit tehnica “Novelty Detection” sau “Novelty Recognition”, traduse “Detectarea Noutăților” sau “Recunoașterea Noutăților”. Pentru a colecta datele pentru acest tip de detecție este nevoie de un sistem ce oferă garanția ca produce doar evenimente normale.

Din cauza faptului că a treia tehnică este cea mai la îndemâna deoarece nu este nevoia de a colecta și valori aberante (cum trebuie la tehnica 2) și din cauză că se pliază bine pe seturile de date dinamice am concluzionat că ar fi cea mai potrivită tehnică pentru acest proiect.

2.2 Monitorizarea Evenimentelor

O dată cu creșterea complexității kernelului Linux printre distribuțiile de Linux a apărut și nevoia monitorizării resurselor și comportamentului acestora pentru a preveni anumite anomalii și pentru a analiza performanța. Din această cauză kernelul Linux vine cu funcționalități pentru a putea urmări evenimentele în sistemul de operare.

2.2.1 Metode Disponibile pentru Monitorizarea Evenimentelor

Printre numeroasele unelte ce vin cu kernelul Linux sunt puse la îndemâna câteva metode de urmărire a evenimentelor din cadrul sistemului de operare. Voi comenta asupra dezvoltării unui modul de kernel pentru a monitoriza evenimentele, despre tehnologia Linux Auditing System și despre tehnologia eBPF menționată și în introducerea lucrării de licență.

Dezvoltarea unui modul de kernel

Un modul de kernel este un program ce poate fi încărcat sau scos din kernel în timp real la cererea utilizatorilor cu privilegii de administrator. Acest program rulează la nivel de kernel de aceea are permisiuni să citească sau modifice orice date din kernel. Poate fi considerat echivalentul "driver-ului" din sistemul de operare Windows. Acest concept este folosit pentru a extinde funcționalitatea kernelului fără a trebui repornită mașina pe care rulează sistemul de operare.

Prin intermediul unui modul de kernel se poate modifica structura *sys_call_table* din kernel pentru a o face să ruleze apelurile de sistem personalizate de către dezvoltatorul modulului. Această metodă este dificil de implementat deoarece este greu de găsit programatic structura *sys_call_table* (trebuie găsită adresa de memorie unde se află) și trebuie ca dezvoltatorul să aibă în vedere și performanța funcțiilor ce vor înlocui apelurile de sistem implicite. Deși implementarea trebuie să fie complexă din punct de vedere tehnic ea oferă și un control deplin asupra apelurilor de sistem și nu provoacă mari daune asupra performanței (în funcție de cât de eficientă este funcția care înlocuiește apelul de sistem). Prin intermediul altui serviciu care comunică cu acest modul de kernel și putem genera evenimente pe baza apelurilor de sistem apelate in timp real.

Pe lângă apeluri de sistem putem urmări toate apelurile de funcții din procesele ce rulează și să generăm evenimente pe baza simbolului funcției apelate. În acest caz ar trebui implementat un serviciu ce se folosește de primitivele ptrace pentru a observa accesul adreselor de memorie a funcțiilor pe care vrem să le urmărim. Aceste adrese pot fi găsite prin parsarea fișierelor executabil în format ELF, sau, deși nu este recomandat, prin parsarea direct a memoriei procesului unde se afla modulul principal încărcat, și verificând apoi adresele de memorie atribuite anumitor simboluri care ne interesează.

Tehnologia Linux Auditing System (Sistemul de auditare din Linux):

Începând cu versiunea de kernel 2.6 este implementată tehnologia Linux Auditing System ce are scopul de a înregistra evenimente din sistemul de operare cum ar fi apelurile de sistem. Această tehnologie este bazată pe reguli și generează un fișier cu înregistrările ce satisfac regulile și informații detaliate despre evenimentele respective, cum ar fi:

- data și timpul

- tipul evenimentului
- rezultatul evenimentului
- utilizatorul ce a generat evenimentul
- modificări a configurării auditului sau încercări de accesare a fișierului de înregistrări
- etc.

Linux Auditing System are două mari componente, cea din spațiul kernelului și cea din spațiul utilizatorului care la rândul ei este împărțita în mai multe subcomponente. Componenta ce se afla în spațiul kernelului se ocupă de captarea evenimentelor ce se potrivesc cu regulile setate de către utilizator și le transmite la componenta principală din spațiul utilizatorului. Cea de-a doua componentă este cea ce rulează în spațiul utilizatorilor și care are rolul de a consuma evenimentele prinse pe baza regulilor de către componenta din kernel urmând ca apoi să le scrie pe disc.

Componenta principală din spațiul utilizatorului este daemonul auditd care este pornit și controlat de către un fișier de configurare ce se găsește la calea: /etc/sysconfig/auditd. Această componentă are rolul de a scrie evenimentele de auditare ce au fost generate de interfață din kernel în fișierul aflat la calea /var/log/audit/audit.log. Funcțiile suplimentare ale sistemului de auditare sunt controlate de configurările găsite la calea /etc/audit/auditd.conf.

O a doua componentă din sistem este audispd ce este dispecerul sistemului și este folosit în trimiterea notificărilor de apariție a evenimentelor către alte aplicații specificate de către utilizator. Daca dispecerul alege să trimită evenimentele către alte aplicații el poate renunța la scrierea pe disc a mesajelor cu evenimente primite din interfață din kernel.

Bazate pe evenimentele scrise pe disc în fișierul audit.log alte două componente ce aureport și respectiv ausearch. În cazul în care utilizatorul a ales să nu scrie evenimentele pe disc aceste două componente sunt inutilizabile. Aureport oferă puterea utilizatorului de a face rapoarte personalizate pentru a fi folosite în diferite situații, cum ar fi generarea unor grafice bazate pe comportamentul sistemului rezultat din raport. Utilitatea ausearch este folosită pentru a căuta anumite evenimente în audit.log după anumite caracteristici dorite de utilizator.

O altă componentă importantă din spațiul utilizatorului este audictl care controlează sistemul prin interpretarea regulilor (comenzi audictl) și trimiterea setărilor la componenta din kernel. Această componentă preia regulile din fișierul aflat la calea: /etc/audit/audit.rules și este pornita la începutul sistemului de operare imediat ce este pornit daemonul auditd.

O componentă des folosită este autrace, unealta ce are rolul de a urmări comportamentul unui proces specificat de utilizator într-o maniera asemănătoare cu strace. Rezultatele capturate de autrace sunt scrise în audit.log și pot fi folosite la fel ca orice alte evenimente capturate de interfață. A se lua în considerare faptul ca autrace nu comunică cu auditd ci consumă evenimentele direct din interfața din kernel.

Din scurta descriere de mai sus a sistemului Linux Auditing System putem observa următoarele cazuri în care poate fi utilizat, deși nu este limitat doar la aceste cazuri:

- Urmărirea accesărilor fișierelor
- Monitorizarea apelurilor de sistem
- Înregistrarea comenzilor rulate de utilizatori
- Înregistrarea evenimentelor de securitate
- Monitorizarea accesului în rețea

Tehnologia eBPF (extended Berkeley Packet Filter):

Tehnologia eBPF este creată pe baza originalului BPF (Berkley Packet Filter) ce avea scopul de a captura și filtra pachetele de rețea ce erau prinse de anumite reguli date de către administratorul sistemului. Aceste reguli erau programe de dimensiuni mici ce aveau menirea să ruleze într-o mașină virtuală situată în kernel. Din cauză că tehnologia BPF era depășită din cauză că mașina virtuală ce rula codul compilat al regulilor avea un număr semnificativ mai mic de instrucțiuni decât cele ale procesorului fizic și nu folosea puterea deplină a procesoarelor moderne, Alexei Starovoitov a venit cu ideea de a moderniza mașina virtuală din kernel pentru a avea setul de instrucțiuni cat mai aproape de instrucțiunile procesoarelor fizice pentru a crește performanța în noua tehnologie eBPF. O altă noutate în eBPF este introducerea a apelului de sistem bpf pentru a avea posibilitatea de a apela funcțiile din kernel fără a afecta prea mult performanța.

Programele eBPF sunt atașate interfeței eBPF în kernel și au acces la toate structurile de date din kernel din această cauză aplicabilitatea eBPF-ului s-a extins cu mult fata de originalul BPF astfel încât rolul tehnologiei nu este doar de a filtra pachetele de rețea. Acum eBPF-ul oferă utilizatorului puterea de a urmări o mare marjă de evenimente din kernel (inclusiv apelul unor funcții specifice unui singur elf prin intermediul simbolurilor din fișierul binar), mai oferă posibilitatea de a restricționa un proces la anumite apeluri de sistem pe care le poate face și chiar poate lăsa utilizatorul să modifice setările unui socket ce are o conexiune deja stabilită.

Un mare avantaj al folosirii tehnologiei eBPF este faptul că suportă ca datele returnate de programul eBPF încărcat în mașina virtuală din kernel să fie selectate de dezvoltator si puse în structuri de date compatibile. Acest avantaj oferă posibilitatea ca evenimentele returnate să conțină orice informație dorită de utilizator în contextul tipului de eveniment.

Un dezavantaj al tehnologiei eBPF este faptul ca orice program eBPF făcut de utilizator trebuie compilat în interiorul directorului cu sursele de kernel din sistemul de operare. Din cauza acestui dezavantaj a apărut proiectul BCC ce oferă o mulțime de utilități pentru a scrie programe eBPF și pentru a le încărca în mașina virtuala eBPF fără a trebui făcută legatura manual cu sursele de kernel.

Interfața eBPF oferă posibilitatea utilizatorului de a încărca programe eBPF de tip KProbes și UProbes în mașina virtuală.

KProbes este un mecanism de debug pentru kernelul Linux folosit pentru a monitoriza evenimente din spațiul kernelului. Kprobes depinde de arhitectura procesorului, de aceia este foarte probabil ca unele caracteristici să difere de la o arhitectura la alta. Un KProbe este definit de o funcție rulată înainte de execuția respectiv una rulata după terminarea execuției instrucțiunii la care este atașat KProbe-ul.

Mecanismul de UProbes este similar cu cel de KProbes, diferența fiind că evenimentele tratate de UProbes sunt funcții din spațiul utilizatorului și nu a kernelului (un exemplu ar fi urmărirea apelului funcției dintr-un fișier executabil ce are simbolul strlen).

2.2.2 Alegerea metodei potrivite pentru monitorizarea evenimentelor

Deși sunt multe metode de monitorizare a evenimentelor toate diferă între ele de aceia fiecare metodă are și avantaje și dezavantaje. Cele trei metode prezentate în secțiunea trecută sunt cele mai promițătoare din punctul meu de vedere deoarece fiecare are câte un avantaj puternic față de celelalte.

Avantajul folosirii unui modul de kernel ce interceptează apelurile de sistem dintr-un sistem de operare și a unui serviciu de urmărire a apelurilor funcțiilor dintr-un proces este faptul că oferă o putere nelimitată în limita faptului că putem scădea performanța sistemului. Totuși, deși această metodă ne oferă o putere extrem de mare asupra informațiilor pe care vrem să le colectăm din sistem timpul și calitatea implementării fac ca această metodă să fie pusa în practică doar în cazul în care celelalte metode nu ne oferă informațiile de care avem nevoie. Știind că celelalte opțiuni oferă destule informații pentru a putea monitoriza evenimentele sistemului concluzia este că această metoda nu va fi folosită în proiectul practic de licență.

Deși Linux Auditing System este bine implementat și este un serviciu de monitorizare puternic am decis ca în proiectul practic de licență să folosesc tehnologiile eBPF din cauza suportului pentru KProbes și UProbes, acestea interceptând tipurile de evenimente țința pentru tipurile de anomalii propuse pentru a fi detectate. O alta cauză ce a determinat folosirea eBPF este faptul ca este foarte ușor de utilizat și informațiile ce pot fi extrase de programele eBPF, fiind ușor de extras și modificat, se mulează mai bine pe nevoile proiectului.

2.2.3 Alegerea evenimentelor monitorizate

Din cauza faptului că pe parcursul implementării am observat că în funcție de comportamentul unui server este nevoie doar de unele tipuri de evenimente, iar alte tipuri de evenimente s-au dovedit a fi inutile, am ajuns la concluzia că în funcție de rolul sistemului ar

trebui monitorizate doar evenimentele cheie pentru a nu încărca setul de date care este stocat pe disc pentru a crește performanța. Din această cauză am modularizat modalitatea de a intercepta un anumit tip de eveniment astfel încât utilizatorul se poate abona din fișierul de configurare a sistemului la ce evenimente dorește.

Lista de evenimente selectate pentru prezentarea proiectului practic din lucrarea de licență este următoarea:

- Evenimentele rezultate în urma deschiderii unui fișier prin intermediul interfeței Virtual File System
- Evenimentele rezultate în urma scrierii într-un fișier prin intermediul interfeței Virtual File System
- Evenimentele rezultate în urma efectuării apelului de sistem recvfrom
- Evenimentele rezultate în urma efectuării apelului de sistem sendto
- Evenimentele rezultate în urma efectuării apelului de sistem execve
- Evenimentele rezultate în urma efectuării unei conexiuni tcpv4

Virtual File System este un strat abstract peste un sistem de fișiere concret. Scopul acestei interfețe este de a oferi posibilitatea utilizatorului de a accesa diferite tipuri de sisteme de fișiere într-un mod uniform. Un exemplu specific ar fi accesul la fișierele de pe disc fără ca utilizatorul să fie afectat de tipul de sistem de fișiere folosit pe acea partiție de disc. VFS acoperă orice caz de operații peste fișierele partițiilor montate. Înafara de operațiile făcute pe un disc fizic mai sunt unele cazuri de operații care ne interesează și VFS ne va ajuta să primim evenimente declanșate de către aceste operații. Unele din cazurile care ne interesează sunt operațiile prin intermediul FUSE (Filesystem în Userspace) care acoperă și operațiile peste sistemele de fișiere făcute în spațiul utilizatorului. FUSE încadrează și operațiile peste fișierele distribuite de altă mașină în rețea printr-un protocol specific pentru acest caz (ex: protocoalele ftp sau samba).

Din cauza faptului că apelurile de sistem send și recv ajung să apeleze sendto respectiv recvfrom, faptul că monitorizam doar evenimentele apelurilor de sistem sendto și recvfrom va acoperi toate operațiile de transmitere de date ce folosesc socket indiferent de protocolul de comunicare folosit.

Decizia de a monitoriza apelurile de sistem execve vine din motivul că această funcție are rolul de a executa un program. Acest eveniment ne va avertiza de fiecare data când este creat un proces nou de aceea vom putea analiza momentele în care apar procese ale programelor ce nu au fost niciodată rulate sau a programelor ce nu obișnuiau să ruleze atât de des.

Ultimul tip de eveniment propus să fie monitorizat este cel al conexiunilor TCP v4. Acest eveniment ne va ajuta să urmărim atacurile de tip DOS și ne va informa în cazul în care un proces nou ce are posibile intenții malițioase face diferite conexiuni, fie că are rolul de server, fie că are rolul de client.

2.3 Abstractizarea Datelor

După ce sistemul de detecție de anomalii termină procesul de colectare de informații legate de evenimentele petrecute în sistem el va stoca datele colectate pe categorii, în funcție de tipul evenimentului, și apoi le va retine în contextul intervalului de timp în care a fost înregistrat. Acest fapt ne va ajuta pe viitor să creăm o stare a sistemului într-un interval de timp, care în cazul nostru, a fost ales să fie o secundă. În figura 2.3.1 se pot observa cum arată datele reținute într-un JSON pentru evenimentele de tip scriere în fișier (file_write).


```
"2019-06-16_15:50:37": {
    "12200": {
        "count": 697,
        "cmdline": "python tracer.py configs/default_config.json ",
        "comm": "tracer.py"
    },
    "total_count": 709,
    "2820": {
        "count": 12,
        "cmdline": "/usr/lib/gnome-terminal/gnome-terminal-server ",
        "comm": "gnome-terminal-"
    }
},
"2019-06-16_15:50:36": {
    "12200": {
        "count": 697,
        "cmdline": "python tracer.py configs/default_config.json ",
        "comm": "tracer.py"
    },
    "total_count": 698,
    "2080": {
        "count": 1,
        "cmdline": "/usr/lib/ibus/ibus-ui-gtk3 ",
        "comm": "gdbus"
    }
},
```


Figura 2.3.1

În contextul fiecărui interval de timp putem observa câmpul total_count ce arată numărul total de evenimente de tip scriere în fișier petrecute în sistem în acel interval temporal și apoi se pot observa pid-ul (id-ul procesului) proceselor ce au generat evenimentele. În cadrul fiecărui pid se găsește în câmpul count ce reprezintă numărul de exercitări a operației respective (in cazul nostru file_write). Câmpul comm din contextul pid-ului face referire la Command Name (numele comenzii) ce este extras din structura din kernel task_struct. Acest câmp are dimensiunea de 16 bytes (TASK_COMM_LEN) și este în general inițializat la apelarea funcției setup_new_exec() cu numele executabilului fără calea completa spre fișier în limita a 15 bytes. Ultimul câmp din contextul pid-urilor este cmdline care, intuitiv, conține linia de comandă folosită la rularea programului.

După ce avem colectate aceste date este posibilă generarea unei stări a sistemului pentru acel interval de timp. Voi defini o stare a sistemului, reprezentată ca în tabelele de mai jos, ca fiind o listă de dimensiunea $len(event) \cdot len(comm)$, unde valoarea $len(event)$ este egală cu dimensiunea listei event respectiv $len(comm)$ este egală cu dimensiunea listei comm, ce conține la linia $\mathrm{i} \cdot \mathrm{nr\_evenimente} + \mathrm{j}$ $\mathrm{numărul}$ de operații de tipul $event_i$ în cadrul procesului $comm_j$. Listele event și comm au o ordine bine definită ce trebuie respectată. De fiecare dată când va apărea un Command Name (comm) nou în sistem acesta va fi adăugat în lista comm iar stările trecute ale sistemului vor fi actualizate să conțină noua dimensiune, iar

datele lipsa vor fi înlocuite cu 0. Din cauză că o stare va conține multe valori nule stările sistemului vor fi ținute în memorie ca o linie dintr-o matrice rara în care vom avea valoarea și poziția pe care se afla a valorilor nenule.

Avem listele:

$$\text{event} = \{\text{"file\_open"}, \text{"file\_write"}\}$$
$$comm = \{\text{"gnome-terminal-"}, \text{"}tracer.py\text{"}, \text{"}gdbus\text{"}\}$$

Având notațiile *fo = fișiere deschise* și *fw = scrieri in fișiere*, o stare de sistem posibilă, formată având câmpurile din listele de mai sus, poate fi următoarea:

Numele Comenzii	gnome-terminal-		tracer.py		gdbus	
Tipul evenimentului	fo	fw	fo	fw	fo	Fw
i	0	1	2	3	4	5
$state_i$	2	12	1	697	0	0

2.4 Principalele stări ale detectorului de anomalii

După ce au fost prezentate în capitolul 2.1 (Detectarea Anomaliilor) decizia a fost să fie folosită în acest sistem de detecție de anomalii metoda a 3-a, o metoda ce memorează stările normale ale sistemului (în unele cazuri memorează și stările anormale, dar la noi nu este cazul) pentru a putea decide daca stările noi generate se încadrează în spațiul normalului. În cazul în care aceste stări nu pot fi considerate similare cu nici una din stările normale ele vor fi considerate anomalii iar sistemul va urma protocolul de comportament în cazul anomaliilor și utilizatorul va fi notificat.

Din cauză că metoda folosită va memora stările normale într-o perioadă iar apoi va trebui să clasifice stările noi putem deduce două moduri în care va funcționa sistemul:

- Modul de Învățare
- Modul de Detecție

2.4.1 Modul de Învățare

Acesta este modul care va colecta evenimentele considerate normale și le va memora după ce acestea sunt transformate în stări ale sistemului. Din cauză că în unele cazuri intervalul de timp ales de către administratorul aplicației poate fi destul de mic stările memorate de către detectorul de anomalii trebuie să fie filtrate. Având în vedere ca stările unui sistem pot coincide în două intervale diferite, sau cel puțin pot fi asemănătoare, am început prin a găsi o soluție pentru a vedea egalitatea și similaritatea a două stări.

Știind că o stare a sistemului are mai multe dimensiuni putem folosi o distanță comun folosită pentru date cu mai multe dimensiuni anume distanța euclidiana, definita ca:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Analizând formula când este aplicată peste două stări diferite ale sistemului, numite x și y, putem observa că distanța dintre x și y este reprezentată de radical din suma diferențelor la pătrat a numărului de evenimente de tipul dat de indexul $i \ mod \ len_{event}$ generate de comanda aflata în lista comm la indexul $\frac{i}{len_{comm}}$. Intuitiv, o distanța mai mare între două stări va fi când cele două stări conțin un număr de evenimente de același tip, făcute de același proces, cu mult diferit una fata de cealaltă. O problemă cu aceasta distanță aplicată pe stările sistemului este faptul că importanța unui tip de eveniment ce rar produce evenimente este la fel ca importanța unui tip de eveniment ce produce foarte multe evenimente. Un exemplu este momentul în care două stări au diferența 5 pe o coloana ce reprezintă un eveniment de proces nou și alte două stări au diferența 5 pe o coloana ce reprezintă un eveniment de scriere într-un fișier. În acest caz distanța dintre primele două stări va fi asemănătoare cu distanța dintre ultimele două stări, deși 5 noi procese apărute sunt mai alarmante decât 5 noi scrieri într-un fișier. Unul din pașii pentru a rezolva aceasta problema este ca sistemul să memoreze media apariției unui eveniment petrecut în contextul unui proces. Dacă avem media a fiecărei operații în parte făcute de un proces putem normaliza stările împărțind fiecare index dintr-o stare cu media ce-i corespunde.

Având listele comm și event, notațiile *fo = fișiere deschise* și *fw = scrieri in fișiere* și tabelul de mai sus:

$$\text{event} = \{\text{"file\_open"}, \text{"file\_write"}\}$$
$$comm = \{\text{"gnome-terminal-"}, \text{"}tracer.py\text{"}, \text{"}gdbus\text{"}\}$$

Numele Comenzii	gnome-terminal-		tracer.py		gdbus	
Tipul evenimentului	fo	fw	fo	fw	fo	Fw
i	0	1	2	3	4	5
$state_i$	2	12	1	697	0	0

Tabelul cu medii:

Numele Comenzii	Tipul Evenimentului	În medie
gnome-terminal-	fișiere deschise	1
gnome-terminal-	scrieri in fișiere	8
tracer.py	fișiere deschise	2
tracer.py	scrieri in fișiere	723
gdbus	fișiere deschise	1
gdbus	scrieri in fișiere	4

Aplicând normalizarea pe starea sistemului, vom avea starea normalizată:

Numele Comenzii	gnome-terminal-		tracer.py		gdbus	
Tipul evenimentului	fo	fw	fo	fw	fo	Fw
i	0	1	2	3	4	5
$state_i$	2	1.5	0.5	0.964	0	0

În această formă asemănarea între două stări va fi mult mai precisă, rezultatul bazându-se și pe media operațiilor petrecute în sistem în perioada de învățare.

Rezolvând problema filtrării a stărilor ce coincid, sau ce sunt asemănătoare cu stările deja memorate, la fiecare iterație a detectorului de anomalii vom înlătura noile evenimente apărute ce au o distanță mai mică de o valoare epsilon, aleasă empiric de administratorul aplicației. În cazul nostru vom folosi epsilon=2 deoarece, bazat pe tipurile de evenimente extrase de noi și pe procesele ce rulează pe mașina de test, stările cu o distanță mai mică decât această valoare au operațiile făcute pe mașină aproape identice.

O altă etapă a modului de învățare este salvarea pe disc a evenimentelor memorate și, în cazul în care administratorul dorește să vadă cam câte tipuri diferite de stări apar pe parcursul învățării se vor clusteriza evenimentele. Aceasta procedură ajută administratorul

aplicației să găsească o distanță potrivită pentru clasificarea stărilor noi care dacă nu se încadrează în raza acelei distanțe vor fi considerate anomalii. Intuitiv, cu atât este distanța mai mică cu atât detectorul de anomalii va fi mai strict cu evenimentele noi apărute pe sistem.

Având în vedere că similaritatea între două stări este indusă de o distanță cât mai mică, faptul ca datele sunt dense și au valori de tip zgomot cel mai potrivit algoritm de clusterizare este DBSCAN (Density-based spatial clustering of applications with noise). Modul în care acționează acest algoritm de clusterizare urmează următorii pași din imagine:

```
def puncteVecin(P, epsilon):
      return toate punctele la o distanta mai mica de epsilon fata de P


def extindeClusterul(P, puncte_vecin, C, epsilon, minim_de_puncte):
      # adaugam punctul P la cluster
      C.adauga(P)

      # parcurgem toate punctele din raza epsilon a lui P
      for P` in puncte_vecin:
            # daca nu este vizitat
            if not vizitat[P`]:
                  # trecem punctul ca vizitat
                  vizitat[P`] = True
                  puncte_vecin` = puncteVecin(P`, epsilon)

                  # daca satisface minimul de puncte dintr-un cluster
                  if len(puncte_vecin`) >= minim_de_puncte
                        puncte_vecin = puncte_vecin U puncte_vecin`

                  # daca punctul vecin nu se afla in alt cluster il adaugam in clusterul nostru
                  if P` not in clustere_formate:
                        C.adauga(P`)


def DBSCAN(D, epsilon, minim_de_puncte):
      # parcurgem fiecare punct nevizitat din setul de date
      for punct nevizitat P in setul_de_date:
            # marcam punctul ca fiind vizitat
            vizitat[P] = True

            # extragem toate punctele la o distanta mai mica de epsilon fata de P
            puncte_vecin = puncteVecin(P, epsilon)

            # daca nu satisface minimul de puncte dorit per cluster ignoram punctul
            if sizeof(puncte_vecin) < minim_de_puncte
                  ignora P
            else:
                  C = cluster nou
                  clustere_formate.adauga(C)

                  # extindem clusterul cu ajutorul punctelor vecine
                  extindeClusterul(P, puncte_vecin, C, epsilon, minim_de_puncte)
```

Figura 2.4.1.1

După ce avem clusterizate toate stările normale vom salva stările, împreună cu eticheta clusterului din care face parte, pe disc. Deși convertirea evenimentelor petrecute în contextul unui pid dintr-un anumit interval de timp la o stare a sistemului se face periodic, clusterizarea stărilor normale se face în următoarele cazuri:

- La intervale semnificative de timp stabilite de administrator pentru a asigura o copie de rezervă în cazul unei închideri bruște a sistemului
- Când sistemul de anomalii trece la modul de detecție
- Când sistemul este în modul de învățare și este închis

2.4.2 Modul de Detecție

Modul de detecție al sistemului de anomalii are rolul de a detecta și de a notifica utilizatorul de prezenta unei anomalii pentru a putea investiga problema și pentru a îndruma sistemul ce masuri să ia. După cum am prezentat în capitolul 2.4.1 ce avea în vedere modul de învățare al sistemului, două stări diferite ale sistemului pot fi comparate cu distanța euclidiană. Pentru a verifica dacă o stare este anomalie vom calcula distanța dintre starea ce trebuie verificata și stările normale. În momentul în care starea evaluata are distanța mai mică de un prag stabilit de către administrator va fi considerată ca fiind o stare normală. În caz contrar starea va declanșa un semnal de anomalie.

O alta componentă a proiectului este componenta ce se ocupa de notificarea utilizatorului. Aceasta componentă este compusă din două servere:

- Serverul de notificare a serviciului de android
- Serverul ce oferă un serviciu REST pentru a cere anomalii și pentru a lua masurile dorite de utilizator pe urma deciziilor făcute din aplicație

Utilizatorului îi va fi prezentată anomalia într-un mod text afișând detalii generice despre sistem și unele cauze ce ar fi putut provoca anomalia bazate pe scorul evenimentului din starea sistemului. Serviciul va suspenda procesele ce au scorul mare și va aștepta utilizatorul să decidă daca oprește de tot acele procese sau daca le va da semnal să își continue rularea. Utilizatorul mai are posibilitatea de a marca starea respectivă ca fiind o stare normală. În acest caz sistemul de detecție va adăuga starea la colecția de stări normale și va actualiza mediile evenimentelor normale din sistem.

În cazul în care statisticile rezultate scot în evidență un comportament ce afectează grav întreg sistemul utilizatorul poate să activeze starea critică. Un exemplu de stare critică poate fi reprezentat de un comportament complet anormal al bazei de date sau de indicii rezultate din raport că s-ar afla un intrus în sistem. Starea critică va intra în modul de salvare a sistemului, nivelul de rulare a daemonului sistemului (systemd) de nivel 1 și care oferă doar mod text, rulează ca root, nu oferă acces la rețele și suportă doar utilizatorul de root logat. În general acest nivel nu oferă aplicațiilor cu comportament de anomalie să corupă datele sistemului și le întrerupe accesul intrușilor întrerupând orice legătura cu rețeaua.

3. Dezvoltarea Proiectului

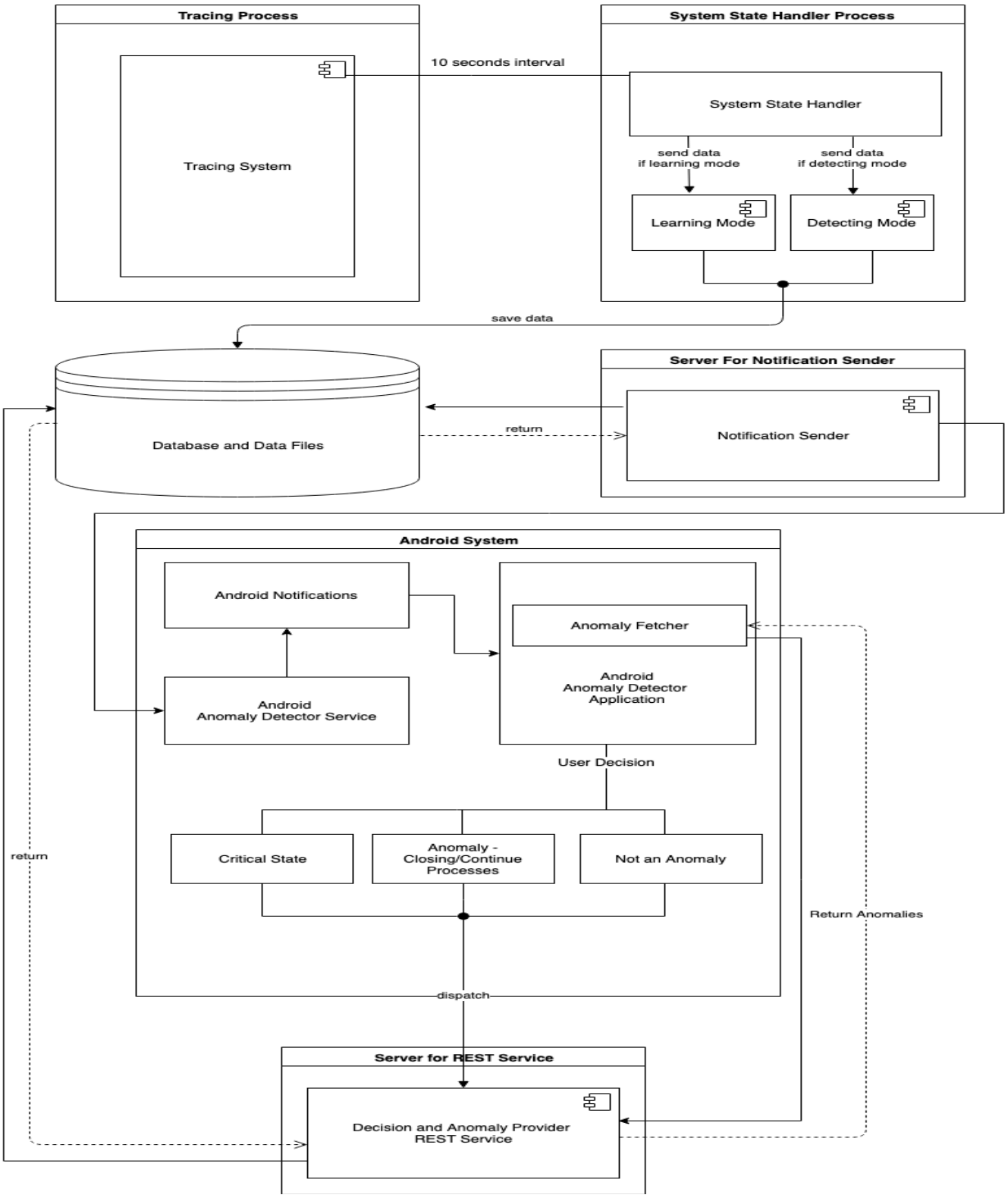

Figura 3.1.1

Din cauza complexității proiectului, pe parcursul dezvoltării au apărut diferite obstacole. Uzabilitatea, performanța și acuratețea detecțiilor au fost cele mai dificile probleme de rezolvat. Cea mai mare proporție din proiect este scris în limbajul de programare Python, urmat de Java pentru aplicația Android și C pentru KProbes ce vor fi compilate de interfața BCC și adăugat codul eBPF în lista de programe eBPF ce vor rula în mașină virtuala din kernel.

3.1 Diagrama Proiectului

După cum se observa în figura 3.1.1 proiectul are 6 mari componente ce lucrează în paralel.

Componenta Tracing System este componenta ce folosește interfața eBPF din kernel ca să capteze evenimentele esențiale mașinii pe care rulează. După cum am precizat și în capitolul 2.2.3, utilizatorul, care coincide cu administratorul sistemului și al aplicației, se poate abona în fișierul de configurare al serviciului la ce evenimente concluzionează că ar trebui să fie capturate. Din cauza nevoii utilizatorilor experimentați de a captura evenimente care nu au fost luate în calcul până acum, lista de evenimente disponibile la care se poate abona este reprezentata de fișierele din directorul “plugins”. Aceste fișiere sunt module ce sunt încărcate dinamic de aplicație în funcție de fișierul de configurare și respecta o structura specifica dar simplă pentru a fi și la îndemâna utilizatorilor începători. Rolul modulelor este de a pune la dispoziție un obiect de tipul BPF ce va comunica cu kernelul pentru a lua evenimentele generate din buffer. Din aceasta cauza fiecare modul va conține funcția “generate” ce va returna obiectul de tip BPF. Implementarea unei funcții generate dintr-un modul nou se face urmând următorii pași:

- crearea obiectului de tip BPF ce primește codul C care va fi compilat în cod mașină eBPF
- atașarea unui KProbe care va rula înainte să ruleze funcția ce generează evenimentul
- atașarea unui KProbe ce va rula după ce rulează funcția ce generează evenimentul, acesta returnând și datele evenimentului
- returnarea obiectului de tip BPF

Pentru o performanță optimă, datele returnate de KProbe sunt pid-ul procesului ce a generat evenimentul și numele comenzii (comm sau Command Name). Având aceste date putem colecta restul datelor ce ne vor ajuta, de exemplu linia de comandă, doar la apariția unui nou pid din intervalul de timp ce va defini starea sistemului. În cazul colectării mai multor date în KProbe-ul atașat pentru eBPF riscăm să pierdem evenimente.

Din cauză că fiecare tip de eveniment are propriul obiect BPF, colectarea datelor din bufferul de tip BPFPerfBuffer se va face iterativ, iterând fiecare obiect BPF și colectând datele într-un interval maxim de timp de 0.3 secunde. Decizia de a avea un interval de timp maxim pentru colectare provine din cauza evenimentelor rare, cum ar fi conexiuni TCP, care nu garantează să fie întâmpinate în fiecare stare de sistem.

În momentul în care am ajuns în momentul să trimitem datele către componenta System State Handler, moment stabilit de administratorul programului, componenta Tracing System va salva toate datele importante în baza de date și apoi va deschide o instanță noua de System State Handler ce va conține toate datele noi colectate de Tracing System și ce va rula ca proces separat.

Odată ce a fost deschisă o instanță de System State Handler aceasta va verifica dacă proiectul se afla în modul de învățare sau în modul de detecție, iar în cazul modului de detecție va verifica daca are activata opțiunea de a face detecție sau dacă trebuie să aștepte un răspuns de la utilizator. După cum am explicat în capitolele 2.4, ambele moduri, și cel de învățare, și cel de detecție, vor transforma datele primite de la componenta Tracing System în stări de sistem.

Modul de învățare va memora aceste stări într-un fișier pe disc ce va conține:
- lista events, este lista ce conține tipurile de evenimente într-o ordine consistentă
- lista comms, este lista ce conține toate numele comenzilor rulate pe mașină
- o listă cu toate stările de sistem considerate normale
- opțional, o listă cu toate etichetele clusterilor din care fac parte stările normale

De asemenea, modul de învățare generează un fișier de excepții ce conține o listă de expresii regulate care vor fi testate pe linia de comanda al proceselor ce urmează să fie închise. Daca aceste procese sunt considerate excepții atunci ele nu vor fi oprite de către sistem în cazul în care se constata ca ele au fost una din cauzele anomaliei. Totuși, administratorul este obligat să verifice corectitudinea comportamentului acelui sistem pentru a nu afecta mașina.

Modul de detecție va clasa evenimentele noi ori ca fiind normale, ori anomalii. În cazul în care o stare de sistem este considerata anomalie va fi trimisa la investigație unde i se va acorda un scor de anomalie și unde vor fi scoase în evidenta evenimentele ce au putut provoca anomalia. În urma acestei investigații se face un raport ce va fi stocat în baza de date și ce va fi trimis de către componenta Decision Handler and Anomaly Provider către utilizator.

Componenta Notification Sender are rolul de a monitoriza baza de date și în cazul unei apariții noi de anomalie să avertizeze utilizatorii. Aceasta componenta comunica cu serviciul de Android numit Anomaly Detector Service ce rulează în fundal și așteaptă să fie notificat. Când acesta este notificat el va face o notificare ce va fi afișata de către sistemul de operare Android.

Aplicația de Android este o alta componenta a proiectului ce are rolul de a primi notificări de anomalii și de a lua decizii asupra raportului stării de sistem. Anomaliile noi sunt primite de la componenta Decision Handler and Anomaly Provider. Când utilizatorul intră în contextul unei anomalii el va avea posibilitatea să citească raportul text, să oprească procesele suspendate care au provocat anomalia în cazul în care acestea nu sunt critice și nu ar trebui sa-si continue execuția și poate decide daca anomalia primita nu este de fapt o anomalie, ci un fals pozitiv, poate decide daca a terminat de investigat anomalia și de închis procesele problematice sau poate decide daca starea sistemului este una critică pentru a intra în modul sigur al sistemului, de nivel 1, folosit pentru sarcini administrative.

După ce una din cele trei decizii a fost luata aplicația o va trimite către componenta Decision Handler and Anomalt Provider pentru ca sistemul să acționeze conform deciziei. În cazul în care este anomalie sistemul va închide procesele alese de utilizator și va continua detecția de anomalii. În cazul în care nu este anomalie starea sistemului va fi salvată ca fiind o stare normală. În cazul în care s-a luat decizia ca situația este critica sistemul va apela comanda “init 1” pentru a pune sistemul să ruleze la nivelul 1, nivel ce suporta un singur utilizator logat și care nu are conexiune la internet.

3.2 Uzabilitatea și prezentarea raportului

Deși, în ansamblu, proiectul este configurat astfel încât să detecteze anomalii pe orice tip de sistem ce rulează o distribuție de Linux ce suporta interfața eBPF în kernel, este recomandată folosirea acestui serviciu doar pe mașinile de tip server ce au un comportament consistent. Un exemplu clar de comportament consistent este comportamentul unui server de baze de date.

Uzabilitatea a fost cea mai grea sarcina de îndeplinit deoarece utilizatorul trebuie să fie unul experimentat și trebuie să poată lua decizii pe baza unui raport ce conține doar informații generice despre starea sistemului ce este considerata anomalie. Un exemplu de raport poate fi observat în imaginile de mai jos:

```
########################################
Anomaly detected at:
   2019-06-17_19:32:06
Minimum distance from normal data:
   344.0223372838747
Anomaly event score:
   350.685192204
List of guessed anomaly cause:
----------------------------------------
process comm:  start_anomaly.py
process score: 3
no operations: 3 of type file_write

Possible processes which caused anomaly:
[s] PID 12281: python start_anomaly.py

----------------------------------------
process comm:  start_anomaly.py
process score: 344
no operations: 344 of type file_open

Possible processes which caused anomaly:
[s] PID 12281: python start_anomaly.py
```

Figura 3.2.1

```
----------------------------------------
All operations json:
{
   "file_open": {
      "sed": 14,
      "start_anomaly.py": 344,
      "lpstat": 50,
      "sh": 6,
      "thermald": 3,
      "tracer.py": 28
   },
   "process_create": {
      "start_anomaly.py": 1,
      "prlshprint": 1,
      "sh": 2
   },
   "network_recvfrom": {
      "cupsd": 9,
      "lpstat": 33
   },
   "file_write": {
      "gdbus": 63,
      "prlsga": 1,
      "start_anomaly.py": 3,
      "lpstat": 1,
      "prlshprof": 1,
      "prlcc": 2,
      "sed": 1,
```

Figura 3.2.2

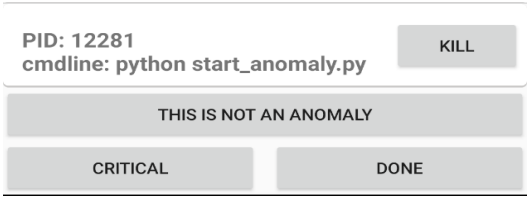

Figura 3.2.3

In figura 3.2.1, ca antet, este prezentată data și intervalul de timp în care a apărut anomalia, o distanță minimă față de datele normale și scorul anomaliei. După antet se observă o listă cu procesele ce ar fi putut provoca anomalia. În cazul nostru aceasta lista conține de două ori același proces din cauza ca el a depășit valorile normale învățate în cadrul a două tipuri de evenimente: file_write (scrieri în fișiere) și file_open (deschidere de fișiere). Fiecare element din aceasta listă mai conține o listă cu posibilele procese ce au provocat anomalia. În cazul nostru prima intrare conține o listă cu un singur element, și anume: “[s] PID 12281: python start_anomaly.py”. Eticheta “[s]” ne spune ca acest proces a fost oprit de către detectorul de anomalii pentru a nu-si desfășura în continuarea activitatea pana nu investighează administratorul. O altă etichetă disponibilă, ce apare foarte des în cazul falselor pozitive, este eticheta “[e]” ce reprezintă faptul ca aceasta linie de comanda este exclusa din clasa anomaliilor pe perioada învățării. Procesele ce au o linie de comandă exceptată nu vor fi oprite.

Se poate observa ca în cazul în care sistemul nu știe media aparițiilor a evenimentelor în sistemul normal el nu va normaliza câmpul respectiv rezultând un scor mai mare pentru procesele noi apărute în sistem. Acest fapt este unul bun având în considerare ca sistemul pe care rulează ar trebui să aibă un comportament consistent și nu ar trebui să permită operații noi. În cazul în care este nevoie să fie executată o operație nouă se va trece din aplicația de android la modul de învățare.

In figura 3.2.2 este reprezentată partea din raport în care sunt afișate sumar toate evenimentele întâmplate în sistem în acel moment. Din cauza dimensiunilor mari al acestui JSON figura 3.2.2 conține un JSON incomplet. Dacă nici aceasta parte din raport, nici partea în care sunt prezentate posibile anomalii nu dezvăluie nimic utilizatorul ar trebui să declare stare critică daca cazul este complet necunoscut, sau să declare că nu este o anomalie daca acțiunile petrecute pe sistem sunt complet normale.

In figura 3.2.3 se regăsește partea din contextul aplicației în care utilizatorul poate lua decizii. O data apăsate butoanele “KILL” vor memora pid-urile ce vor fi oprite și apoi vor dispărea. Butonul “THIS IS NOT ANOMALY” va ignora orice buton “KILL” apăsat și va spune componentei “Decision Handler and Anomaly Provider” să adauge starea sistemului respectiva la stările normale. Butonul “DONE” va colecta procesele oprite de utilizator și va verifica dacă starea respectivă este anomalie sau nu apoi va trimite serverului decizia și detaliile aferente. Butonul “CRITICAL” trimite serverului ca decizia a fost ca sistemul este într-o stare critică. Butonul “DONE” și butonul “CRITICAL” au scopul de a închide cazul acestei anomalii.

3.3 Performanța

Deși proiectul are o structură complex, filtrările datelor ce trebuie prelucrate si salvate și modul în care au fost implementate operațiile afectează foarte puțin performanța sistemului pe care rulează serviciul. Testele de performanță au fost făcute pe o mașină virtuala cu următoarele specificații:

Sistemul de Operare: Ubuntu 16.04.5 cu arhitectura x64 și cu versiunea de kernel 4.15.0-51-generic
Număr Nuclee: 2 cu frecventa 2.7 GHz, 8MB cache
Memoria RAM: 4GB

În imaginile de mai jos pot fi observate statistici din aplicația “System Monitor” din distribuția de Linux Ubuntu. În figura 3.3.1 este reprezentată starea sistemului fără ca serviciul de detecție de anomalii să fie pornit. Se poate observa ca procesoarele sunt folosite la 5% respectiv 3.9% din capacitatea lor. Se mai observă că memoria RAM este ocupată undeva la 30% din capacitate.

În figura 3.3.2 este reprezentată starea sistemului când serviciul de detecție de anomalii rulează în modul de învățare. Având în vedere faptul că intervalul în care se afișează consumul de resurse este de 60 de secunde se poate observa că memoria RAM nu este afectată cu mai mult de 5% și că procesorul, o data la 10 secunde, ajunge să aibă un singur nucleu ce ajunge să ruleze la capacitate de aproximativ 95%. Faptul ca de 6 ori într-un minut se întâmpla acest lucru este din cauza ca perioada în care este setat proiectul să salveze noile stări și să le clusterizeze este la intervalul de 10 secunde.

În figura 3.3.3 este reprezentata starea sistemului când serviciul de detecție de anomalii rulează în modul de detecție. Din punctul de vedere al performanței, diferența dintre acest mod și cel de învățare este ca acesta afectează mult mai puțin nucleul folosit de către proces, folosind între 60% și 80% din capacitatea lui.

Din cele 3 statistici putem trage concluzia că interfața de monitorizare a evenimentelor eBPF și colectarea de informații privind acele evenimente nu afectează performanța sistemului. Performanța este afectată doar în cazul în care rulează modul de învățare sau cel de detecție doar la un anumit interval de timp. Din aceasta cauză este recomandat ca mașina pe care rulează detectorul de anomalii să aibă un nucleu rezervat special pentru acest serviciu.

Figura 3.3.1

Figura 3.3.2

Figura 3.3.3

3.4 Acuratețea detecțiilor

Într-o perioadă de testare de o săptămâna, pe mașina de test menționata în capitolul 3.3, sistemul de detecție de anomalii a generat doar 3 alarme de anomalii care au fost fals pozitive, dar care nu au afectat sistemul din cauza excepțiilor generate automat din perioada de învățare, iar comportamentele diferite testate au generat notificare de anomalie în 100% din situații. Aceste date au fost colectate folosind distanța, aleasa empiric, de 20 dintre un eveniment nou și unul normal. După acest test am adăugat în fișierul de configurare trei modalități de detecție în funcție de distanta maximă ce este considerată normală de la un eveniment nou la unul normal:

- Modul STRICT: cere ca distanța să fie de maxim 10
- Modul NORMAL: cere ca distanța să fie de maxim 20
- Modul LOOSE: cere ca distanța să fie de maxim 60

Concluzii

După cum am precizat si în introducerea lucrării, pe lângă interesul față de atacurile informatice, tehnologiile eBPF mi-au sporit motivația dezvoltării temei „Sistem de Detecții de Anomalii folosind tehnologii eBPF”. După dezvoltarea proiectului pot concluziona că interfața eBPF oferă capabilități de monitorizare a sistemului informatic extrem de folositoare fără de care nu cred că proiectul ar fi atins această performanță și acuratețe cu un efort similar si nu ar fi putut monitoriza atât de multe situații.

Sistemul de detecție implementat a reușit să își demonstreze capabilitățile în diferitele teste personale peste un sistem raspberry pi 3 model B+. Deși în prezentarea acurateței din capitolul 3.4 am demonstrat că sistemul s-a descurcat foarte bine pe testele personale, proiectul încă are nevoie de îmbunătățiri pentru a scuti utilizatorul să trebuiască să configureze proiectul pentru a obține un rezultat cât mai bun. Pentru a rezolva această problemă modul de învățare al proiectului ar putea extrage automat tipurile de evenimente ce ar putea fi interesante dintr-o listă definită de dezvoltator. Un exemplu concret ar putea fi momentul când serverul rulează un serviciu posgresql deoarece sistemul de detecții de anomalii ar putea scana toate serviciile urmând să adauge evenimente specifice acestor servicii, cum ar fi interogările SQL în cazul serviciului postgresql.

O alta îmbunătățire a proiectului ar putea fi o componenta ce asigura execuția neîntreruptă a sistemului de detecții. În stadiul în care este momentan proiectul, acesta ar putea fi închis de către un intrus eșuând ca acesta sa fie prins.

În concluzie, cu toate că acest serviciu poate fi îmbunătățit în diferite moduri, acesta reușește să îndeplinească cerințele propuse fiind implementat cu grijă având performanța și acuratețea ca obiective principale.

Bibliografie

1. **Victoria J. Hodge, Jim Austin.** A Survey of Outlier Detection Methodologies. [Interactiv] http://eprints.whiterose.ac.uk/767/1/hodgevj4.pdf.
2. **Abraham Silberschatz, Greg Gagne, Peter B. Galvin.** *Operating System Concepts.*
3. **ArchLinux.** Audit Framework. [Interactiv] https://wiki.archlinux.org/index.php/Audit_framework.
4. **Redhat.** Documentation. [Interactiv] https://access.redhat.com/documentation/.
5. **lwn.net.** A thorough introduction to eBPF. [Interactiv] https://lwn.net/Articles/740157/.
6. —. An introduction to KProbes. [Interactiv] https://lwn.net/Articles/132196/.
7. **Libfuse.** libfuse. [Interactiv] https://github.com/libfuse/libfuse.
8. **David A. Rusling.** The Virtual File System (VFS). [Interactiv] http://www.science.unitn.it/~fiorella/guidelinux/tlk/node102.html#SECTION001120000000000000000.
9. **Orest Zborowski, Ross Biro, Fred N. van Kempen.** linux/net/socket.c kernel implementation. [Interactiv] http://lxr.linux.no/linux+v3.9.7/net/socket.c#L1775.
10. **Jing Gao, Suny Buffalo.** Clustering. [Interactiv] https://cse.buffalo.edu/~jing/cse601/fa13/materials/clustering_density.pdf.
11. **Robert Love.** *Linux Kernel Development.* 2010.
12. **Richard Fox.** *Linux with Operating System Concepts.*
13. **iovisor.** BCC-toolkit. [Interactiv] https://github.com/iovisor/bcc/.