

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IASI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

LUCRARE DE DIPLOMĂ

Coordonator științific:
Ş.l.dr.ing. Cătălin MIRONEANU

Absolvent:
Anamaria-Larisa PAŞA

Iași, 2024

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IASI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Tehnici de asigurare a continuității serviciilor EDR

LUCRARE DE DIPLOMĂ

Coordonator științific:
Ş.l.dr.ing. Cătălin MIRONEANU

Absolvent:
Anamaria-Larisa PAŞA

Iași, 2024

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII PROIECTULUI DE DIPLOMĂ

Subsemnatul PASA ANAMARIA - LARISA,
legitimat cu ci seria XJ nr. 873478, CNP 6010121330790
autorul lucrării TEHNICI DE ASIGURARE A
CONTINUITĂȚII SERVICIILOR EDR
,

elaborată în vederea susținerii examenului de finalizare a studiilor de licență, programul de studii TI organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea iulie a anului universitar 2023-2024, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 - Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data

1.07.2024

Semnătura



Cuprins

Introducere	1
1 Fundamentarea teoretică și documentarea bibliografică	5
1.1 Domeniul și contextul abordării temei	5
1.2 Tema propusă	5
1.3 Prezentare succintă și comparativă privind realizările actuale pe aceeași temă	6
1.3.1 Cyphon	6
1.3.2 CrowdStrike Falcon Insight	6
1.3.3 Carbon Black	7
1.3.4 Stamus Network	7
1.3.5 Metasploit Framework	7
1.4 Analiza tipurilor de produse existente din perspectiva tehnologiilor folosite pentru implementare	7
1.4.1 Tehnologia IDS	7
1.4.1.1 Instrumente similare	8
1.4.1.2 Instrumentul ales	8
1.4.2 Tehnologia SIEM	9
1.4.2.1 Instrumente similare	9
1.4.2.2 Instrumentul ales	10
1.4.3 Tehnologia <i>Command&Control</i>	10
1.4.3.1 Instrumente similare	11
1.4.3.2 Instrumentul ales	11
1.4.4 Cozi de mesage	11
1.4.4.1 Instrumente similare	11
1.4.4.2 Instrumentul ales	12
1.4.5 Mașini virtuale în <i>cloud</i>	12
1.4.5.1 Instrumente similare	12
1.4.5.2 Instrumentul ales	12
1.5 Elaborarea specificațiilor privind caracteristicile așteptate de la aplicație	13
2 Proiectarea aplicației	15
2.1 Analiza platformei hardware, abordarea soluției	15
2.1.1 Mașini virtuale vs Containerizare	15
2.1.2 Aplicații native vs aplicații independente de platformă	16
2.1.3 Arhitectură monolit vs Servicii	17
2.1.4 Abordarea soluției	17
2.2 Arhitectura soluției	18
2.2.1 Descriere generală	18
2.2.2 Componența SIEM	18
2.2.3 Componența <i>Log Collector</i>	19

2.2.4	Componenta Exchange	20
2.2.5	Componenta <i>Command&Control</i>	20
2.2.6	Componenta <i>Endpoints</i>	21
2.3	Avantajele și dezavantajele metodei alese	22
2.3.1	Avantaje	22
2.3.2	Dezavantaje	23
2.4	Limitele de funcționare	23
3	Implementarea aplicației	25
3.1	Descrierea generală a implementării	25
3.1.1	Abordarea generală a implementării	25
3.1.2	Componenta SIEM	26
3.1.2.1	Descriere generală	26
3.1.2.2	Structura configurării	26
3.1.2.3	Comunicarea cu alte componente	29
3.1.3	Componenta <i>Log Collector</i>	29
3.1.3.1	Descriere generală	29
3.1.3.2	Structura codului	29
3.1.3.3	Comunicarea cu alte componente	33
3.1.4	Componenta <i>Exchange</i>	33
3.1.4.1	Descriere generală	33
3.1.4.2	Structura codului/configurării	34
3.1.4.3	Comunicarea cu alte componente	34
3.1.5	Componenta <i>Command&Control</i>	34
3.1.5.1	Descriere generală	34
3.1.5.2	Structura codului/configurării	35
3.1.5.3	Comunicarea cu alte componente	39
3.1.6	Componenta <i>Endpoints</i>	39
3.1.6.1	Descriere generală	39
3.1.6.2	Structura codului/configurării	39
3.2	Dificultăți întâmpinate și modalități de rezolvare	41
3.2.1	Funcționarea corectă a EC2 cu VPN	41
3.2.2	Alte dificultăți întâmpinate	42
3.3	Idei originale, soluții noi	42
3.3.1	Componenta <i>Log SIEM</i>	42
3.3.1.1	Rol uzual	42
3.3.1.2	Scopul abordării personalizate	42
3.3.2	Componenta <i>Command&Control</i>	43
3.3.2.1	Rol uzual	43
3.3.2.2	Scopul abordării personalizate	43
3.3.3	Componenta <i>Endpoints</i>	44
3.3.3.1	Rol uzual	44
3.3.3.2	Scopul abordării personalizate	44
3.4	Comunicarea cu alte sisteme și salvarea/stocarea informațiilor	44
3.5	Funcționarea sistemului	44
3.5.1	Colectare periodică alerte în componenta <i>Log Collector</i>	44
3.5.2	Reacție agent Wazuh închis	45
3.5.3	Atac Brute Force SSH	45
4	Testarea aplicației și rezultate experimentale	47

4.1	Punerea în funcțiune/lansarea aplicației	47
4.2	Testarea sistemului (hardware/software)	48
4.2.1	Planul urmat în realizarea testării	48
4.2.2	Testare pentru simplificarea formatului alertelor	48
4.2.2.1	Scopul testului	48
4.2.2.2	Modalitatea de testare	48
4.2.2.3	Instrumente folosite	48
4.2.3	Testare pentru prevenția închiderii agentului Wazuh	49
4.2.3.1	Scopul testului	49
4.2.3.2	Modalitatea de testare	49
4.2.3.3	Descrierea cazurilor de test	49
4.2.3.4	Instrumente folosite	49
4.2.4	Testare pentru reacția diferențiată la diferite tipuri de alarme	49
4.2.4.1	Scopul testului	49
4.2.4.2	Modalitatea de testare	49
4.2.4.3	Descrierea cazurilor de test	49
4.2.4.4	Instrumente folosite	50
4.2.5	Testare pentru observarea timpilor de procesare al alertelor	50
4.2.5.1	Scopul testului	50
4.2.5.2	Modalitatea de testare	50
4.2.5.3	Instrumente folosite	50
4.3	Aspecte legate de fiabilitate/securitate	50
4.4	Rezultate experimentale	50
4.4.1	Rezultate pentru simplificarea formatului alertelor	50
4.4.1.1	Prezentare rezultat	50
4.4.1.2	Modalități de îmbunătățire rezultat	52
4.4.2	Rezultate pentru prevenția închiderii agentului Wazuh	52
4.4.2.1	Prezentare rezultat	52
4.4.2.2	Modalități de îmbunătățire rezultat	53
4.4.3	Rezultate pentru reacția diferențiată la diferite tipuri de alarme	53
4.4.3.1	Prezentare rezultat	53
4.4.3.2	Modalități de îmbunătățire rezultat	55
4.4.4	Rezultate pentru observarea timpilor de procesare a alertelor	55
4.4.4.1	Prezentare rezultat	55
4.4.4.2	Modalități de îmbunătățire rezultat	55
4.5	Utilizarea sistemului	55
Concluzii		57
Bibliografie		59
1	Componenta C2 - Fișier consumer_mq.py	63
2	Componenta C2 - Fișier c2.py	64
3	Componenta C2 - Fișier agent_c2.py	65
4	Componenta C2 - Scenariu activare agent Wazuh	66

Tehnici de asigurare a continuității serviciilor EDR

Anamaria-Larisa PASĂ

Rezumat

Sistemele de detecție și preventie a atacurilor cibernetice (IDS/IPS) au ca scop monitorizarea fluxurilor de date care intră și ies într-un/dintr-un sistem în vederea detectării și alertării comportamentelor suspecte (rolul IDS) sau în scopul detectării, alertării și blocării directe a atacurilor (rolul IPS). Un IDS de tip NIDS monitorizează traficul de rețea și vizează identificarea atacurilor care fac parte din tipul *System Intrusion*, precum *Backdoor Installation*, *Denial of Service*, *Trojan Horse Attacks*. NIDS nu pot asigura de sine stătător protecția sistemelor conectate la rețea. În primul rand, NIDS se ocupă doar de detectarea și generarea de alerți, necesitând integrarea sa în cadrul unei soluții mai complexe care asigură reacția la alerți. În al doilea rând, NIDS pot rata atacuri de tip *Social Engineering* care urmăresc influențarea și exploatarea vulnerabilităților umane, dar și atacuri de tip *Zero-Day* prin prisma faptului că nu există încă reguli create pentru atacurile noi.

Ca o abordare complementară a NIDS sunt HIDS, iar împreună fac parte dintr-o soluție de securitate, mai amplă, de tip SIEM. Un sistem de Securitate a Informației și Managementul Evenimentelor include funcționalitățile HIDS, adică poate detecta activități malicioase la nivel de sistem, precum modificări de fișiere și nu este afectat de traficul criptat, deoarece operează pe date deja decriptate pe gazdă. Datorită multitudinii de date care sunt prelucrate în cadrul unui SIEM, detectarea și reacția automată la atacuri reprezintă o rezolvare raportată la necesitățile actuale. Astfel, SIEM permite integrarea IDS pentru a acoperi o plajă mai largă de atacuri,

Arhitectura propusă în această lucrare este modulară și are la bază fluxuri separate de colectare a activității sistemelor protejate față de răspunsul la potențialele alarme. Fluxurile separate sunt materializate prin crearea de canale distințe de comunicare de tip server-client. Solutia tehnică de implementare are la bază tehnologii de detecție a intruziunilor, precum server și agenți Wazuh cu rol de HIDS și NIDS Suricata, integrate într-un sistem coherent de monitorizare și analiză a evenimentelor de securitate.

Gestionarea și coordonarea reacțiilor potrivite comportamentelor suspecte detectate de NIDS, corelate cu HIDS, se realizează printr-un server centralizat de comandă și control (C2). Agenții reactivi C2 sunt instalati pe aceleași sisteme pe care sunt configurați și agenții proactivi SIEM. Agentul SIEM preia alertele generate de NIDS, le transferă către server SIEM. Serverul C2 preia, după o serie de procesări, alertele și transmite reacții potrivite pentru acestea către agentul C2. Agentul C2 execută instrucțiunile pentru oprirea comportamentelor suspecte.

Proiectul are la bază regulile unui sistem EDR prin care se iau acțiuni imediate la diverse comportamente anormale detectate pe sistemele monitorizate. Beneficiarii proiectului propus în această lucrare sunt echipele *Blue Team*, folosind tehnici *Red Team*, precum C2.

Introducere

Motivația alegerii temei

Este bine cunoscut faptul că rețea de comunicări globală, internetul, a devenit un instrument central în viața oamenilor, atât la nivel individual, cât și în cadrul companiilor și instituțiilor, reprezentând platforma principală de transfer, de educație și de procesare a datelor. Progresele accelerate în domeniul tehnologiei au facilitat o creștere substanțială a accesibilității la dispozitive conectate la internet, stimulând astfel fluxul de informații.

Odată cu această evoluție, s-a observat și o tendință crescătoare a atacurilor cibernetice care vizează să exploateze vulnerabilitățile specifice factorului uman. Așa cum este analizat în studiul [1], se evidențiază că printre cele mai frecvent întâlnite slăbiciuni se numără dificultatea de concentrare, procesul de luare a deciziilor și predispoziția de a se angaja în ajutorarea altor persoane. Se accentuează importanța înțelegerii complexe a factorilor psihologici și comportamentali, ceea ce evidențiază că reacția manuală la atacurile cibernetice este mai dificila de realizat. Automatizarea acestei reacții este un pas important în dezvoltarea proiectelor de securitate.

Relevanța temei

Relevanța temei lucrării este dată, de necesitatea eliminării acțiunii umane din procesul de reacție și neutralizare a unui atac, precum și de dezavantajul temporal al analizei manuale a fișierelor de jurnalizare, a datelor și a metricilor pentru fiecare sistem monitorizat în parte. O altă justificare a relevanței acestei teme este ultimul raport Verizon [2] care constată pe baza metricilor VERIS (Vocabulary for Event Recording and Incident Sharing) 4A (Actor, Action, Asset, Attribute) faptul că atacurile de tip *System Intrusion* s-a menținut la cel mai mare procent (~40%) față de *Social Engineering* (~30%), *Miscellaneous Errors* (~20%) în clasificarea incidentelor. Cu alte cuvinte, atacurile vizează preponderent intrarea neautorizată în sisteme cu scopul de a fură date și de a altera fișiere, ceea ce poate conduce la semne precum nefuncționarea aplicațiilor în parametri normali, procese care îngreunează sistemul și modificarea datelor din sistem. Unele dintre aceste semne sunt, de cele mai multe ori, ignorate.

Contextul alegerii temei

În urma analizei contextului curent al securității cibernetice, se observă un mediu care s-a dezvoltat rapid și care încearcă să păstreze propriul nivel de securitate ridicat. Dezvoltarea unui proiect în acest domeniu implică o cunoaștere atentă a nevoilor, a limitărilor și posibilităților tehnologice actuale. Este important de cunoscut și nivelul de dezvoltare al atacatorilor pentru proiectarea unor soluții sustenabile. Se observă o predispoziție a acestora de a împărtăși soluțiile de atac în vederea preluării și îmbunătățirii lor de către alți atacatori.

De exemplu, în anul 2017, un grup numit *Shadow Brokers* a publicat o serie de exploatari furate de la Agenția de Securitate Națională din Statele Unite, printre care și *EternalBlue*, care a fost folosit pentru a răspândi atacul de tip *ransomware*, numit *WannaCry*, detaliat în [3]. Acest context conduce către rezultatele scrise în raportul global CrowdStrike [4] din anul 2024 care evidențiază faptul că atacatorii utilizează tactici noi pentru campaniile de atac, reușind să crească intruziunile în *cloud* și atacurile bazate pe identitate prin intermediul AI¹ generativ.

¹Artificial Intelligence

Obiectivele generale

Cresterea securitatii sistemelor monitorizate reprezinta unul dintre principalele obiective ale lucrării, urmărindu-se eliminarea nevoii de a reacționa manual în cazul suspiciunii prezenței unui atac, aşa cum s-a evidențiat și în secțiunile anterioare.

Asigurarea unui nivel suplimentar de securitate în procesul de proiectare al arhitecturii proiectului este un alt obiectiv al temei lucrării. Se urmărește crearea unei arhitecturi care să ofere o abordare diferențiată în ceea ce privește aspectele principale ale unei astfel de aplicații: colectarea datelor și reacția la informațiile suspecte extrase din date. Din acest motiv, se va studia mecanismul de *backdoor* cu scopul de a-l include în cadrul proiectării.

Un alt obiectiv general al temei lucrării este de a oferi o implementare independentă de platformă astfel încât doar configurațiile specifice fiecărei platforme software să fie elementele care necesită adaptare.

Metodologii și instrumente folosite

Tehnici și metode folosite

S-au utilizat o serie de tehnici de securitate din aria defensivă a acesteia. Prima tehnică este reprezentată de un Sistem de Detectie a Intruziunilor a cărui scop este de a inspecta toate fluxurile de date care intra și ies dintr-un sistem monitorizat pentru a descoperi comportamente suspecte și a genera alerte în consecință. Un IDS este împărțit în 2 categorii: *Network intrusion detection system* care inspecțează tot traficul de rețea al sistemului monitorizat și *Host-based intrusion detection system* care analizează sistemul de configurații și activitatea aplicațiilor la nivel de *host*.

O altă tehnică folosită este cea a unui sistem de Securitate a Informației și Managementul Evenimentelor. Aceasta are un caracter flexibil, putând fi integrat cu alte tehnici precum IDS pentru acoperirea unei plaje mai mari de atacuri. Metoda folosită pentru reacția la atac este utilizarea propriului server de comandă și control. Prin colectarea datelor asigurată de SIEM și reacțiunea la alerte asigurată de serverul de comandă și control va rezulta un sistem de tip *Endpoint Detection and Response*.

Instrumente folosite

Suricata este un IDS de tip NIDS care inspecțează fluxurile de date de rețea din sistemul pe care este instalat. Funcționează pe bază de reguli Snort predefinite, generând alerte și date NSM într-un format standard de tip JSON.

Un alt instrument folosit este Wazuh SIEM, o platformă *open-source* pentru monitorizare și detectare de amenințări. Prin intermediul agentului său instalat pe sistemul monitorizat, acesta preia datele spre a le compara cu modelele proprii de atacuri pentru a genera alerte.

Din nevoie de a asigura existența unui număr mare de sisteme monitorizate, se utilizează instrumentul Amazon *Elastic Compute Cloud* pentru scalabilitatea sa.

Pentru a asigura securitatea mediului de lucru și comunicarea dintre instanțele software, se folosește un VPN, mai exact Wireguard, pentru faptul că elimină necesitatea transferului de date între spațiul utilizator și spațiul nucleului.

Coadă de mesaje reprezintă un alt instrument folosit prin care se asigură lipsa dependenței dintre 2 componente ale arhitecturii.

Structura lucrării

În această lucrare se va detalia modalitatea de proiectare a arhitecturii și implementarea proiectului cu titlul *Tehnici de asigurare a continuității serviciilor EDR*.

În primul capitol se subliniază contextul general al dezvoltării temei lucrării. Se detaliază produsele existente deja pe piață și avantajele utilizării lor. La momentul actual, piața propune o

serie de produse dezvoltate pe tehnologii mature, utilizabile atât în scopurile echipei *Blue Team*, cat și în cele ale echipei *Red Team*. Tot în acest capitol sunt descrise instrumente similare cu cele folosite în implementare pentru a justifica alegerea utilizării lor.

În al doilea capitol este evidențiată arhitectura sistemului alături de componentele sale descrise. Pentru a înțelege mediul de dezvoltare, se explică modalitatea de abordare în ceea ce privește conceptele generale în cadrul unei aplicații: alegerea tipului de virtualizare, optarea pentru o aplicație nativă sau una *cross-platform*, dezvoltarea unui monolit sau a unui proiect distribuit pe servicii. Tot în cadrul discuției de proiectare sunt evidențiate avantajele și dezavantajele alegерilor făcute, realizând o balanță între alegările făcute și alte posibilități de abordare.

În al treilea capitol se descrie modalitatea de implementare a proiectului pe baza alegărilor discutate în capitolul 2. Se explică modalitatea de configurare a mediului de lucru. Ulterior, fiecare componentă a proiectului este descrisă în detaliu, explicând cele mai importante funcționalități ale acesteia. Pe lângă descrierea generală, este de interes structura componentei și modul în care se interconectează cu celelalte elemente din arhitectură. Tot în acest capitol se vor descrie și dificultățile întâmpinate în procesul de implementare, alături de modalitatea în care acestea au fost rezolvate. Ideile originale sunt organizate în funcție de componente: se descrie care ar fi rolul uzuial al unui concept, dar și modul în care acesta a fost utilizat/modificat/adaptat. În cele din urmă, capitolul se încheie cu o prezentare scurtă a modului în care soluția rulează.

Al patrulea capitol este dedicat testării proiectului. Se detaliază ce teste au fost efectuate, ce s-a urmărit în aplicarea lor, dar și modalitatea de testare și instrumentele care s-au utilizat. După detalierea testelor, se pot observa și rezultatele obținute, acestea fiind descrise în detaliu și susținute de imagini, date procesate și preluate din aplicație. În cele din urmă se explică nișele pentru care a fost creată soluția.

Concluziile înglobează toate informațiile importante detaliate pe parcursul lucrării, oferind o perspectivă concisă a ideii proiectului și modalitatea sa de dezvoltare.

Capitolul 1. Fundamentarea teoretică și documentarea bibliografică

1.1. Domeniul și contextul abordării temei

Domeniul securității cibernetice se referă la tehnici și tehnologii care au scopul de a proteja rețelele, programele și informațiile de atacuri și acces ilegal la acestea [5]. O preocupare importantă o reprezintă abordarea eficientă a detecției și reacțiunii la atacuri cibernetice într-o manieră care să asigure îndeplinirea atributelor de securitate a sistemelor informatice și a datelor acestora - integritatea, confidențialitatea și disponibilitatea.

Tema lucrării se încadrează în domeniul securității cibernetice, utilizând tehnologii din diverse arii. Din aria *Network Security*, sunt folosite Sistemele de Detectare a Intruziunilor (IDS) pentru monitorizarea și protejarea rețelelor. Prin *Incident Response* sunt analizate mecanismele de comandă și control (C2) pentru gestionarea eficientă a incidentelor de securitate. De asemenea, se abordează *Security Operations* prin utilizarea Sistemelor de Management al Informațiilor și Evenimentelor de Securitate (SIEM) pentru centralizarea și corelarea datelor de securitate.

Contextul actual al securității cibernetice înglobează: nevoia de a utiliza tehnologii avansate precum inteligența artificială și tehnici de învățare automată în soluțiile propuse, rolul important al factorului uman în menținerea și asigurarea securității și influența politicilor internaționale privind standardizarea practicilor de securitate cibernetică, conform studiului [6]. Ca o validare, în anul 2022 s-au aprobat legi care includ directiva NIS2 (Network and Information Security Directive 2) care impune măsura standardizării cerințelor de securitate la nivelul Uniunii Europene [7]. Factorul uman este, în egală măsură cu organizațiile și domeniul politic, vizat, deoarece atacatorii urmăresc scurgerile de informații sensibile și îngreunarea comunicării. Astfel, tema lucrării este abordată într-un context al dezvoltării continue atât în cazul atacatorilor, cât și al măsurilor de preventie.

1.2. Tema propusă

Tema lucrării este reprezentată de titlul *Tehnici de asigurare a continuității serviciilor EDR* care propune crearea unui sistem de detectare și reacție automată la atacuri folosind fluxuri separate de colectare a activității sistemelor protejate față de răspunsul la potențialele alerte.

Unul dintre principalele obiective ale acestei lucrări este reprezentat de creșterea securității sistemelor monitorizate prin eliminarea tratării și analizei manuale a comportamentelor suspecte și automatizarea acestora. Componenta umană în cadrul reacției la un atac cibernetic sau la identificarea anomaliei poate introduce erori. Un argument sustenabil pentru această abordare îl constituie audierea desfășurată la Comitetul Congresului pentru Energie și Comerț în data de 16 noiembrie 2016, ca urmare a atacului devastator de tip DDoS asupra companiei Dyn care oferă servicii de DNS (DynDNS), din 21 octombrie același an.

Criptologul și expertul în securitate informatică, Bruce Schneier, a evidențiat în cadrul acestei ședințe că „Internetul este un instrument masiv pentru eficientizare, iar acest lucru este valabil și în cazul atacurilor. Internetul permite atacurilor să scaleze la un nivel imposibil altfel.” [8]. Această afirmație subliniază faptul că, odată cu răspândirea IoT², puterea de escaladare a atacurilor cibernetice a atins un nou nivel.

Evoluția tehnologică a facilitat, de asemenea, partajarea cunoștințelor cu potențial malicioș, exemplul notabil fiind publicarea codului sursă al *botnet-ului Mirai*, utilizat în atacul mai sus amintit asupra companiei Dyn [9]. Această audiere a conturat natura colaborativă și distribuită a comunității atacatorilor cibernetici și modul în care aceștia își împărtășesc resursele și expertiza.

Un alt obiectiv al temei propuse este cel de a asigura un strat suplimentar și separat de

²Internet of Things

acțiune în cadrul proiectării soluției în vederea completării rolului tehnicilor folosite. În cazul unui atac de tip System Intrusion, de exemplu, care vizează îngreunarea serviciilor sau oprirea lor, este folosit conceptul de *second layer* ce asigură reacția la atac fără a trezi suspiciuni. Această abordare este des întâlnită în aplicații. De exemplu, în [10] se detaliază modul în care un IDS este completat de un clasificator SVM³: stratul secundar distinge atacurile *Remote2Local* și *User2Root* de comportamentele normale. Prin utilizarea conceptului în aceasta manieră, se asigură obținerea de performanțe superioare în comparație cu alte tehnici existente. Un alt exemplu este prezentat în [11] unde se evidențiază că utilizarea unui strat suplimentar de securitate care utilizează un cod unic îmbunătățește securitatea tranzacțiilor de plată de două ori mai mult prin comparație cu soluțiile existente.

Independența de platformă este un obiectiv vizat în dezvoltarea implementarea arhitecturii, întrucât tehnicele utilizate permit crearea unei soluții generalizabile și personalizabile în funcție de particularitățile fiecărei platforme. Această abordare oferă posibilitatea reutilizării codului și a unor configurații, rămânând doar configurațiile specifice platformei să fie adaptate. După cum se menționează în [12], utilizarea unei aplicații cu arhitectură independentă de platformă amortizează investițiile în privința noilor proiecte prin reutilizarea domeniilor, codului, configurațiilor.

1.3. Prezentare succintă și comparativă privind realizările actuale pe aceeași temă

Opțiunile actuale de protejare împotriva atacurilor cibernetice includ o serie de soluții avansate și tehnologii specializate precum Machine Learning: Windows Defender Advanced Threat Protection, DarckTrace, Cisco AI Network Analytics, IBM QRadar Advisor with Watson și Sophos Intercept X, conform [13]. Aceste soluții integrează, adesea, funcționalități NIDS (Network Intrusion Detection System) și HIDS (Host-based Intrusion Detection System) pentru a monitoriza și a proteja rețelele și dispozitivele împotriva atacurilor.

1.3.1. Cyphon

Este o platformă bazată pe tehnologii *cloud* ce utilizează o arhitectură distribuită pentru gestionarea evenimentelor de securitate, care integrează monitorizarea, detecția și răspunsul la incidente. Cyphon poate fi conectat la diverse surse de date prin intermediul integrărilor predefinite sau a API-urilor personalizate. Datele sunt trimise la serverul central pentru procesare și analiză [14]. Cyphon, cu arhitectura sa, își propune să fie o comandă centrală pentru gestionarea incidentelor, conectând surse diverse de date și oferind o interfață unificată pentru monitorizare și răspuns rapid. Poate integra IDS Suricata prin capabilitățile sale de preluare a alertelor din fișierele de jurnalizare a IDS. Poate integra SIEM Wazuh prin utilizarea Wazuh API.

Platforma Cyphon și soluția proprie reprezintă abordări centrate pe automatizare și prioritizare, urmărind să ofere echipei de securitate o vizionare clară asupra amenințărilor și un cadru pentru acțiuni rapide și eficiente. Cu toate acestea, Cyphon, cu flexibilitatea sa și integrărilor predefinite, poate fi văzut ca un ecosistem mai vast și mai adaptabil, în timp ce proiectul propus se poate mai degrabă încadra ca o soluție mai focalizată și specializată doar pentru anumite tipuri de amenințări.

1.3.2. CrowdStrike Falcon Insight

Soluțiile EDR sunt concepute pentru a identifica și atenua amenințările pe sistemele monitorizate. CrowdStrike Falcon este un produs folosit de organizații care este conceput pentru operații defensive, oferind funcționalități EDR pentru detectie, investigare și răspuns la atacuri în timp real. Identifică în mod automat comportamentul atacatorilor prin utilizarea IOAs⁴ și permite personalizarea sa în privința modului de acțiune. Mapează alertele generate la *MITRE Adversarial Tactics, Techniques and Common Knowledge* pentru a reduce timpul de triere al alertelor. Crow-

³Support vector machine

⁴Indicators of attack

dStrike Falcon utilizează analiza comportamentală și algoritmi de învățare automată în vederea detectării anomalieiilor, conform [15].

1.3.3. Carbon Black

Este una dintre cele mai populare opțiuni în privința produselor de tip EDR datorită abilității sale de a monitoriza orice acțiune a sistemului, precum modificări de registri, flux de date în rețea, conform [16]. Dispune de un centru pentru operații de securitate pentru detectări bazate pe indicatori de compromis. Aceștia din urmă sunt indicatori de reacție care identifică anomalii de rețea. Carbon Black se bazează pe apelurile de tip *callback* ale nucleului și multe dintre funcționalitățile sale sunt stocate în *driver-ul* de filtrare al rețelei. Acest filtru monitorizează și modifică interacțiunile dintre *endpoint-uri*, controlând pachetele care trec prin rețea u de calculatoare.

1.3.4. Stamus Network

Este o companie de securitate cibernetică specializată în detectarea atacurilor la nivel de rețea și asigurarea reacției la acestea. Este cunoscută pentru folosirea avansată a IDS Suricata într-o manieră mult mai complexă decât utilizarea clasică a acestui IDS. În anul 2022, compania a publicat o carte intitulată *The first practical guide for unlocking the potential of Suricata* al cărei scop este, conform lui Éric Leblond, co-fondator al *Stamus Network*, de a oferi un ghid de utilizare a IDS Suricata într-un mod eficient, utilizând toate capabilitățile de care dispune.

1.3.5. Metasploit Framework

Este special proiectat pentru a evita detectarea de către sistemele de securitate precum IDS, IDPS și utilizează metode de împiedicare a analizelor de securitate prin criptare și ștergere permanentă de date. Vine preinstalat pe sistemele de operare folosite pentru *pentest*, precum Kali Linux, Parrot O.S. Acest *framework* utilizează C2 pentru a stabili conexiuni cu sistemele compromise. Articolul [17] se evidențiază încercările de modificare a Metasploit pentru evitarea traficului criptat specific C2 de către sistemele de securitate.

Prin urmare, *framework-urile* și instrumentele detaliate utilizează tehnologii precum cele folosite în dezvoltarea proiectului: EDR, IDS Suricata, Server *Command&Control*, SIEM Wazuh. Acestea le utilizează în diverse contexte, de la măsură principală de detectare a atacurilor, cum este compania Stamus Network, până la folosirea lor pe post de instrument pentru atingerea obiectivelor, cum este Metasploit și utilizarea C2.

1.4. Analiza tipurilor de produse existente din perspectiva tehnologiilor folosite pentru implementare

În prezent există o multitudine de tehnologii de securitate pentru diferite necesități pe care o organizație trebuie să le acopere. Dintre acestea, cele care sunt mai apropiate de scopul temei alese sunt cele de detectare și răspuns la amenințări asupra dispozitivelor conectate în rețea (EDR), care protejează dispozitivele împotriva amenințărilor, oferind o vizibilitate detaliată în timp real asupra activității dispozitivelor și capacitatea de a răspunde într-un timp cât mai scurt la incidente. Alte soluții similare sunt cele de detectie a intruziunilor (IDS), platforme de analiză a evenimentelor de securitate (SIEM) și tehnologii de monitorizare și analiză a traficului de rețea.

1.4.1. Tehnologia IDS

Conform [18], sistemele de detectie a intruziunilor (IDS) inspectează detaliat toate fluxurile de date care intră și ies dintr-un sistem informatic, identificând comportamentele suspecte și generând alerte pentru administratori în vederea aplicării de măsuri corective. Uneori, un IDS poate fi catalogat ca fiind și un sistem de prevenire a intruziunilor (IPS) atunci când este capabil să intervină activ pentru a bloca tentativele de atac, în conformitate cu reguli preconfigurate. Această

funcționalitate este denumită sistem de detectare și prevenire a intruziunilor (IDPS). IDS are un rol pasiv și se limitează la monitorizarea și înregistrarea activităților suspecte, fără a interveni în mod activ pentru a le opri. O altă clasificare este dată de poziționare: IDS care monitorizează o rețea (NIDS) și IDS care monitorizează un sistem gazdă (HIDS). NIDS are capacitatea de a colecta și analiza datele de trafic pe baza mecanismului *wiretapping* - practică de interceptare neautorizată a comunicațiilor. Informațiile monitorizate includ fluxurile de date și alte date relevante pentru rețea.

NIDS identifică tendințe sau semnături care ar putea indica o problemă de securitate, cum ar fi atacurile de tip *denial-of-service*, eventual precedate de scanările de porturi. Prin această analiză a traficului de rețea se efectuează o comparare cu o bază de date de semnături sau modele de atac recunoscute, trimițând alerte sau activând măsuri de protecție pentru a asigura securitatea rețelei [19]. Deoarece majoritatea traficului de internet din prezent este criptat, NGFW (Next-Generation Firewall) oferă suport suplimentar pentru NIDS, furnizându-i spre analiză traficul decriptat. Implementarea NIDS în rețele cu trafic intens poate duce la probleme de performanță, deoarece trebuie să analizeze un volum mare de date în timp real [20]. Problemele de performanță apar atunci când NIDS nu poate ține pasul cu volumul traficului și rata de transfer a datelor. Ca metode de rezolvare a problemei, se pot menționa scalarea orizontală, filtrarea prealabilă a traficului, înregistrarea selectivă. Printre cele mai cunoscute și folosite NIDS sunt Snort și Suricata, conform [18].

1.4.1.1. Instrumente similare

Snort este o platformă *open-source* IDS și este scris în limbajul de programare C. Utilizează un limbaj bazat pe reguli care utilizează semnături, protocoale și metode euristice de detecție a activităților malicioase, precum *denial-of-service*, *brute force*. Poate analiza traficul în timp real și poate fi configurat ca senzor pasiv care monitorizează traficul fără a fi prezent în fluxul principal de procese, conform [21].

Regulile Snort au o sintaxă simplă, permitând crearea unor reguli personalizate. Snort a fost dezvoltat pe o arhitectură *single threaded*, în timp ce Suricata pe *multi threaded*. Snort prezintă o compatibilitate mai largă în privința utilizării pe diverse dispozitive. Snort prezintă performanțe mai mari în medii cu resurse limitate, în timp ce Suricata este adaptată pentru medii cu trafic mare.

Zeek este un IDS hibrid ce detectează atacurile pe baza semnăturilor și a metodelor bazate pe anomalii. Utilizează motorul de evenimente pentru a crea o varietate de evenimente care să indice prezența tranzacțiilor în sistem. Identifică intruziunile monitorizând traficul de rețea pentru a extrage structura la nivel de aplicație. Rulează analize orientate pe eveniment pentru a potrivi traficul cu modele de atacuri, conform [22]. În timp ce Suricata acționează ca un IDS în timp real, Zeek este utilizat ca un analizator pasiv de trafic.

1.4.1.2. Instrumentul ales

Suricata este un IDS proiectat să funcționeze cu seturile de reguli Snort, furnizate de Cisco *Emerging Threats* sau *Emerging Threats Pro*, conform [23]. Suricata are avantajul rulării native *multi-threading*, ceea ce aduce eficiență în medii cu trafic intens. Suricata poate genera date NSM⁵ în formate standard precum JSON (EVE). Are avantajul de a captura doar pachetele care îndeplinesc anumite criterii, fiind o funcționalitate care a fost folosită în implementarea arhitecturii propuse. Astfel, combinând avantajele enunțate anterior cu posibilitatea de a dezvolta reguli personalizate și cu o sintaxă ușor de înțeles și de utilizat, Suricata este ales ca IDS în această lucrare. Arhitectura sa este disponibilă în Figura 1.1.

⁵Network Security Monitoring

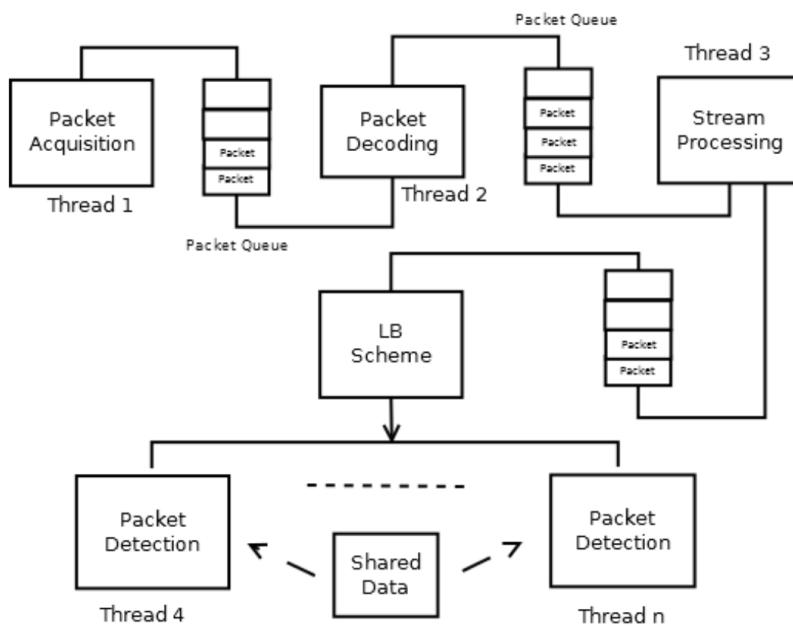


Figura 1.1. Arhitectura Suricata, conform [23]

1.4.2. Tehnologia SIEM

Security Information and Event Management (SIEM) este o tehnologie care combină două funcționalități principale: managementul informațiilor de securitate (SIM) și managementul evenimentelor de securitate (SEM). Această tehnologie colectează, analizează și prezintă informații despre securitatea sistemelor informatici în timp real, ajutând organizațiile să detecteze, să răspundă și să gestioneze incidentele de securitate.

Scopul principal al SIEM este de a oferi o înțelegere centralizată și completă a sistemului informatic protejat în ceea ce privește securitatea, prin colectarea și compararea evenimentelor provenite dintr-o varietate de surse, inclusiv dispozitive de rețea, servere și aplicații. SIEM are atât capacitate de monitorizare, cât și de reacție supervizată sau nesupervizată la atacurile identificate. Orice tip de reacție ar fi implementată într-un SIEM, este absolut necesar ca elementul de decizie să poată comunica cu elementul de execuție de pe *endpoint*. În unele cazuri, atacatorii vizează neutralizarea comunicației cu agenții de pe *endpoint*-uri, moment în care SIEM își pierde capacitatea de acțiune.

1.4.2.1. Instrumente similare

Splunk este un alt instrument SIEM care se remarcă prin scalabilitatea și capacitatea de a gestiona volume mari de date, în timp ce Wazuh se concentrează pe detectarea amenințărilor la nivelul sistemelor monitorizate. Splunk este un produs comercial, pe când Wazuh este *open-source*, ceea ce a constituit un argument în adoptarea sa în implementarea arhitecturii propuse. Deși Splunk oferă o soluție SIEM comprehensivă, Wazuh este un sistem orientat IDS.

IBM QRadar și Wazuh sunt ambele soluții puternice de securitate cibernetică, însă ele diferă semnificativ în domeniul lor de acțiune. QRadar oferă o gamă largă de funcționalități de securitate, inclusiv analiza comportamentului utilizatorilor, scalabilitatea și capacitatele robuste de raportare, făcând din el o alegere potrivită pentru organizațiile mari cu medii IT complexe. Pe de altă parte, Wazuh valorifică funcționalitatea OSSEC⁶, oferind analiză în timp real a alertelor de securitate generate de datele de jurnal și evenimentele de sistem.

⁶Open Source HIDS SECurity

1.4.2.2. Instrumentul ales

Se consideră că un instrument NIDS nu este suficient pentru monitorizarea completă a întregului sistem, acesta având limitări. Altfel, pentru a completa și extinde capacitatele de monitorizare și detecție a amenințărilor, este necesară integrarea IDS Suricata, în cadrul unui instrument mai amplu, de tip SIEM. În acest context, se alege ca tehnologie platforma Wazuh, integrarea acesteia cu Suricata fiind prezentată în Figura 1.2:

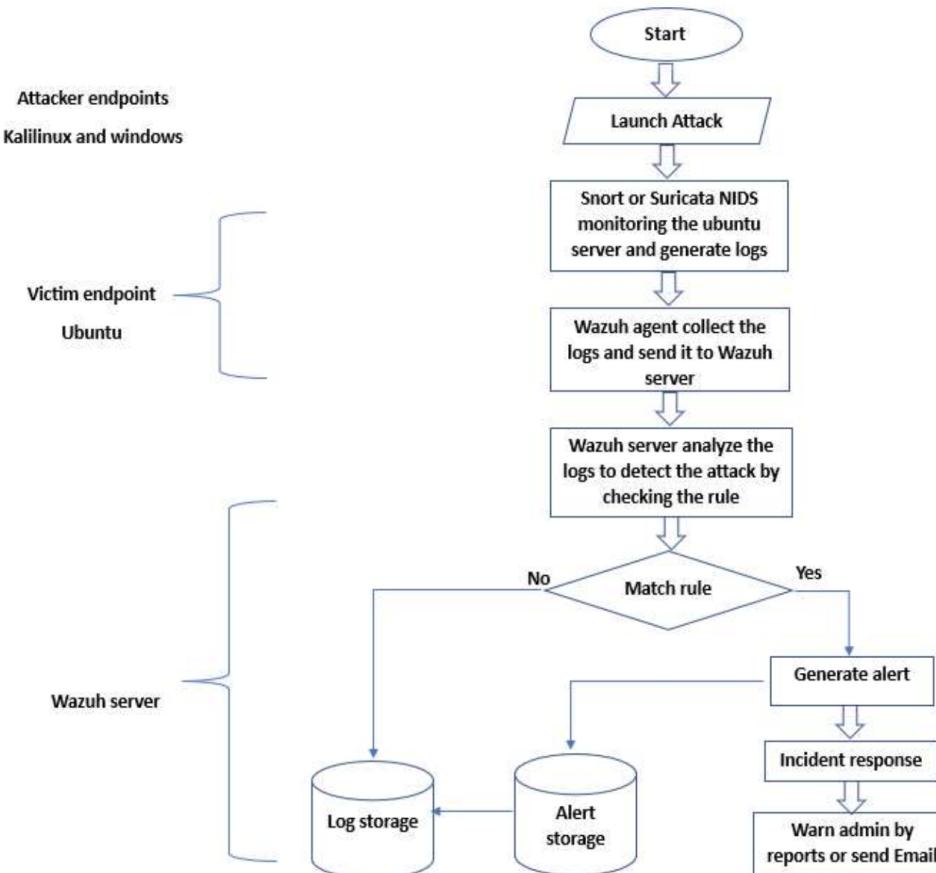


Figura 1.2. Wazuh integrat cu IDS, conform [19]

Wazuh SIEM este o platformă *open-source* pentru monitorizarea și detectarea amenințărilor [19]. Concepță ca un instrument de securitate, această soluție oferă o gamă variată de funcționalități, inclusiv colectarea de jurnale, detectarea amenințărilor, gestionarea incidentelor, monitorizarea integrității fișierelor și altele. Prin centralizarea managementului jurnalelor și utilizarea unor mecanisme avansate de analiză, Wazuh SIEM permite identificarea amenințărilor și a anomaliei într-un mod optim, oferind organizațiilor flexibilitatea de a adapta și modifica caracteristicile SIEM în conformitate cu cerințele specifice ale mediului lor.

În arhitectura propusă, agenții Wazuh sunt instalati și configurați pe sistemele monitorizate pentru a colecta jurnalele și alertele generate de IDS Suricata, trimițându-le apoi serverului Wazuh pentru analiză, detectare și gestionare a incidentelor. Comunicarea cu serverul Wazuh se realizează prin intermediul unui canal securizat și autentificat, asigurând transferul în siguranță al datelor. În aceeași manieră, IDS Suricata este instalat și funcțional pe *endpoints* pentru a se integra cu Wazuh, efectuând monitorizarea în timp real a traficului de rețea și identificând semnături de atacuri cunoscute, bazându-se pe seturi de reguli predefinite sau personalizate.

1.4.3. Tehnologia Command&Control

Conform [24], un server de comandă și control este un sistem informatic controlat de către un atacator, utilizat în scopul transmiterii de instrucțiuni către sistemele compromise și pentru 10

recepționarea datelor furate din cadrul unei rețele țintă. Rulează în mod obișnuit pe o mașină dedicată care este separată de restul rețelei, facilitând gestionarea și monitorizarea centrală a rețelei de calculatoare de către atacator.

Există mai multe tipuri de arhitecturi ale unui astfel de sistem, însă cea mai întâlnită este cea centralizată în care există un singur server care acționează ca punct central de control. Un astfel de server are mai multe funcționalități: poate fi dispersat în diverse locații și găzduit pe servere anonime. Odată instalat pe sistemele vizate, agenții stabilesc conexiuni de tip *callback* inițiate prin protocoale de rețea standard, precum HTTP, DNS. Din acest moment, C2 pot trimite comenzi directe ce sunt executate de către sistemele aflate sub control.

Servelele C2 asigură stabilirea mecanismelor de persistență pe dispozitive, prin faptul că agentul rămâne activ și ascuns. Aceste mecanisme pot include intrări de *registry* (în cazul sistemelor Windows), sarcini programate sau instalări de servicii. În ansamblu, funcționalitățile unui server C2 sunt orientate către executarea unor acțiuni dăunătoare asupra sistemelor în care agenții sunt configurați.

1.4.3.1. Instrumente similare

Cobalt Strike este un *framework* de comandă și control care a fost utilizat la scară largă în cadrul atacurilor în ultimii ani datorită capacitatei sale de a stabili canale de comunicare între victime și sistemul atacatorului. Este un instrument care abordează comportament imprevizibil și este greu de detectat de către sistemele de securitate: traficul C&C este criptat și poate imita trafic legitim care folosește protocoale clasice de comunicare, precum HTTP, după cum este detaliat în [25].

Silver este un alt instrument de tip comandă și control proiectat pentru a asigura capabilități de gestionare și control *remote* al sistemelor. Este folosit de echipele *Red Team*, acestea putând stabili canale de comunicare prin diferite protocoale precum HTTP, DNS sau Wireguard cu un *endpoint* țintă.

1.4.3.2. Instrumentul ales

Pentru componenta *Command&Control* detaliată în Capitolul 2 se va implementa propriul server de acest tip. Scopul este de a dezvolta un program C2 care să dețină funcționalitățile de bază ale unui server *Command&Control*: persistență pe sistemele monitorizate, colectarea de date, distribuirea de sarcini, controlul agenților instalati pe sisteme.

1.4.4. Cozi de mesaje

Sistemele bazate pe cozi de mesaje stochează date și permit diferitelor aplicații să se conecteze la cozile existente și să publice sau să consume date. Sistemul de mesaje distribuite este format din mai multe mașini de lucru care lucrează împreună pentru a furniza serviciul de mesaje. Entitatea care publică mesajele este denumită producător, iar entitatea care preia mesajele din sistem se numește consumator. Mesajele sunt grupate în cozi.

1.4.4.1. Instrumente similare

Apache Kafka este un sistem distribuit de mesagerie care se caracterizează prin fiabilitate și scalabilitate. Kafka furnizează API-uri de producător și consumator utilizate pentru alimentarea și colectarea de mesaje către/de la subiecte, conform [26]. RabbitMQ, unul dintre primii și cei mai cunoscuți intermediari de mesaje pentru microservicii, este construit pentru a livra mesaje prin metoda *point-to-point*. Apache Kafka, pe de altă parte, a fost dezvoltat pentru a face față unui volum mare de date cu procesare de joasă latență. Kafka este optimizat pentru a procesa datele de *streaming* în timp real, însă folosirea lui corespunzătoare este mai complexă.

Redis poate fi folosit ca o coadă de mesaje *in-memory*. Acesta lucrează cu perechi cheie-

valoare. Scrierea în memorie se face în mod *monothread* pentru a izola scrierea și a evita pierderea datelor, conform [27]. Folosește tranzacții pentru a evita inconsistența datelor. Redis poate stoca tabele, liste, astfel încât poate asigura transportul unor date complexe. Redis este un alt intermediar de mesaje care servește și ca opțiune de stocare de date în memorie. Deși poate trimite până la un milion de mesaje pe secundă, Redis nu oferă persistența datelor, ceea ce înseamnă că datele pot fi pierdute în cazul întreruperilor neașteptate, aspect care nu este permis într-o soluție de securitate.

1.4.4.2. Instrumentul ales

Instrumentul ales este RabbitMQ, o platformă pentru trimitera și primirea mesajelor *open-source* [28]. RabbitMQ oferă o implementare mai scalabilă și eficientă a AMQP printr-un mecanism eficient de recunoaștere a producătorului. RabbitMQ utilizează, în principal, un algoritm de tip FIFO pentru transmiterea mesajelor între producător și consumator. Proprietățile obligatorii care definesc comportamentul cozii sunt: numele cozii, durata existenței mesajului, ștergerea automată, exclusivitatea și argumentele. Datele rămân într-o coadă până când sunt șterse sau coada este setată în modul automat pentru ștergere.

1.4.5. Mașini virtuale în cloud

O mașină virtuală *cloud* este versiunea digitală a unui computer fizic care poate funcționa în *cloud*. La fel ca o mașină fizică, aceasta poate rula un sistem de operare, stoca date, se poate conecta la rețele și poate îndeplini toate celelalte funcții de calcul. Mai multe mașini virtuale pot partaja o singură gazdă, adică pot funcționa pe aceeași mașină fizică. Acest lucru crește eficiența utilizării mașinii fizice și permite mașinilor virtuale să nu țină cont de mediul lor fizic. Numărul de mașini virtuale pe o singură gazdă este limitat de resursele mașinii fizice.

1.4.5.1. Instrumente similare

Mașinile virtuale oferite de Microsoft Azure sunt scalabile și permit utilizarea lor în implementarea diverselor proiecte care variază din punctul de vedere al complexității. Sunt ideale pentru testare și dezvoltare de programe, potrivite pentru un nivel mediu de trafic web dintre servere. Permit crearea și managementul unui grup de mașini virtuale identice prin clonarea lor și modificarea în funcție de preferințe. Dispun de o serie de servicii ce pot fi integrate, precum *Azure Hybrid Benefit* care permite utilizarea de licențe Windows. De asemenea, serviciul de mașini virtuale oferit de Microsoft Azure dispune și de recuperare în cazul compromiterii acestora prin intermediul *Azure Site Recovery*.

În aceeași manieră, mașinile virtuale asigurate de *Google Cloud Platform* sunt potrivite pentru o varietate de cazuri datorită capabilității de personalizare a acestora.

1.4.5.2. Instrumentul ales

Pentru crearea unui prototip al proiectului, este necesar un număr considerabil de mașini virtuale distincte. Soluția poate fi implementată local sau în *cloud*, fiind aleasă utilizarea Amazon Elastic Compute Cloud (EC2), care este un serviciu web care oferă capacitate de calcul redimensionabilă în *cloud* [29]. O instanță EC2 este un server virtual care poate fi folosit pentru a rula aplicații în AWS, caracteristicile instanței putând fi personalizate în funcție de buget și cerințe specifice. Alocarea resurselor la cerere este posibilă în modelele de calcul în *cloud* datorită capacitatii lor flexibile și duce la soluții optime din punct de vedere al costurilor și al energiei. Utilizarea unui sistem de *cloud computing* reduce semnificativ costul de configurare a unei infrastructuri software, performanțele variind în funcție de timp, tipul de operații efectuate, dimensiunea și tipul de fișier.

1.5. Elaborarea specificațiilor privind caracteristicile așteptate de la aplicație.

Întrucât este un proiect de o complexitate ridicată prin raportarea la tehnologiile folosite și metoda de abordare a problemei, rezultatul așteptat ar trebui să fie reprezentat de identificarea corectă a unui tip de atac și reacția automată de neutralizare a acestuia. De asemenea, o altă caracteristică este cea de asigurare a comunicării agentilor prin două canale de comunicare distințe: agentul Wazuh să comunice cu manager Wazuh printr-un canal, iar agentul C2 să comunice cu serverul său printr-un alt canal. Pentru a păstra soluția proiectului funcțională, este importantă tratarea cazului în care agentul Wazuh poate fi neutralizat, aceasta constituind o a treia caracteristică așteptată de la aplicație.

Capitolul 2. Proiectarea aplicației

2.1. Analiza platformei hardware, abordarea soluției

Conceptul de ”performanță a sistemului de calcul” reprezintă abilitatea unui sistem de calcul, prin hardware și software, de a efectua sarcini specifice, precum stocare, procesare video, rulare de programe, conform [30]. Contextul hardware în cadrul căruia va fi dezvoltată și implementată tema lucrării este unul hibrid, cuprindând atât computerul personal, cât și mașini virtuale pentru simularea existenței mai multor sisteme distincte. În ceea ce privește mașinile virtuale folosite, vor fi descrise două variante folosite: mașini virtuale locale și mașini virtuale în *cloud*.

Pentru mașinile virtuale în *cloud* se vor folosi instanțele EC2⁷ disponibile de la AWS⁸, cu sistem de operare Ubuntu Server 22.04 LTS⁹. Aceste specificații indică faptul că instanța este compatibilă cu ambele arhitecturi, x86 și Arm, și suportă rețele de înaltă performanță datorită ENA¹⁰. De asemenea, utilizarea EBS¹¹ pentru dispozitivul root oferă performanță și flexibilitate în gestionarea stocării.

În ceea ce privește mașinile virtuale locale, acestea au nevoie de o serie de specificații minime pentru a putea rula în parametri normali: configurarea cu 4 procesoare asigură funcționarea optimă pentru sarcini uzuale, cum ar fi monitorizarea sistemului și preluarea evenimentelor de securitate în timp real. Memoria RAM de 8 GB este adekvată pentru sarcini simultane, însă destul de simpliste. În cazul nevoii de configurare a unui server de monitorizare a unui număr mare de agenți, aceasta ar putea deveni insuficientă. În privința comunicării în rețea, se poate folosi configurarea *NAT Network* alături de configurarea unui client VPN¹² care permite accesul din mediul extern către mașina virtuală. În acest mod se asigură actualizările și comunicările constante între programe de tip server-client.

Există o multitudine de opțiuni de dezvoltare a temei lucrării, de la alegerea tipului de aplicație până la distribuirea acestora pe platforme hardware diferite sau înglobarea într-un singur sistem de calcul. În continuare vor fi evidențiate aceste opțiuni spre găsirea unei abordări optime.

2.1.1. Mașini virtuale vs Containerizare

Mașina virtuală este o implementare software a unui sistem de calcul care execută programe la fel ca o mașină fizică [31]. Prin rularea unui număr redus de servere, metoda *Virtual machine consolidation* poate reduce puterea de consum și să îmbunătățească utilizarea resurselor [32], rolurile sale variind de la sisteme de operare și limbaje de programare până la arhitecturi de procesoare.

Containerizarea sau virtualizarea bazată pe containere este o tehnică diferită de virtualizare, unde este creat un mediu izolat similar unei mașini virtuale, dar fără a simula *hardware-ul* virtual, conform [33]. Un container este un mediu izolat unde o aplicație rulează într-un mediu securizat fără a afecta sau impacta restul sistemului. Mai exact, containerizarea poate fi considerată o virtualizare la nivel de sistem de operare, deoarece containerele sunt create în spațiul utilizatorului, deasupra nucleului sistemului de operare, utilizând mai puține resurse decât mașinile virtuale. Deși oferă avantaje, această metodă nu este potrivită pentru tema lucrării, fiind nevoie de modificarea configurațiilor tehnologii utilizate în mod constant.

⁷Elastic Compute Cloud

⁸Amazon Web Services

⁹Long-Term Support

¹⁰Elastic Network Adapter

¹¹Elastic Block Store

¹²Virtual Private Network

Avantaje

Avantajul mașinilor virtuale este faptul că acestea salvează resurse hardware și îmbunătătesc costurile de energie, permitând utilizatorilor să acceseze o multitudine de medii distințe, fără a plăti multiple mașini fizice. Astfel, se realizează o izolare completă a aplicațiilor, îmbunătățind robustețea sistemului și versatilitatea platformei [34].

În privința containerelor, avantajul principal este ușurința de a încapsula, implementa și izola aplicații prin prisma flexibilității în partajarea resurselor [35]. O imagine Docker, de exemplu, elimină nevoie de a instala sistemul de operare și toate programele software necesare pe fiecare mașină de calcul.

Dezavantaje

În privința dezavantajelor, atât în cazul mașinilor virtuale locale, cât și în a celor în *cloud*, acestea necesită resurse pentru mențenanță și funcționarea optimă, ceea ce implică posibile costuri adiționale sau configurații hardware suplimentare. Deși mașinile virtuale în *cloud* pot partaja un set de hardware, resursele dintr-o mașină virtuală nu pot fi transferate cu ușurință către o alta [35].

Un dezavantaj al containerelor este izolarea mai puțin strictă în comparație cu mașinile virtuale. De asemenea, pentru a rula o imagine Docker este necesară configurația acestei platforme în prealabil.

2.1.2. Aplicații native vs aplicații independente de platformă

Aplicațiile native sunt dezvoltate utilizând limbaje de programare specifice diverselor sisteme de operare. Dezvoltarea mobilă nativă necesită nu doar SDK¹³ specifice sistemului de operare, ci și un mediu de dezvoltare integrat (IDE¹⁴).

Aplicațiile independente de platformă sunt dezvoltate dintr-un singur cod de bază, acestea funcționând pe orice sistem de operare. Aceste aplicații pot fi personalizate pentru a utiliza elemente de UI¹⁵ spre a crea impresia de aplicație nativă.

Avantaje

Aplicațiile native oferă o experiență de utilizare mai placută și poată valorifica toate funcționalitățile sistemelor pe care acestea rulează, asigurând și creșterea securității mediului în care rulează.

Spre deosebire de aplicațiile native, cele *cross-platform* au costuri scăzute, asigurând posibilitatea de reutilizare a codului. Timpul de dezvoltare și implementare este semnificativ mai mic.

Dezavantaje

Deși experiența utilizatorului este mai placută în cadrul unei aplicații native, aceasta este mult mai costisitoare, fiind necesară dezvoltarea de aplicații separate pentru fiecare platformă. Sunt necesare echipe separate care lucrează la aceeași aplicație, însă fără a avea posibilitatea de a reutiliza cod și a îmbunătăți într-un mod sau altul timpul de lucru.

Aplicațiile *cross-platform* au și o serie de dezavantaje: integrarea dificilă în unele contexte, performanța mai scăzută față de aplicațiile native.

¹³Software development kit

¹⁴Integrated development environment

¹⁵User Interface

2.1.3. Arhitectură monolit vs Servicii

Studiul [36] pune în balanță două tipuri de arhitecturi ale aplicațiilor: monolită și distribuită, explicând detaliat rolul lor, avantajele și dezavantajele. Arhitectura monolită este o abordare clasică în care aplicația este construită într-un singur cod de bază care include funcționalitățile mai multor servicii. Serviciile în acest context nu sunt independent executabile. Din perspectiva sistemului de operare, o aplicație monolică rulează ca un singur proces în mediul serverului de aplicații.

Arhitectura bazată pe servicii, pe de altă parte, constă din programe distințe, fiecare încorporând propria sa logică de funcționare și propria bază de date, în cele mai multe cazuri. Serviciile comunică între ele folosind protocoale de comunicare facil de utilizat, precum HTTP¹⁶ și standardul REST¹⁷ sau protocoale de mesagerie, precum AMQP¹⁸.

Avantaje

Unul dintre cele mai importante avantaje ale arhitecturii monolitice este simplitatea sa, prin comparație cu aplicațiile distribuite. Toate datele sunt reținute într-o singură bază de date și este eliminată nevoia de sincronizare între acestea. Toate comunicările interne sunt realizate prin mecanisme intra-proces, această arhitectură fiind o alegere naturală în prima fază de dezvoltare.

Avantajul folosirii arhitecturii bazate pe servicii într-o aplicație este reprezentat de proiectarea sa slab cuplată care conferă toleranță mai ridicată la defecte, prin comparație cu monolitul. Aplicațiile de acest tip pot fi oricând dezvoltate fără a necesita schimbarea codului existent.

Dezavantaje

Deși este preferată prin ușurința dezvoltării sale, arhitectura monolită implică și o serie de dezavantaje care apar odată cu creșterea complexității aplicațiilor. Printre acestea se enumeră: nevoie de modificări codului în cazul unei actualizări, generarea de erori în cascadă la schimbarea unor funcționalități sau transformarea într-un sistem care nu mai este rentabil din punct de vedere al costurilor.

De asemenea, arhitectura bazată pe servicii prezintă câteva dezavantaje prin nevoie de dezvoltare continuă, scalarea periodică sau monitorizarea unui întreg sistem de servicii și comunicarea dintre ele spre buna funcționare a aplicației.

2.1.4. Abordarea soluției

În urma unei analize detaliate a contextului hardware actual și a opțiunilor software disponibile pentru proiectare, s-a concluzionat că tema lucrării va fi implementată prin adoptarea unui mediul hardware hibrid. Se va prioritiza utilizarea mașinilor virtuale în detrimentul containerelor, iar soluția va fi concepută pentru a fi compatibilă cu mai multe platforme, integrându-se într-o arhitectură bazată pe servicii.

Mediul hibrid este reprezentat de utilizarea propriului dispozitiv computațional pentru rulearea programelor implementate pentru accesul facil la acestea în vederea modificării lor constante și de mașinile virtuale în *cloud* pentru a evita consumul de resurse locale în vederea asigurării existenței mai multor sisteme pentru monitorizare. Având în vedere nevoie de configurare constantă a mediului de lucru astfel încât să cuprindă tehnologiile dorite și de a asigura o comunicare facilă între sistemele de calcul, s-a optat pentru mașina virtuală în detrimentul unui container.

Independentă de platformă este un obiectiv al implementării soluției, întrucât se folosesc tehnici și tehnologii care nu depind de sistemul de operare, ceea ce conduce către posibilitatea dezvoltării unui sistem generalizabil și scalabil. Arhitectura pentru care se optează este cea bazată

¹⁶Hypertext Transfer Protocol

¹⁷Representational State Transfer

¹⁸Advanced Message Queuing Protocol

pe servicii care utilizează comunicarea prin AMQP, atât din motive de asigurare a scalabilității, cat și a securității pe care o oferă soluția în sistemele monitorizate.

2.2. Arhitectura soluției

2.2.1. Descriere generală

Arhitectura proiectului se încadrează în caracteristicile unui sistem distribuit de tip P2P¹⁹, fiind o arhitectură hibridă care îmbină concepțele de centralizare și descentralizare, conform explicațiilor prezentate în [37]. Conceptul de *Super Node* reprezintă un pilon al acestei arhitecturi, având rolul serverului dintr-o arhitectură centralizată. În cadrul unei arhitecturi distribuite, aceste *Super Nodes* acționează ca noduri de coordonare și control, asigurând distribuția optimă a sarcinilor și a resurselor în întregul sistem. Prin separarea funcționalităților în noduri distințe, arhitectura permite o gestionare modulară a resurselor, fiecare componentă a sistemului fiind proiectată să funcționeze independent. Prin urmare, similar cu programele responsabile de monitorizare și reacție la alarme, toate elementele din arhitectură acționează ca noduri specializate, fiecare având un rol distinct în asigurarea securității și integrității sistemului în ansamblu.

Proiectul se concentrează pe componente specializate, fiecare cu roluri distințe și bine definite (Figura 2.1):

- *Super Node Log Collector*
- *Super Node Command&Control*
- *Nodes Monitorized Endpoints*
- *Super Node Server SIEM*

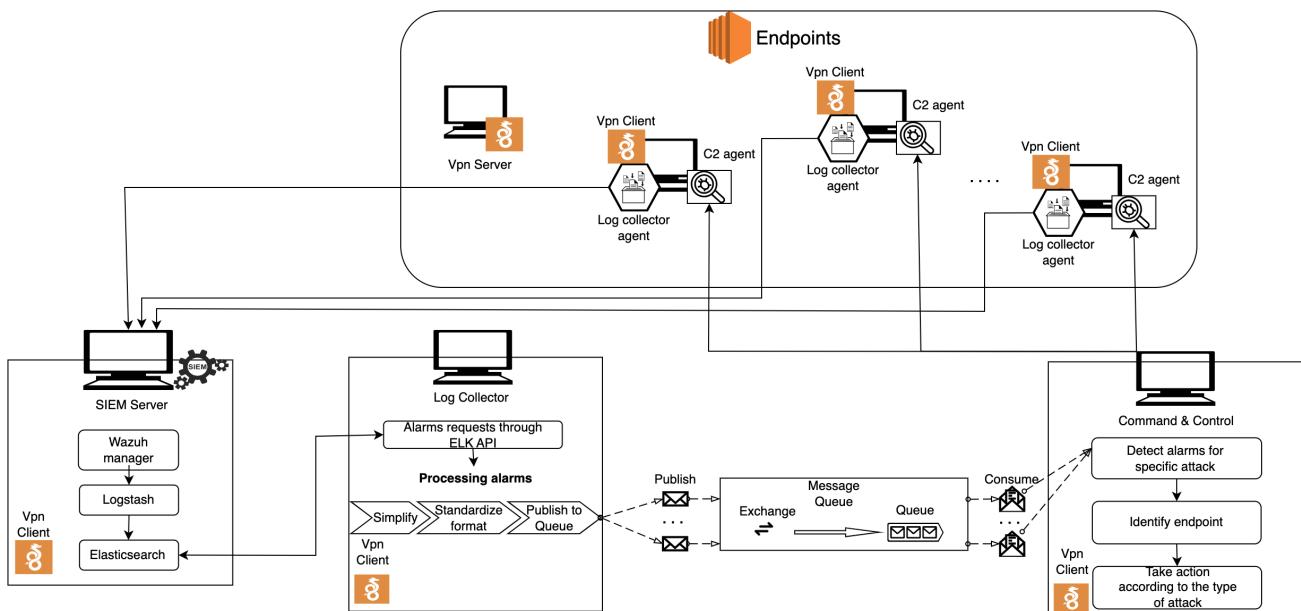


Figura 2.1. Arhitectura hibridă a soluției propuse, de tip P2P

2.2.2. Componența SIEM

Componența SIEM este cea care gestionează toate funcționalitățile care realizează monitorizarea, analiza și colectarea alertelor prin intermediul agentilor instalati pe sistemele monitorizate. Conform analizei din Capitolul 1 unde s-au detaliat mai multe opțiuni pentru sistemul SIEM, s-a concluzionat că instrumentul Wazuh va servi drept tehnologie SIEM datorită ușurinței cu care poate fi utilizat și configurat. O caracteristică importantă a acestuia este faptul că poate integra

¹⁹Point-to-Point

concepte multiple pentru extinderea capabilităților sale. Prin urmare, serverul Wazuh este configurat pentru a utiliza o componentă de colectare a log-urilor (asigurată prin motorul de procesare a datelor *server-side* Logstash) și o componentă de procesare a log-urilor (asigurată de motorul de căutare și analiză Elasticsearch) pentru a facilita captarea alarmelor din sistemele monitorizate (Figura 2.2).

Mai exact, Wazuh Manager primește datele sau alertele generate de Suricata de la agentii, le procesează și generează alerte pe baza regulilor de securitate predefinite, după cum este menționat în articolul [38]. Ulterior, alertele și datele adiționale sunt trimise către filtrul intermediu, Logstash, pentru filtrare, conversie și adăugiri. Logstash inserează apoi alertele într-un cluster Elasticsearch, motorul distribuit de căutare și analiză din centrul stivei ELK, detaliată în articolul [39].

Prin integrarea Wazuh cu stiva ELK (Elasticsearch, Logstash, and Kibana) se asigură un flux clar, complex și sigur de preluare a datelor și alertelor pentru accesarea acestora atât prin intermediul interfeței grafice, cât și prin API Elasticsearch. Aceasta oferă posibilitatea de a accesa indexul Wazuh și de a prelua alarmele într-un mod programatic, abordând o manieră flexibilă, fără a depinde de analiza manuală a alarmelor din *dashboard*-ul Wazuh.

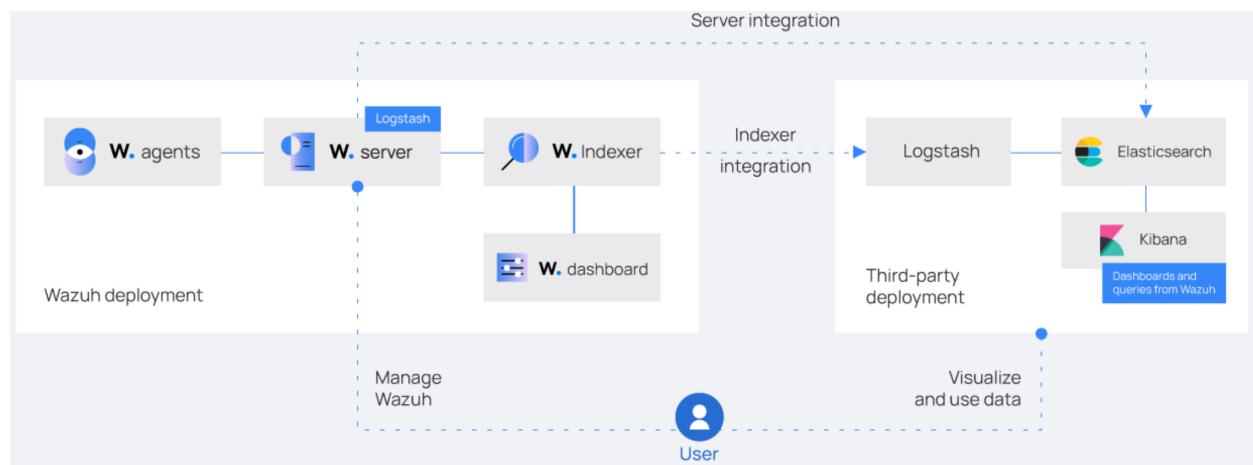


Figura 2.2. Integrare Wazuh și ELK, conform documentației Wazuh [40]

2.2.3. Componenta Log Collector

Conceptul de *logging* reprezintă procesul de colectare a înregistrărilor de evenimente în jurnale, ca de pildă stocarea acestora într-un fișier text sau în baze de date, conform relațiilor definite în MITRE® Common Event Expression [41]. Cu cât sistemele devin mai complexe, cu atât volumul informației captate este mai mare și solicitant pentru sistem.

Metoda tradițională de analiză a jurnalelor, care se bazează în mare măsură pe verificarea manuală, a devenit o sarcină consumatoare de timp și predispusă la erori umane, fiind înlocuită de instrumente care automatizează aceste sarcini. Prin urmare, componenta *Log Collector* din arhitectura prezentă (Figura 2.1) elimină procesul manual de colectare a alertelor, accesând informațiile gata structurate prin intermediul API ELK configurația anterior. Preluarea se realizează periodic, la intervale stabilite în prealabil.

Scopul acestei componente este de a realiza o procesare personalizată a alertelor, în funcție de particularitățile fiecărui context de utilizare (simplificare, clasificare cu metode AI²⁰). La nivelul dezvoltării curente, componenta realizează o simplificare a alertelor, păstrând doar informațiile considerate necesare.

Pentru a menține timpii de lucru în parametri normali în raport cu ideea de prototip, procesarea este realizată în mod concurrent, folosind modelul *multithreading*. "Normal", în contextul curent, se referă la valorile de timp care sunt considerate standard sau acceptabile în cadrul unor soluții de securitate. După procesarea alertelor, se realizează publicarea lor într-o coadă de mesaje

²⁰Artificial intelligence

spre a fi preluate de alte componente ale arhitecturii.

2.2.4. Componența Exchange

Componența Exchange are scopul de a asigura lipsa interdependentelor între diferite componente ale arhitecturii. Din bazele de date *in-memory* analizate, se propune implementarea prin RabbitMQ, deoarece prin protocolul AMQP, descris în articolul [28], oferă mecanisme de confirmare a mesajelor, asigurând persistența alertelor într-un mediu de stocare până la consum.

În acest context, pe măsură ce sunt procesate de componenta *Log Collector*, alertele sunt adăugate într-o listă specifică fiecărui agent Wazuh care este ulterior publicată și consumată. Prin utilizarea acestui protocol în componenta care asigură transferul de informații între servicii, se elimină nevoia de implementare a componentelor mijlocitoare, cum ar fi necesar în cazul utilizării protocolelor HTTP și standardului REST.

2.2.5. Componența Command&Control

Consumul alertelor este asigurat de următoarea componentă a arhitecturii, cea de *Comandă și Control*. Funcționarea și rolul acesteia presupune identificarea sistemului conectat la rețea din care provine alerta și inițierea unei acțiuni adecvate în funcție de contextul și cerințele stabilită.

Dispozitivele și sistemele monitorizate primesc și interpretează aceste comenzi, executându-le conform specificațiilor primite de la serverul C2²¹. Prin urmare, C2 funcționează ca un *Super Node* pentru asigurarea capacitații de reacție la atac, coordonând și gestionând activitățile sistemelor monitorizate din cadrul infrastructurii. Cu alte cuvinte, scopul acestei componente este atât de a asigura reacții potrivite la alertele generate de tehnologiile configurate pe sistemele monitorizate, cât și de a asigura prevenția atacurilor asupra agentilor Wazuh prin verificarea periodică a stării acestora.

Conform articolului [24], un server de comandă și control este un sistem informatic controlat de către un atacator, utilizat în scopul transmiterii de instrucțiuni către sistemele compromise și pentru receptionarea datelor furate din cadrul unei rețele întărită. Rulează în mod obișnuit pe o mașină dedicată care este separată de restul rețelei, facilitând gestionarea și monitorizarea centrală a rețelei de calculatoare de către atacator.

Există mai multe tipuri de arhitecturi ale unui astfel de sistem, însă cea mai întâlnită este cea centralizată în care există un singur server care acționează ca punct central de control. Un astfel de server are mai multe funcționalități: poate fi dispersat în diverse locații și găzduit pe servere anonime. Odată instalată pe sistemele vizate, agentii stabilesc conexiuni de tip *callback* inițiate prin protocole de rețea standard, precum HTTP, DNS.

Din acest moment, C2 poate trimite comenzi directe ce sunt executate de către sistemele aflate sub control. Serverele C2 asigură stabilirea mecanismelor de persistență pe dispozitive, prin faptul că agentul rămâne activ și ascuns. Aceste mecanisme pot include intrări de *registry* (în cazul sistemelor Windows), sarcini programate sau instalări de servicii. În ansamblu, funcționalitățile unui server C2 sunt orientate către executarea unor acțiuni dăunătoare asupra sistemelor în care agenții sunt configurați.

Deși, în mod tradițional, un C2 este asociat cu activități dăunătoare, în acest caz acest tip de server va fi folosit într-un mod proactiv, cum este abordat și în articolele [42, 43]. Prin urmare, ca tehnică de asigurare a continuității serviciilor EDR²², funcționalitățile C2 descrise anterior pot fi utilizate și în scopuri defensive în arhitectura de securitate propusă. Mai exact, utilizarea funcționalităților specifice acestuia, precum mecanismele de persistență, comunicare și execuție de comenzi, realizează o metodă de tip *backdoor* care oferă un strat suplimentar de securitate, dacă evidențiem faptul că un atacator vizează, de obicei, vulnerabilitățile tehnologiilor bine cunoscute.

²¹Command&Control

²²Endpoint Detection and Response

2.2.6. Componența Endpoints

Toate reacțiile stabilite de Componența *Command&Control* vor fi executate pe sistemele monitorizate care fac parte din componența *Endpoints*. Această componență înglobează o serie de servere virtuale care pot fi folosite pentru a rula aplicații în AWS, caracteristicile acestora putând fi personalizate în funcție de buget și cerințe specifice.

Un agent reprezintă un program care are rolul de a lua decizii, în anumite cazuri, și de a întreprinde acțiuni pentru a atinge un obiectiv specific de securitate. Agenții reactivi sunt cei care răspund imediat la stimuli din mediul lor și iau decizii pe baza acestor stimuli. Agenții proactivi, pe de altă parte, iau inițiativa și planifică în avans pentru a-și atinge obiectivele.

Mediul în care operează un agent poate fi, de asemenea, fix sau dinamic. Mediile fixe au un set static de reguli care nu se schimbă, în timp ce mediile dinamice se schimbă constant și necesită ca agenții să se adapteze la situații noi. În cadrul implementării proiectului au fost folosite ambele tipuri de agenți: agentul proactiv este agentul Wazuh care monitorizează continuu și preia alarmele de pe sisteme într-un mod autonom de managerul Wazuh. Agentul reactiv este cel al serverului C2 care execută comenziile primite.

Prin urmare, fiecare instanță va avea doi agenți instalati și configurați: un agent Wazuh integrat cu IDS Suricata, pentru a prelua alarmele, și un agent *Command&Control* care va executa comenziile venite de la componența *Command&Control*. Un sistem SIEM precum Wazuh monitorizează, analizează și răspunde la evenimentele și amenințările de securitate, folosindu-se de agenți pentru colectarea datelor de securitate din diverse surse.

Suricata reprezintă un sistem de detecție a intruziunilor bazat pe analiza traficului de rețea, având scopul de a identifica și preveni atacurile cibernetice. Integrarea sa cu agentul Wazuh facilitează interceptarea și preluarea alertelor, beneficiind de fluiditatea colectării datelor oferită de SIEM. Astfel, se elimină dependența de specificitatea unui anumit instrument SIEM prin faptul că agentul Wazuh este utilizat ca un container de transfer, conferind un grad de generalitate soluției propuse și permitând extinderea ulterioară cu utilizarea altor tehnologii echivalente.

Fluxul proiectului evidențiat în arhitectură implică două canale de comunicare distincte: partea de monitorizare și colectare a alertelor se realizează prin conexiunea agentului Wazuh cu managerul Wazuh (urmând ca mai apoi să fie preluate alertele în restul componentelor), iar partea de reacție se realizează prin canalul stabilit de agentul *Command&Control* către serverul său.

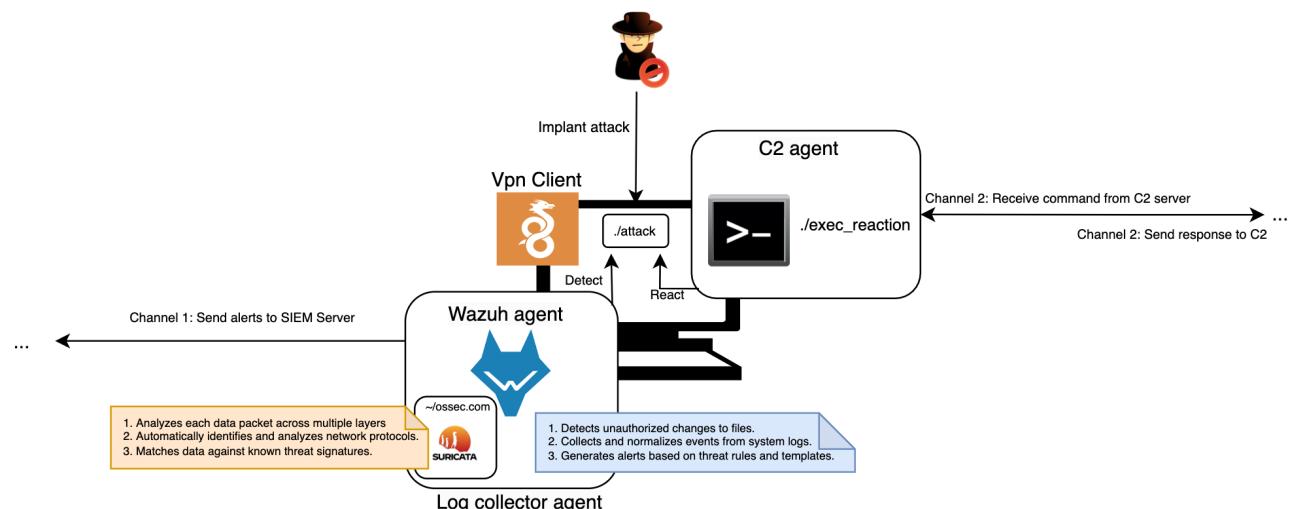


Figura 2.3. Configurare instanță monitorizată

2.3. Avantajele și dezavantajele metodei alese

2.3.1. Avantaje

Continuitatea serviciilor EDR este asigurată prin implementarea distinctă a monitorizării, prin canale de comunicare diferite față de canalele de răspuns care asigură neutralizarea atacurilor, după cum este evidențiat în Figura 2.3. Această abordare oferă avantajul unei separări clare, oferind senzația de componente care nu au legătură între ele. În cazul unui atac, de cele mai multe ori, sunt vizate serviciile, programele și aplicațiile bine cunoscute, ale căror vulnerabilități pot fi studiate. Programele cu nume banale, nesemnificative, care nu par a fi indispensabile în interacțiunea utilizatorului cu propriul computer, nu prezintă mereu interes. Ar însemna consum de resurse inutil pentru a îngreuna unele funcționalități care nu sunt necesare utilizatorului atacat. Totuși, platforma *open-source* Wazuh este bine cunoscută și oferă documentații complexe ce pot fi analizate.

Dacă atacatorii ar avea acces la sistemul compromis și observă existența unor tehnologii precum IDS, SIEM, le pot închide doar (pentru a nu trezi suspiciuni), modifica sau altera spre a compromite sistemul de monitorizare sau aplicația de securitate din care acesta face parte. Cum serverul *Command&Control* deține controlul sistemelor monitorizate, acesta poate nu doar să reacționeze când ajunge să consume alertele generate de agenții Wazuh, ci poate și să dezvolte un mecanism de *heartbeat* pentru a menține o înregistrare strictă a stărilor agenților SIEM și a preveni modificarea sau închiderea acestora. Un prim avantaj evidențiat este, aşadar, asigurarea unui strat suplimentar de securitate prin metoda de *backdoor* care se ocupă de reacție în cadrul unui canal distinct de comunicare.

Un alt avantaj al soluției propuse este rolul atribuit componentei *Command&Control*, acela de a acționa ca un element specific echipelor *Blue Team*. C2 este un sistem informatic folosit pentru a trimite instrucțiuni către alte sisteme controlate printr-un agent plasat strategic pentru a funcționa în mod continuu și neobservat. În general, funcționalitățile unui server C2 sunt interpretate ca fiind dăunătoare asupra sistemelor în care agenții sunt configurați, aşa cum s-a descris în secțiunile anterioare.

Scopul unui server C2 poate fi și reactiv, mecanismele de persistență, comunicare și execuție de comenzi fiind avantaje pentru controlul acțiunilor asupra sistemelor care necesită intervenție. Un argument pentru aceste afirmații îl constituie experimentul descris în [44], unde s-a abordat importanța evitării detectiei bazate pe rețea, aplicând teste atât pentru versiunile criptate, cât și necriptate ale C2-ului pentru a determina eficacitatea acestora în evitarea detectiei. Astfel, utilizarea infrastructurilor C2 într-un context defensiv prezintă interes și oferă echipelor de apărare instrumentele necesare pentru a detecta, răspunde și îmbunătăți reziliența împotriva atacurilor cibernetice.

Folosirea tuturor funcționalităților sistemelor SIEM în vederea creării unei soluții generalizate care folosește doar alertele generate de IDS spre a nu depinde de instrumentele SIEM care au, fiecare, particularitățile sale, reprezentă un avantaj al soluției propuse. Un agent SIEM are capacitatea de a monitoriza, analiza și răspunde la evenimentele și amenințările de securitate în mod autonom. Alertele colectate sunt trimise către serverul SIEM spre analiză și afișarea lor în *dashboard*. Un sistem SIEM poate fi implementat într-un mediu mai complex care include integrarea unor tehnologii suplimentare precum IDS.

Prin adăugarea acestor tehnologii și instrumente, sistemul devine mai cuprinzător și poate detecta o gamă mai largă de amenințări și activități suspecte în rețea. Ieșirile colectate de un astfel de sistem pot fi folosite în programe dezvoltate spre a facilita accesul la alarmele detectate. Această abordare proiectează, însă, o cuplare strânsă cu un anumit SIEM, fiecare instrument de acest tip având particularitățile proprii. Soluția este folosirea unei configurații solide a IDS, care permite transferul întregii logici de securitate către acesta, agentul SIEM având rolul exclusiv de intermediar în transferul datelor. Astfel, se evită cuplarea strânsă a componentelor arhitecturii spre a crea o soluție generalizată ce oferă posibilitatea modificării instrumentului SIEM în funcție de

cerințe și necesități.

Folosirea sistemului SIEM ca un container de transfer a alertelor IDS, implementarea unui server *Command&Control* în scop defensiv, implementarea acestor două componente principale prin canale de comunicare distincte oferă arhitecturii avantajul general de a fi scalabilă, extensibilă în funcție de dorințele și cerințele de implementare.

2.3.2. Dezavantaje

Unul dintre dezavantajele proiectului este limitarea dată de gradul de configurare particulară a IDS. O configurare matură a IDS permite ca toată logica de securitate de rețea să fie susținută de către acesta, agentului Wazuh revenindu-i doar rolul de transfer al datelor. Deși utilizarea exclusivă a IDS Suricata a fost prezentată și în secțiunea de avantaje, oferind un grad mare de generalizare și asigurând lipsa de dependență de instrumentul SIEM, această abordare are și o serie de dezavantaje în ceea ce privește asigurarea captării majorității tipurilor de atacuri, nu doar cele de tip *System Intrusion*. IDS Suricata este specializat pe detectarea atacurilor la nivel de rețea, dar poate avea dificultăți în a detecta atacuri foarte specifice la nivel de aplicație, ca de exemplu *SQL injection* sau *Cross-Site Scripting* în aplicații web.

De asemenea, nu sunt în atenția IDS Suricata atacurile care implică manipularea utilizatorilor pentru a oferi informații sensibile, cum ar fi *phishing*. Acestea se bazează pe vulnerabilitățile umane și nu pe anomalii de rețea. Ulterior, după instalarea codului malicios pe sistem, IDS poate reuși să identifice comportament suspect. Așadar, acest dezavantaj poate fi tratat prin renunțarea la folosirea SIEM drept container de transfer și utilizarea funcționalităților care tratează și aceste atacuri. De exemplu, integrarea IDS Suricata cu IBM QRadar și folosirea tuturor funcționalităților oferite de ambele sisteme poate conduce către acoperirea unei plaje mult mai largi de tipuri de atacuri, însă se renunță la ideea de generalitate.

Arhitectura soluției a fost proiectată pentru un mediu potrivit dezvoltării unui prototip funcțional. Abordarea folosită este cea de a crea mai multe sisteme monitorizate prin utilizarea de mașini virtuale în *cloud*, mai exact utilizarea *Amazon Elastic Compute Cloud*. Alocarea resurselor la cerere este posibilă în modelele de calcul în *cloud* datorită capacitații lor flexibile și duce la soluții optime din punct de vedere al costurilor și al energiei.

Utilizarea unui sistem de *cloud computing* reduce semnificativ costul de configurare al unei infrastructuri software, performanțele variind în funcție de timp, tipul de operații efectuate, dimensiunea și tipul de fișier. Prin urmare, deși reprezintă un avantaj toate aceste funcționalități care permit accesul și configurarea rapidă a mașinilor, această metodă trebuie abandonată odată ce proiectul încheie faza de prototip. Este necesară testarea și simularea atacurilor pe sisteme în context real pentru a observa comportamentul în cadrul utilizării uzuale a *endpoint-urilor* monitorizate.

Un alt dezavantaj identificat este lipsa unei componente care să aibă rolul exclusiv de a clasifica, grupa și organiza alarmele într-un mod automatizat, astfel încât componenta *Command&Control* să preia clasificările directe ale alertelor și nu să se ocupe de iterarea alertelor pentru analiza independentă a acestora în vederea stabilirii reacției potrivite.

2.4. Limitele de funcționare

Așa cum s-a prezentat în secțiunea de dezavantaje, o limitare menționată este dată de gradul de tratare clasică, iterativă a alertelor în cadrul componentei *Command&Control* din arhitectura 2.1 în vederea identificării și neutralizării fiecărui tip de atac identificat în alerte. Una dintre cele mai mari limitări ale IDS în sine este abordarea bazată pe semnături, deoarece nu sunt recunoscute atacurile noi sau modificările și extensiile celor existente. Mai mult decât atât, menținerea bazei de date cu reguli la zi reprezintă un cost semnificativ, ceea ce înseamnă că, uneori, sistemele IDS pot să nu fie actualizate în acord cu atacurile noi.

Autorii articolelor [45] au concluzionat că o metodă de a îmbunătăți acuratețea alertelor și clasificarea corespunzătoare a acestora constă în utilizarea funcționalităților IDS alături de tehnici

specifice domeniului Învățare Automată. Mai exact, s-a detaliat că metoda hibridă propusă poate construi automat distribuția pachetelor normale și detecta deviațiile.

Capitolul 3. Implementarea aplicației

3.1. Descrierea generală a implementării

3.1.1. Abordarea generală a implementării

Implementarea soluției prezentate în Capitolul 2 a presupus stabilirea inițială a unor etape clare, bine structurate, pentru a urma un flux de lucru cât mai coerent. Începutul proiectului a constat în configurarea tuturor mașinilor virtuale de care a fost nevoie: o mașină virtuală locală pentru instalarea și configurarea serverului SIEM și un număr de cel puțin 3 mașini virtuale în *cloud* care au rolul de sisteme monitorizate. Mașinile virtuale în *cloud* sunt cele disponibile pe AWS. Configurarea lor a constat în crearea propriu-zis a mașinilor virtuale alături de un grup de securitate care reprezintă o listă de reguli care controlează traficul de intrare și ieșire pe baza porturilor și adreselor IP permise.

Conectarea la acestea a fost realizată prin protocolul de rețea criptografic *Secure Shell* de pe mașina gazdă. Configurarea serverului SIEM a fost realizată pe o mașină virtuală locală prin urmarea tutorialului disponibil în documentație, [46]. S-a instalat și importat fișierul OVA²³ în VirtualBox, un software de virtualizare dezvoltat de Oracle. Ulterior a fost realizată integrarea serverului SIEM cu stiva ELK. S-a instalat motorul de procesare a datelor *server-side* Logstash, configuriările necesare fiind aplicate în fișierul `wazuh-elasticsearch.conf`.

După aceasta etapă, există *set-up*-ul necesar pentru mașinile virtuale, însă nu este asigurată o modalitate ca acestea să comunice între ele. Pentru aceasta, se folosește un VPN a cărui server este configurat pe o instanță separată EC2. Clientii VPN sunt toate celelalte mașini virtuale, inclusiv mașina gazdă.

Din acest punct se poate începe instalarea și configurarea instrumentelor care generează, colectează și transmit alertele de pe *endpoints*. A fost instalat IDS Suricata care a fost configurat astfel încât să monitorizeze interfața VPN, adresa IP corectă și a fost adăugat un fișier de reguli personalizabil. Pentru testarea funcționării IDS s-a scris o regulă Suricata, apoi a fost urmărită ieșirea fișierului `fast.log` pentru a observa funcționarea corectă. După aceasta, a fost instalat agentul Wazuh și configurat fișierul `ossec.conf` pentru a prelua și evenimentele din fișierul de jurnalizare `eve.json` al IDS Suricata. Testarea funcționării a fost realizată în aceeași manieră, observând de această dată *dashboard*-ul Wazuh pentru validare.

În această etapă, mediul de lucru este pregătit pentru a începe dezvoltarea componentelor care includ cod și implementare de funcționalități. Așa cum este detaliat în Capitolul 2, se folosește instrumentul Wazuh pentru colectarea și preluarea alertelor generate de Suricata pentru a elimina reimplementarea unui mecanism clasic de *Log Collector*. Pentru asta, se folosește API ELK.

Odată ce alertele sunt preluate periodic în componenta *Log Collector*, se realizează o procesare a lor prin păstrarea doar a informațiilor considerate necesare și prin adăugarea timpilor când a început procesarea, respectiv publicarea în coada de mesaje. Acești timpi sunt folosiți pentru a obține rezultate legate de timpul mediu de procesare, timpul mediu dintre publicare și consumare a alertelor fiecărui agent.

Pentru a evita tratarea secvențială a alertelor, se folosește modelul *multithreading* cu un *ThreadPool Executor* care asignează un fir de execuție pentru fiecare agent existent. Se folosește protocolul AMPQ, mai exact coada de mesaje RabbitMQ pentru a asigura transferul de informații dintre componente fără ca acestea să comunice între ele. Configurarea RabbitMQ se realizează în cele două componente care presupun scriere de cod, *Log Collector* și *Command&Control*. Totuși, pentru a nu crea cod obfuscat și a separa funcționalitățile pentru verificarea mai ușoară a corectitudinii, a fost creat un program temporar în care s-a implementat procesul de consum în

²³Open Virtualization Format

aceeași manieră periodică și folosind aceeași metodă amintită. Se validează funcționalitatea curentă prin testare și observarea că alertele au fost procesate corect, publicate fiecare în lista proprie a unui agent, respectiv consumate total, în timpii stabiliți în program.

Odată ce prima parte a soluției este implementată, atenția este îndreptată către componenta Command&Control. Aceasta constă în crearea unui program automatizat care așteaptă conexiuni de tip *callback* de la agenții pe aceleași sisteme monitorizate pe care rulează și Suricata și Wazuh pentru a putea trimite comenzi către aceștia când vor fi consumate alertele. Prevenția închiderii agentului Wazuh este implementată în această componentă, atribuindu-se acestui obiectiv un fir de execuție separat.

Se realizează, în acest fel, mecanismul de *heartbeat* subliniat în Capitolul 2. Testarea componentei se realizează prin observarea execuției comenziilor și întoarcerea răspunsului corect de la agent la server. După aceasta, se poate realiza integrarea funcționalității de consum și renunțarea la programul temporar. Se adaugă verificarea tipurilor alarmelor și se stabilesc reacții specifice pentru unele dintre acestea (raportat la faza de prototip a soluției).

Pentru testarea funcționalității complete se folosesc două instrumente de *hacking* pentru a genera atacuri asupra sistemelor monitorizate. Primul instrument este Hydra care este folosit pentru atacuri specifice de tip *Brute Force* și care, print-o singură linie de comandă, alături de două fișiere care conțin liste de parole și nume de utilizatori, generează un atac. Al doilea instrument este o platformă utilizată pentru dezvoltarea și executarea *exploit*-urilor, oferind posibilitatea de a testa securitatea într-un mod controlat.

Așadar, se verifică următorul flux: se asigură că instrumentele necesare sunt funcționale, se pornesc programele ce reprezintă componentele Command&Control și Log Collector și se pornește un atac de tip *SSH Brute Force*, de exemplu, folosind unul dintre cele 2 instrumente menționate anterior. După perioada de timp prestabilită în componentă, se realizează preluarea alertelor generate de IDS, se proceseză, se publică în coadă, se consumă câte un element din coadă reprezentat printr-o listă specifică unui agent, se identifică tipul atacului și se transmite o comandă specifică agentului de pe care a fost generată alerta. În acest mod, atacul este neutralizat.

3.1.2. Componența SIEM

3.1.2.1. Descriere generală

Componența SIEM are rolul de a monitoriza, analiza, colecta, și transfează alertele generate în vederea observării lor în *dashboard*-ul Wazuh sau prelucrarea lor adițională în programe separate. Detalii conceptuale suplimentare despre componența sunt aduse în capitolul precedent, 2. Din perspectiva implementării, posibilitatea de accesare a alertelor în afara interfeței asigurate de Wazuh implică o serie de configurații și integrări adiționale în cadrul serverului SIEM. Mai exact, este necesară integrarea sa cu stiva ELK care asigură o filtrare suplimentară a alertelor și indexarea acestora pentru a oferi acces mai rapid și mai flexibil. Detaliile de implementare sunt detaliate în secțiunea următoare.

3.1.2.2. Structura configurației

Implementarea acestora se începe prin instalarea Elasticsearch și configurația minimală a fișierului `elasticsearch.yml`, modificând `network.host` pentru a permite Elasticsearch să asculte pe toate interfețele de rețea. Partea de setări de securitate a fost păstrată așa cum este în configurația inițială, după cum este evidențiat în Figura 3.1.

```

# By default Elasticsearch is only accessible on localhost. Set a different
# address here to expose this node on the network:
#
#network.host: 192.168.0.1
network.host: 0.0.0.0
#
# By default Elasticsearch listens for HTTP traffic on the first free port it
# finds starting at 9200. Set a specific HTTP port here:
#
#http.port: 9200
#
# For more information, consult the network module documentation.
#
# ----- Discovery -----
#
# Pass an initial list of hosts to perform discovery when this node is started:
# The default list of hosts is ["127.0.0.1", "[::1]"]
#
#discovery.seed_hosts: ["host1", "host2"]
#
# Bootstrap the cluster using an initial set of master-eligible nodes:
#
#cluster.initial_master_nodes: ["node-1", "node-2"]
#
# For more information, consult the discovery and cluster formation module documentation.
#
# ----- Various -----
#
# Allow wildcard deletion of indices:
#
#action.destructive_requires_name: false
action.auto_create_index: .monitoring*,.watches,.triggered_watches,.watcher-history*,.ml*,wazuh-alerts-*
#
#----- BEGIN SECURITY AUTO CONFIGURATION -----
#
# The following settings, TLS certificates, and keys have been automatically
# generated to configure Elasticsearch security features on 02-04-2024 15:03:11
#
#
# Enable security features
xpack.security.enabled: true #HERE I MODIFIED

xpack.security.enrollment.enabled: true

# Enable encryption for HTTP API client connections, such as Kibana, Logstash, and Agents
xpack.security.http.ssl:
  enabled: true
  keystore.path: certs/http.p12

# Enable encryption and mutual authentication between cluster nodes
xpack.security.transport.ssl:
  enabled: true
  verification_mode: certificate
  keystore.path: certs/transport.p12
  truststore.path: certs/transport.p12
# Create a new cluster with the current node only
# Additional nodes can still join the cluster later
cluster.initial_master_nodes: ["MacBook-Pro-Adim.local"]

# Allow HTTP API connections from anywhere
# Connections are encrypted and require user authentication
http.host: 0.0.0.0

# Allow other nodes to join the cluster from anywhere
# Connections are encrypted and mutually authenticated
#transport.host: 0.0.0.0

#----- END SECURITY AUTO CONFIGURATION -----

```

Figura 3.1. Configurare Elasticsearch

Mai departe, se trece la instalarea Logstash ce presupune urmarea unor tutoriale pentru instalarea sa și a unor *plugins*. Configurarea începe prin crearea unor indici noi și maparea lor în Elasticsearch pentru a asigura stocarea datelor indexate corect. Setarea corectă a mapării ajută Elasticsearch în procesul de căutare a datelor. Se folosește `es_template.json` pentru a configura inițializarea indecșilor. Ulterior, se modifică `alerts.json` pentru a crea un pipeline Logstash care permite utilizarea de *plugins* pentru citirea datelor și trimiterea lor către Elasticsearch. Acest

pipeline este stocat în `wazuh-elasticsearch.conf` și conține 2 componente, fiecare cu propriile configurări, după cum arată Listing 3.1.

```

input {
    file {
        id => "wazuh_alerts"
        codec => "json"
        start_position => "beginning"
        stat_interval => "1 second"
        path => "/var/ossec/logs/alerts/alerts.json"
        mode => "tail"
        ecs_compatibility => "disabled"
    }
}
output {
    elasticsearch {
        hosts => "10.177.186.2"
        index => "wazuh-alerts-4.x-%{+YYYY.MM.dd}"
        user => '${ELASTICSEARCH_USERNAME}'
        password => '${ELASTICSEARCH_PASSWORD}'
        ssl => true
        cacert => "/etc/logstash/elasticsearch-certs/http_ca.crt"
        template => "/etc/logstash/templates/wazuh.json"
        template_name => "wazuh"
        template_overwrite => true
    }
}

```

Listing 3.1. Configurare pipeline în `wazuh-elasticsearch.conf`

Prin urmare, componenta - input are scopul de a indica locul unde sunt stocate datele - path atribuind un tip acestora - id , wazuh-alerts, și specifică modul în care datele din fișier sunt decodificate codec, în acest caz fiind JSON. Componenta output definește destinația datelor procesate. Se specifică adresa serverului Elasticsearch - hosts, alături de numele indexului creat zilnic unde vor fi stocate datele.

Testarea stării serviciilor poate fi verificată conform Figurilor 3.2 și 3.3 și prin requests către serviciul Elasticsearch pentru a observa dacă returnează alertele afișate și în dashboard Wazuh.

```

[2024-06-30T17:22:55,796][INFO ][o.e.x.s.a.Realms      ] [MacBook-Pro-Adim.local] license mode is [basic], currently
licensed security realms are [reserved/reserved,file/default_file,native/default_native]
[2024-06-30T17:22:55,799][INFO ][o.e.i.ClusterStateLicenseService] [MacBook-Pro-Adim.local] license [17a28d1c-c0b4-4acc
-bfe2-2dfbbee7e681] mode [basic] - valid
[2024-06-30T17:22:55,802][INFO ][o.e.g.GatewayService     ] [MacBook-Pro-Adim.local] recovered [25] indices into cluste
r_state
[2024-06-30T17:22:57,905][INFO ][o.e.h.n.s.HealthNodeTaskExecutor] [MacBook-Pro-Adim.local] Node [{MacBook-Pro-Adim.loc
al}{DIRzI6czRa2quFZoNzFHUQ}] is selected as the current health node.
[2024-06-30T17:23:17,079][INFO ][o.e.c.r.a.AllocationService] [MacBook-Pro-Adim.local] current.health="GREEN" message=
Cluster [health status changed from [RED] to [GREEN]] (reason: [shards started [[wazuh-alerts-4.x-2024.04.02][2]]]." pre
vious.health="RED" reason="shards started [[wazuh-alerts-4.x-2024.04.02][2]]"

```

Figura 3.2. Status activ Elasticsearch

```

[[wazuh-user@wazuh-server ~]$ sudo systemctl status logstash
● logstash.service - logstash
   Loaded: loaded (/usr/lib/systemd/system/logstash.service; enabled; vendor preset: disabled)
   Active: active (running) since Du 2024-06-30 17:24:13 UTC; 2h 57min left
     Main PID: 2418 (java)
        CGroup: /system.slice/logstash.service
                  └─2418 /usr/share/logstash/jdk/bin/java -Xms1g -Xmx1g -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djru...

jun 30 14:26:27 wazuh-server logstash[2418]: [2024-06-30T14:26:27,466][INFO ][logstash.runner           ] Jackson...0000` 
jun 30 14:26:27 wazuh-server logstash[2418]: [2024-06-30T14:26:27,467][INFO ][logstash.runner           ] Jackson...0000` 
jun 30 14:26:29 wazuh-server logstash[2418]: [2024-06-30T14:26:29,295][INFO ][logstash.agent          ] Success...also}
jun 30 14:26:30 wazuh-server logstash[2418]: [2024-06-30T14:26:30,225][INFO ][org.reflections.Reflections] Refle...alue
jun 30 14:26:30 wazuh-server logstash[2418]: [2024-06-30T14:26:30,970][INFO ][logstash.codecs.json    ] ECS compatib...
jun 30 14:26:31 wazuh-server logstash[2418]: [2024-06-30T14:26:31,523][WARN ][logstash.outputs.elasticsearch] Yo...u hav
jun 30 14:26:31 wazuh-server logstash[2418]: [2024-06-30T14:26:31,538][WARN ][logstash.outputs.elasticsearch] Yo...s abo
jun 30 14:26:31 wazuh-server logstash[2418]: [2024-06-30T14:26:31,602][INFO ][logstash.javapipeline       ] Pipelin...wise.
jun 30 14:26:31 wazuh-server logstash[2418]: [2024-06-30T14:26:31,621][INFO ][logstash.outputs.elasticsearch][ma...2"]
jun 30 14:26:32 wazuh-server logstash[2418]: [2024-06-30T14:26:32,157][INFO ][logstash.outputs.elasticsearch][ma...0/}]
Hint: Some lines were ellipsized, use -l to show in full.
[[wazuh-user@wazuh-server ~]$ 

```

Figura 3.3. Status activ Logstash

3.1.2.3. Comunicarea cu alte componente

Comunicarea cu alte componente poate fi detaliată din 2 perspective. Prima ține de mecanismul ce cuprinde componentele Wazuh Manager, Wazuh Indexer, Logstash și Elasticsearch, mecanism descris în secțiunea anterioară. A doua perspectivă se referă la comunicarea cu restul componentelor arhitecturii detaliate în 2. Mai exact, componenta SIEM oferă acces și trimite alerte, la cerere, către componenta Log Collector prin ELK API. O altă comunicare o constituie cea dintre Agent Wazuh și Server Wazuh, serverul primind datele generate de agent și alertele IDS Suricata prin primul canal de comunicare al soluției propuse.

3.1.3. Componenta Log Collector

3.1.3.1. Descriere generală

Componenta Log Collector a fost implementată folosind limbajul de programare multiparadigmă *Python* pentru a aplica o procesare alertelor preluate de la componenta SIEM și a le transfера către componenta care se ocupă de funcționalitatea de reacție la alerte. Are un rol de intermediere, implementarea sa realizându-se ca un proiect separat ce utilizează o serie de biblioteci specifice pentru a crea funcționalitățile descrise în Capitolul 2 într-un mod în care timpii nu depășesc limitele acceptabile pentru o aplicație prototip. Mai departe, implementarea va fi detaliată pentru fiecare fișier al componentei.

3.1.3.2. Structura codului

Fișierul `log_elk_collector.py` conține toate funcțiile necesare pentru orchestrarea tuturor proceselor prin care alertele trebuie să treacă. Funcția principală a fișierului, prezentă în Listing 3.2, este `collect_and_send_alerts()`. Aceasta începe cu linia 18 care conține un decorator utilizat pentru a marca funcția ca fiind repetată la intervale de timp definite de funcția `every` din biblioteca `schedule`. Acest lucru este verificat și asigurat de `schedule.run_pending()` din fișierul `main.py`.

Prin apelarea funcției `get_all_agents()` prezentă în Listing 3.6, se preia lista tuturor agentilor înregistrați de Wazuh Manager. După aceasta, este setat numărul de fire de execuție pentru a se potrivi cu numărul de procesoare disponibile, asigurând astfel că fiecare procesor este utilizat eficient fără a supraîncărca sistemul. Se folosesc funcționalitățile clasei `ThreadPoolExecutor` din modulul `futures` a bibliotecii `concurrent`, în vederea eliminării abordării secvențiale a tratării alarmelor.

Pentru asigurarea orchestrării tuturor funcțiilor care prelucrează alertele, este utilizată funcția `process_agent_alerts()`, cu parametrul `agent_id`, a cărei cod se află în Listing 3.2. Pentru început, se preiau alertele din intervalul stabilit astfel încât să se eliminate posibila preluare repetată a alarmelor și se păstrează doar lista cu alerte din întreg răspunsul venit pentru `request`-ul realizat la linia 9 din Listing 3.5. Ulterior, se apelează funcția de filtrare a alertelor pentru a le simplifica în acord cu modelul `ALERT_BODY` stabilit în Listing 3.5. După procesare, se apelează funcția `publish_to_rabbitmq(formatted_alerts)` al cărei rol este de a crea un mecanism *thread-safe* pentru publicarea în coadă. Se folosește un obiect `mutex` pentru a bloca accesul la coada cat timp un fir de execuție are acces la aceasta resursă.

```

1  from concurrent.futures import ThreadPoolExecutor
2  from wazuh.wazuh_api_client import *
3  from schedule import every, repeat
4  import multiprocessing
5  from util.alert_processing import *
6  from util.constants import *
7  def process_agent_alerts(agent_id):
8      try:
9          alerts = get_suricata_alerts_last_scheduled_interval(agent_id)

```

```

10 if alerts:
11     formatted_alerts = filter_alert_body(alerts)
12     publish_to_rabbitmq(formatted_alerts)
13
14 except Exception as e:
15     logging.error(f"[!] [process_agent_alerts({agent_id})] ERROR when collection
16         → alerts for agent {agent_id}: {e}")
17     print(f"[!] [process_agent_alerts({agent_id})] ERROR when collection alerts for
18         → agent {agent_id}: {e}")
19
20 @repeat(every(SCHEDEDL_INTERVAL).seconds)
21 def collect_and_send_alerts():
22     agent_ids = get_all_agents()
23     num_cpus = multiprocessing.cpu_count()
24     with ThreadPoolExecutor(max_workers=num_cpus) as executor:
25         _ = [executor.submit(process_agent_alerts, agent_id) for agent_id in agent_ids]

```

Listing 3.2. Funcțiile principale ale *Log Collector*

Fișierul `rabbit.py` implementează o clasă care inițializează credențialele și parametrii necesari pentru accesul la coada de mesaje. Acesta conține funcția `send_message(formatted_alerts)` care este apelată în fișierul `log_elk_collector.py` în contextul unui mutex pentru a limita accesul simultan al firelor de execuție la aceeași funcție. În cadrul funcției de la linia 6 se inițializează o conexiune către RabbitMQ folosind tipul de conexiune *BlockingConnection* care blochează firul curent până când operațiunea este completă.

După inițializare, se deschide un canal care permite trimitera și primirea de mesaje. Se realizează în acest moment adăugarea timpului de publicare în alerte. Acest pas a fost realizat aici și nu în funcția de simplificare, încrucișând posibilitatea salvării unui timp mai aproape de momentul producerii evenimentului.

După ce alertele sunt pregătite, se realizează publicarea personalizată (linia de cod 22 din Listing 3.3). Parametrul `exchange` stabilește locul unde sunt trimise mesajele înainte de a fi redirecționate către o coadă specifică. Parametrul `routing_key` decide către ce coadă va fi trimis mesajul și în ce mod - direct. Parametrul `body` primește conținutul mesajului, adică lista de alerte specifică unui agent. Pentru acestea s-a folosit biblioteca `pika`, deoarece oferă suport pentru confirmările mesajelor, asigurând procesarea corectă a acestora.

```

1 import datetime
2 import json
3 from util.constants import *
4 import pika
5
6 def send_message(self, formatted_alerts):
7     try:
8         if not all(self.parameters):
9             logging.error(f"[!] [send_message({self, formatted_alerts})] ERROR Parameters
10                 → of connection are missing or are invalid")
11         with pika.BlockingConnection(self.parameters) as connection:
12             with connection.channel() as channel:
13                 if not channel.is_open or not
14                     → channel.queue_declare(queue=self.config['queue'], passive=True):
15                     logging.error(f"[!] [send_message({self, formatted_alerts})] ERROR Channel
16                         → or queue {self.config['queue']} does not exists")
17                     add_publishing_timestamp = lambda alert: {**alert, TIMESTAMP_SUBDOCUMENT:
18                         → {**alert[TIMESTAMP_SUBDOCUMENT], PUBLISHING_FIELD:
19                             → str(datetime.datetime.now())}}
20                     formatted_alerts = list(map(add_publishing_timestamp, formatted_alerts))
21                     formatted_alerts = json.dumps(formatted_alerts, indent=4)

```

```

17     logging.info(f"[+] [send_message(formatted_alerts)] INFO: Successfully
18         ↳ processed alerts {formatted_alerts}")
19
20     formatted_alerts = formatted_alerts.encode()
21     channel.basic_publish(exchange=self.config['exchange'],
22                           routing_key=self.config['routing_key'],
23                           body=formatted_alerts)
24     except pika.exceptions.AMQPError as e:
25         logging.error(f"[!] [send_message(self, formatted_alerts)] ERROR AMQP: {e}")
26         print(f"[!] [send_message(self, formatted_alerts)] ERROR AMQP: {e}")
27     except Exception as e:
28         logging.error(f"[!] [send_message(self, formatted_alerts)] ERROR Sending the
29             ↳ message: {e}")
30         print(f"[!] [send_message(self, formatted_alerts)] ERROR Sending the
31             ↳ message: {e}")

```

Listing 3.3. Funcția de publicare în coada de mesaje RabbitMQ

Fișierul `alert_processing.py` conține funcția de simplificare a alertelor. Aceasta constă în dezvoltarea unei funcții interne recursive care iterează prin alertele primite și creează noi dicționare, păstrând doar cheile care sunt prezente în constanta `ALERT_BODY`. S-a ales o modalitate recursivă pentru a asigura iterarea în documentele imbricate. În acest pas se înregistrează timpul de începere al procesării observabil la linia 27 din Listing 3.4.

```

1 def filter_recursive(original, structure):
2     simplified = {}
3     # Iterate over the keys and values in the filter structure
4     for key, value in structure.items():
5         # If the value is a dictionary, recursively filter the original alert
6         if isinstance(value, dict):
7             if key in original and isinstance(original[key], dict):
8                 simplified[key] = filter_recursive(original[key], value)
9
10    # If the key exists in the original alert and matches the filter, ignore the alert
11    elif key in original and original[key] == FILTER_ON_CATEGORY_FIELD.get(key):
12        return {}
13
14    # If the key exists in the original alert, include it in the simplified alert
15    elif key in original:
16        simplified[key] = original[key]
17
18    # If there is an empty entry, the alert is ignored
19    if {} in simplified.values():
20        return {}
21    else:
22        return simplified
23
24    for alert in alerts:
25        # Filter the data and agent fields of the alert and create a simplified alert
26        simplified_alert = {
27            'test_latency': {PROCESSING_FIELD: str(datetime.datetime.now())},
28            'data': filter_recursive(alert.get('data', {}), ALERT_BODY.get('data', {})),
29            'agent': filter_recursive(alert.get('agent', {}), ALERT_BODY.get('agent', {}))
30        }
31        # timestamp for publishing is saved in rabbit.py: send_message()
32        simplified_alerts.append(simplified_alert)
33
34    return simplified_alerts
35
36    except json.JSONDecodeError as e:
37        logging.error(f"[!] [filter_alert_body()] ERROR JSON decoding error: {e}")

```

```
37     print(f"[!] [filter_alert_body()] ERROR JSON decoding error: {e}")
38     sys.exit(1)
39
40 except KeyError as e:
41     logging.error(f"[!] [filter_alert_body()] ERROR Key error:{e}")
42     print(f"[!] [filter_alert_body()] ERROR Key error:{e}")
43     sys.exit(1)
44
45 except Exception as e:
46     logging.error(f"[!] [filter_alert_body()] ERROR - unknown: {e}")
47     print(f"[!] [filter_alert_body()] ERROR - unknown: {e}")
48     sys.exit(1)
```

Listing 3.4. Funcția de simplificare a alertelor

Fișierul constants.py conține toate constantele considerate necesare în program: parametri necesari pentru *request* către API ELK, informațiile despre conectarea către Wazuh API, respectiv API ELK. Două dintre cele mai importante constante sunt evidențiate în Listing 3.5 unde se observă *template*-ul pe care trebuie să îl urmeze alertele procesate. Mai exact, informațiile considerate necesare sunt cele despre IP-urile sursei și destinației, astfel încât în componenta de reacție să se poată observa dacă există comunicări suspecte între sistemul monitorizat și un altul. Câmpul timestamp oferă un alt punct de reper în vederea evaluării timpilor operațiilor importante ale alertei. Sub-documentul alert oferă informațiile de bază despre tipul alertei generate, astfel încât, în faza de prototip, se pot implementa reacții diferite în funcție de aceste informații. Ultimul sub-document evidențiat este agent care conține informațiile despre agentul care a generat alerta, astfel încât componenta Command&Control să acceseze rapid canalul de comunicare cu acel agent și să trimită reacția.

```
ALERT_BODY = {
    "data": {
        "dest_ip": None,
        "timestamp": None,
        "dest_port": None,
        "src_port": None,
        "src_ip": None,
        "proto": None,
        "event_type": None,
        "alert": {
            "signature_id": None,
            "action": None,
            "severity": None,
            "signature": None,
            "category": None
        }
    },
    "agent": {
        "id": None,
        "name": None,
        "ip": None
    },
    "test_latency": {
    }
}
```

Listing 3.5. Template cu informațiile păstrate pentru alerte

Fișierul wazuh_api_client.py conține funcțiile de autentificare necesare pentru accesarea datelor de la Wazuh Manager, precum și funcția de primire a listei de agenți înregistrați, evidențiată în Listing 3.6. Pentru început, la fiecare apel al funcției, decoratorul @authenticate

verifică dacă un *token* este furnizat înainte de a permite funcției `get_all_agents(*, token=None)` să fie executată. Ulterior, se realizează un *request* către Wazuh API și răspunsul și se elimină id-ul serverului din listă, conform liniei 19 din Listing 3.6.

```

1  @authenticate
2  def get_all_agents(*, token=None):
3      global frame, filename
4      funcname = inspect.getframeinfo(frame).function
5
6      url = f"{PROTOCOL}://{WAZUH_HOST}:{WAZUH_PORT}/{LIST_AGENTS_IDS_QUERY}"
7      headers = {
8          "Content-Type": "application/json",
9          "Authorization": f"Bearer {token}"
10     }
11
12     try:
13         # Send a GET request to the Wazuh API endpoint
14         response = requests.get(url, headers=headers, verify=False)
15         response.raise_for_status()
16         agent_ids = response.json()["data"].get("affected_items", [])
17
18         # Filter out any agent ID equal to "000"
19         filtered_agent_ids = [agent["id"] for agent in agent_ids if agent["id"] != "000"]
20         print(f"agent_ids = {filtered_agent_ids}")
21         return filtered_agent_ids
22
23     except requests.exceptions.RequestException as e:
24         # If an error occurs during the request, print the error message
25         logging.error(f"[!] [{filename}] [{funcname}] ERROR while getting agents list: {e}")
26         print(f"[!] [{filename}] [{funcname}] ERROR while getting agents list: {e}")
27         sys.exit(1)

```

Listing 3.6. Funcția de accesare a listei de agenți înregistrați de Wazuh

3.1.3.3. Comunicarea cu alte componente

Componenta Log Collector comunică cu două dintre componente arhitecturii. Primește periodic alerte prin *request* către API ELK și le transmite către componenta Command&Control. Transmiterea alertelor se realizează, astă cum s-a detaliat și mai sus, indirect, prin utilizarea cozii de mesaje RabbitMQ.

3.1.4. Componenta Exchange

3.1.4.1. Descriere generală

Componenta Exchange are rolul de a ruta evenimentele către componenta Command&Control. Comunicarea între servicii se poate realiza prin protocole ușor de utilizat, precum HTTP, AMQP. În cadrul soluției propuse, s-a optat pentru utilizarea protocolului AMQP, în special prin intermediul RabbitMQ, datorită funcționalităților sale extinse și a accesibilității în utilizare. Această alegere a fost preferată în detrimentul complexității ridicate asociate cu Kafka sau a lipsei de persistență a datelor pe care o oferă Redis, astă cum s-a arătat în Capitolul 1. De asemenea, a fost ales RabbitMq care urmează protocolul AMPQ în detrimentul HTTP pentru ușurință și configurația sa rapidă, nefiind necesar o altă componentă separată API Gateway.

3.1.4.2. Structura codului/configurării

Pentru utilizarea RabbitMQ a fost necesară configurarea cozii de mesaje atât în interfață, cât și în cod. Se observă în Figura 3.4 coada configurată cu tipul `classic` și numele `tpeexample.queue`. Configurarea de tip `classic` oferă funcționalități precum persistența mesajelor, aplicarea principiului FIFO²⁴, opțiunea de a configura distribuirea sarcinilor între mai mulți consumatori direct din coadă, confirmarea mesajelor, dar și durabilitate. S-a ales configurarea clasică în detrimentul tipurilor *Quorum Queue*, *Stream Queue*, *Lazy Queue*, deoarece statusul de prototip al soluției nu implică nevoie de utiliza o configurare mai complexă. În configurarea din interfață se observă și cheia de rutare folosită și numele `exchange` de unde vin mesajele.

Overview				Messages			Message rates			
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	tpeexample.queue	classic	D Args	running	0	0	0	0.00/s	0.00/s	0.00/s

▼ Bindings (2)

From	Routing key	Arguments
(Default exchange binding)		
tpeexample.direct	tpeexample.routingkey	Unbind

Figura 3.4. Configurare coadă RabbitMQ în aplicație

Odată ce coada are statusul activ în interfață, se poate configura mediul de accesare a acesteia din cod, conform Listing 3.7. Se utilizează biblioteca `pika` pentru a crea credențialele și configura parametrii necesari pentru stabilirea conexiunii. Datele prezentate în clar au scop informativ.

```
config = {
    'host': '0.0.0.0',
    'port': 5672,
    'username': 'student',
    'password': 'student',
    'exchange': 'tpeexample.direct',
    'routing_key': 'tpeexample.routingkey',
    'queue': 'tpeexample.queue'
}
credentials = pika.PlainCredentials(config['username'], config['password'])
parameters = (pika.ConnectionParameters(host=config['host']),
              pika.ConnectionParameters(port=config['port']),
              pika.ConnectionParameters(credentials=credentials))
```

Listing 3.7. Configurare coadă RabbitMQ în cod

3.1.4.3. Comunicarea cu alte componente

Ca o componentă de transfer de informații, aceasta trebuie să stabilească comunicarea cu cele două componente care trimit și primesc alertele.

3.1.5. Componenta Command&Control

3.1.5.1. Descriere generală

Așa cum s-a detaliat și în Capitolul 2, are un rol de stabilire și transmitere de reacții la atacuri, implementarea sa realizându-se ca un proiect separat pentru a putea distinge complet cele 2 servicii între ele. Prin implementarea acestei componente, se renunță la analiza manuală a anomaliei și stabilirea reacției la acestea, limitând considerabil nevoia de acțiune umană. Se abordează aceeași manieră de eliminare a metodei secvențiale prin încercarea de a crea fire de execuție specializeze pe principalele atribuții ale componentei.

²⁴First In First Out

3.1.5.2. Structura codului/configurării

Fișierul `consumer_mq.py` asigură funcțiile necesare consumului listelor de alerte din coadă și procesarea lor în vederea stabilirii deciziei potrivite. Funcția principală a fișierului, prezentă în Listing 3.8, este `main_consumer()`. Aceasta începe cu linia 1 care conține același decorator utilizat și în fișierul `log_elk_collector.py` din componenta Log Collector pentru a asigura o periodicitate în apelul său.

Urmează același tipar de creare a unui obiect `ThreadPoolExecutor` unde atribuie câte un fir de execuție pentru fiecare listă consumată din coadă spre a separa procesările alertelor pentru fiecare agent în parte.

Funcția de orchestrare în acest caz este `receive_process_alerts_from_rabbitmq()` care asigură consumul concurrent din coadă folosind un obiect mutex. După primirea listei, se execută o serie de procesări în funcția clasei RabbitMQ populează o coadă globală cu lista. La ieșirea din funcția `receive_message()` a RabbitMQ se verifică dacă este goală coada creată.

Se preia lista și se apelează funcția `handle_agent_alerts(list_alerts_for_agent)` prezentată în Anexa 1. În cadrul acesteia, se identifică agentul care a generat alertele și se preia numărul și detaliile sesiunii specifice adresei IP preluate din alerte, după cum arată linia 5.

Ulterior, se iterează prin alertele listei și, înainte de a identifica semnătura alertei, se realizează salvarea informațiilor de timpi într-o bază de date SQLite pentru a asigura persistența datelor și folosirea lor în calcule. După aceasta, se identifică semnătura alertei și, în funcție de caz, se intră pe o ramură condițională, se stabilește reacția și serverul trimite comanda către agent cu ajutorul funcției `shell()`, după cum arată linia 27.

Întrucât proiectul este în faza de prototip și deoarece modalitatea de stabilire a reacției se poate îmbunătăți folosind diverse mecanisme recomandate, s-a evidențiat în anexa amintită doar cazul pentru un atac de tip *SSH Brute Force*. În restul cazurilor, se asigură o salvare a alertelor pentru a ține evidența lor.

```

1 @repeat (every(SCHEDULED_INTERVAL) .seconds)
2 def main_consumer():
3     num_cpus = multiprocessing.cpu_count()
4     with ThreadPoolExecutor(max_workers=num_cpus) as executor:
5         _ = [executor.submit(receive_process_alerts_from_rabbitmq) for _ in
6              range(num_cpus)]

```

Listing 3.8. Funcția de consum apelată periodic

Fișierul `c2.py` conține logica de inițializare, persistență și comunicare a serverului cu agenții C2 instalati pe sistemele monitorizate. Agenții C2 se comportă ca niște servicii care rulează continuu și încearcă în mod periodic să se conecteze la server, realizând o conexiune de tip *callback*. Prin aceasta se asigură că în cazul închiderii serverul, agentul continuă să inițieze comunicarea până când reușește, adică în momentul în care se repornește programul server.

Funcția principală a fișierului este cea de pornire a serverului, evidențiată în Listing 3.9. Aceasta este apelată în fișierul `main.py`. Funcția începe cu asocierea de la linia 14 a `socket`-ului la adresa IP și portul dorit, urmând ca mai apoi să deschidă canalul pentru ascultarea conexiunilor. Așa cum s-a menționat și anterior, sunt create fire de execuție separate pentru funcționalitățile ce necesită rulare paralelă. Prin urmare, s-au definit două fire de execuție, una atribuindu-se rolul de a asculta continuu pentru conexiunile agenților, iar celuilalt rolul de a verifica periodic starea agenților Wazuh pentru a preveni închiderea lor în mod intentionat. Ulterior este pornită execuția `threads`.

O altă funcție prezentă în codul listat 3.9 este cea stabilită drept `target` pentru unul dintre firele de execuție discutate anterior. Mai exact, cel de ascultare continuă pentru conexiuni. Funcția arată faptul că, după primirea datelor noii conexiuni, acestea sunt salvate în liste globale pentru a fi accesibile în momentul în care se alege adresa IP, respectiv restul informațiilor de conectare ale unui sistem monitorizat.

```
1 def listening_for_connection():
2     global clients
3     while True:
4         C2_SOCKET.settimeout(1)
5         try:
6             target, ip = C2_SOCKET.accept()
7             TARGETS.append(target)
8             IPS.append(ip)
9             clients += 1
10        except Exception as e:
11            pass
12
13 def start_server():
14     C2_SOCKET.bind((SERVER_ADDRESS, SERVER_PORT))
15     C2_SOCKET.listen(5)
16
17     listening_thread = threading.Thread(target=listening_for_connection)
18     check_wazuh_thread = threading.Thread(target=check_wazuh_agents)
19
20     listening_thread.start()
21     check_wazuh_thread.start()
```

Listing 3.9. Funcția de pornire a serverului C2

Cea de-a două funcție folosită drept *target* este listată în 3.10. I se aplică decoratorul @repeat, fiind o funcție periodică ce implementează mecanismul de *heartbeat* detaliat în Capitolul 2. Aceasta iterează prin listele globale populate la linia 7 din Listing 3.9 și trimită comanda de verificare a statusului serviciului. Dacă se identifică textul clasic stării active, se trece la următorul agent. În caz contrar, se transmite comanda specifică de pornire a serviciului și se verifică din nou starea pentru a stabili dacă se generează eroare sau s-a repornit serviciul cu succes.

Funcția *shell()* din Anexa 2 conține funcțiile interne de trimitere a comenziilor și primire a răspunsurilor care utilizează JSON pentru serializarea și deserializarea datelor, respectiv pentru codificarea și decodificarea acestor obiecte JSON.

```
1 @repeat(every(SCHEDEDUL_INTERVAL).seconds)
2 def check_wazuh_agents():
3     logging.info("[+] [check_wazuh_agents()] INFO: Check Wazuh Agents Status")
4     length_of_targets = len(TARGETS)
5     i = 0
6     try:
7         while i < length_of_targets:
8             target = TARGETS[i]
9             ip = IPS[i]
10            response = shell(target, ip, CHECK_WAZUH_STATUS)
11            # Here can be multiple states of wazuh-agent: inactive, failed, activating,
12            # → reloading, maintenance
13            # I manage classic state of agent : inactive -> active
14            # In some cases agent can be in failed state, but just when is smth wrong
15            # → configured
16            if PREVENTION_STOPPING_AGENT_CHECK not in response:
17                logging.info(f"[+] [check_wazuh_agents()] INFO: Agent <{ip}> seems to be
18                # → inactive")
19
20                _ = shell(target, ip, ACTIVATE_WAZUH_AGENT)
21                response = shell(target, ip, CHECK_WAZUH_STATUS)
22
23            if PREVENTION_STOPPING_AGENT_CHECK not in response:
24                logging.error(f"[!] [check_wazuh_agents()] ERROR: Agent <{ip}> could not
25                # → be activated.")
```

```

22     else:
23         logging.info(f"[++][check_wazuh_agents()] INFO: Agent <{ip}>
24             ↳ successfully activated.")
25         i += 1
26
27 except Exception as e:
28     logging.error(f"[!] [check_wazuh_agents()] ERROR Failed to check to all
29             ↳ targets: {e}")

```

Listing 3.10. Verificare status agent Wazuh

Fișierul agent_C2.py conține codul necesar agentului C2. Una dintre cele mai importante funcții este persistence() listată în 3.11 care creează un serviciu utilizator din acest executabil pentru a permite pornirea acestuia odată cu pornirea sistemului. În acest mod se asigură persistența pe sistemul monitorizat și faptul că agentul va fi mereu activ pentru a iniția și a se conecta la server.

Funcția check_wazuh(command) lansează un proces separat de funcționare ușuală a serviciului astfel încât să ofere posibilitatea serverului să execute alte comenzi până la primirea răspunsului.

Funcția shell(), listată în Anexa 3, asigură implementarea funcționalităților în funcție de comanda venită de la server și trimit rezultatele către server. Se asigură opțiuni de navigare prin directoare, descărcare și încărcare de fișiere pe/din server, pornirea unor aplicații și verificarea rolului utilizatorului. În cazul în care se identifică o eroare de conexiune, se repeleză funcția de conectare pentru a reîncepe comunicarea cu serverul.

```

def persistence():
    if getattr(sys, 'frozen', False):
        script_location = sys.executable
    else:
        script_location = __file__

    location = os.path.expanduser("~/local/bin/victim")
    service_file = os.path.expanduser("~/config/systemd/user/victim.service")

    if not os.path.exists(location):
        os.makedirs(os.path.dirname(location), exist_ok=True)
        shutil.copyfile(script_location, location)
        os.chmod(location, 0o755)

    if not os.path.exists(service_file):
        os.makedirs(os.path.dirname(service_file), exist_ok=True)
        with open(service_file, 'w') as f:
            f.write(f"""
[Unit]
Description=Run this service at startup

[Service]
ExecStart={location}

[Install]
WantedBy=default.target
""")

    subprocess.call(['systemctl', '--user', 'enable', 'victim.service'])
    subprocess.call(['systemctl', '--user', 'start', 'victim.service'])

def check_wazuh(command):
    try:
        proc = subprocess.Popen(command[6:], 

```

```
        shell=True,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        stdin=subprocess.PIPE)
stdout, stderr = proc.communicate()
result = stdout + stderr
reliable_send(result)
except Exception as e:
    reliable_send(f"[!] ERROR executing heartbeat: {e}".encode())

```

Listing 3.11. Asigurare persistență agent C2

Fișierul constants.py conține datele despre comenzi de executat în linia de comandă, despre datele serverului și socket-ul pregătit pentru inițierea conexiunii către server.

Fișierul database_manage.py are rolul de a asigura funcționalitățile ce țin de managementul bazei de date în care sunt stocați timpii care au fost adăugați în alarme pe parcursul procesării acestora. Funcția de initializare este apelată în fișierul consumer_mq.py pentru a crea o conexiune nouă către baza de date pentru fiecare agent. Funcția de închidere a bazei de date este configuriată, cu ajutorul modulului atexit, să fie apelată odată cu închiderea întregului program. Funcțiile de inserare a noilor agenți identifică și a timpilor din alerte sunt implementate simplu, folosind sintaxa SQL.

Fișierul rabbit.py implementează, în acest program, funcționalitățile de primire a elementelor din coadă. Funcția receive_message() din Listing 3.12 realizează, în aceeași manieră ca în componenta Log Collector, crearea unei conexiuni, urmând ca mai apoi să apeleze funcția de callback on_received_message(...) unde se preia mesajul, se adaugă timpul de consumare în sub-documentul specific din fiecare alertă și se adaugă lista de alerte în coada globală creată. Ulterior se confirmă livrarea prin folosirea funcției confirm_delivery() din biblioteca pika.

```
def on_received_message(self, blocking_channel, deliver, properties, message):
    try:
        result = json.loads(message.decode('utf-8'))
        # Add timestamp for moment of consuming from queue
        add_publishing_timestamp = lambda alert: {**alert, TIMESTAMP_SUBDOCUMENT:
            {**alert[TIMESTAMP_SUBDOCUMENT], CONSUMPTION_FIELD:
                str(datetime.datetime.now())}}
        formatted_alerts = list(map(add_publishing_timestamp, result))

        ALERTS_QUEUE.put(formatted_alerts)
        blocking_channel.confirm_delivery()

    except json.JSONDecodeError as e:
        logging.error(f"[!] [on_received_message] ERROR: decode JSON: {e}")
        print(f"[!] [on_received_message] ERROR: decode JSON: {e}")

    except ALERTS_QUEUE.Full as e:
        logging.error(f"[!] [on_received_message] ERROR: Queue is full: {e}")
        print(f"[!] [on_received_message] ERROR: Queue is full: {e}")

    except Exception as e:
        logging.error(f"[!] [on_received_message] ERROR: {e}")
        print(f"[!] [on_received_message] ERROR: {e}")

    finally:
        blocking_channel.stop_consuming()
```

```

def receive_message(self):
    with pika.BlockingConnection(self.parameters) as connection:
        with connection.channel() as channel:
            method_frame, header_frame, body = channel.basic_get(self.config['queue'])
            if body is not None:
                self.on_received_message(channel, method_frame, None, body)
                channel.basic_ack(method_frame.delivery_tag)
            else:
                pass

```

Listing 3.12. Consum element din coada RabbitMQ

3.1.5.3. Comunicarea cu alte componente

După cum s-a detaliat și în arhitectura din Capitolul 2, componenta Command&Control comunică indirect cu Log Collector și în mod direct prin conexiuni multiple cu fiecare dintre proprii agenți instalati pe sistemele monitorizate. Comunicare cu aceștia se realizează prin al doilea canal de comunicare, primul fiind cel care asigură conexiunea dintre agentul Wazuh și serverul său. Astfel, se asigură un strat suplimentar de securitate, creând o soluție a cărei proiectare sugerează componente total distincte și a cărei implementare automatizează logica legăturilor dintre ele.

3.1.6. Componenta Endpoints

3.1.6.1. Descriere generală

Componenta Endpoints înglobează totalitatea mașinilor virtuale care au rolul: de a simula sistemele monitorizate de agentul Wazuh alături de IDS Suricata, de a executa instrucțiuni trimise de un Command&Control Server și de a fi atacate de instrumente specializate precum Metasploit sau Hydra. Fiecare mașină virtuală a fost configurată în același mod. Configurarea este detaliată în secțiunea următoare.

3.1.6.2. Structura codului/configurării

Pentru început, configurarea unui sistem a constat în instalarea IDS Suricata și configurarea sa pentru a monitoriza mașina virtuală conectată la VPN. IDS este un instrument care asigură monitorizarea fluxului de trafic care intră șiiese din rețea. Astfel, IDS poate fi configurat pe o singură mașină virtuală și configurat astfel încât să monitorizeze întreaga rețea. Totuși, în scop de exercițiu și luând în considerare implementarea unui prototip, s-a optat pentru instalarea IDS pe fiecare sistem monitorizat pentru verificarea constantă a funcționării mașinii virtuale în parametri normali. Această abordare poate fi înlocuită cu instalarea și configurarea unui singur IDS care să monitorizeze întreaga rețea. Se observă în Listing 3.13 configurarea rețelei, alături de modificarea interfeței monitorizate cu cea VPN din Listing 3.14. Pentru dezvoltarea regulilor de test și pentru a nu modifica fișierul inițial *suricata.rules*, am inclus un fișier personalizat în Listing 3.15.

```

vars:
    # more specific is better for alert accuracy and performance
address-groups:
    HOME_NET: "[ 10.177.186.5/24]"
    #HOME_NET: "[ 172.31.93.122/20]"
    #HOME_NET: "[192.168.0.0/16]"
    #HOME_NET: "[10.0.0.0/8]"
    #HOME_NET: "[172.16.0.0/12]"
    #HOME_NET: "any"

#EXTERNAL_NET: "!"$HOME_NET"
EXTERNAL_NET: "any"

```

Listing 3.13. Configurare adresă IP

```
#Shell
ubuntu@ip-172-31-93-122:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    ↓ qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default
    ↓ qlen 1000
        link/ether 12:66:fd:80:52:25 brd ff:ff:ff:ff:ff:ff
        inet 172.31.93.122/20 metric 100 brd 172.31.95.255 scope global dynamic eth0
            valid_lft 2909sec preferred_lft 2909sec
        inet6 fe80::1066:fdff:fe80:5225/64 scope link
            valid_lft forever preferred_lft forever
3: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 8921 qdisc noqueue state UNKNOWN group default
    ↓ qlen 1000
        link/none
        inet 10.177.186.5/24 scope global wg0
            valid_lft forever preferred_lft forever

#Suricata.yaml
# Linux high speed capture support
af-packet:
- interface: wg0 #eth0
```

Listing 3.14. Configurare interfață monitorizată - VPN

```
default-rule-path: /var/lib/suricata/rules

rule-files:
- /etc/suricata/rules/custom.rules
- suricata.rules
```

Listing 3.15. Adăugare fișier de reguli personalizat în configurarea Suricata

Următoarea configurare de pe un sistem monitorizat este cea a agentului Wazuh care, după rularea comenzi oferite în Wazuh *dashboard* și activarea serviciului, se realizează integrarea cu IDS Suricata, conform Listing 3.16.

```
<!--
Wazuh - Agent - Default configuration for ubuntu 22.04
More info at: https://documentation.wazuh.com
Mailing list: https://groups.google.com/forum/#!forum/wazuh
-->
<ossec_config>
    ...
    <localfile>
        <log_format>syslog</log_format>
        <location>/var/log/kern.log</location>
    </localfile>

    <localfile>
        <log_format>json</log_format>
        <location>/var/log/suricata/eve.json</location>
    </localfile>
</ossec_config>
```

Listing 3.16. Integrare Wazuh & Suricata

3.2. Dificultăți întâmpinate și modalități de rezolvare

3.2.1. Funcționarea corectă a EC2 cu VPN

Una dintre problemele întâmpinate pe parcursul implementării este reprezentată de creația corectă a grupurilor de securitate în EC2 pentru a redirecționa traficul prin tunelul asigurat de VPN. În acest mod, peste regulile de bază în cazul creării unui astfel de grup, precum comunicația HTTP, SSH și HTTPS, se aplică grupul de securitate al VPN care permite pentru tot traficul redirecționarea către el, conform Figurii 3.5.

The screenshot shows the AWS Security Groups interface. It displays two tables of rules:

- Inbound rules:** This table lists four rules. All rules allow traffic from 0.0.0.0/0 to the security group. The protocols and port ranges are All. The source is 0.0.0.0/0. The security groups associated with these rules are Endpoint2Rules.
- Outbound rules:** This table lists one rule. It allows traffic from the security group to 0.0.0.0/0. The protocol and port range are All. The destination is 0.0.0.0/0. The security group associated with this rule is Endpoint2Rules.

Figura 3.5. Grup de securitate pentru sistem monitorizat

Se observă în Figura 3.5 că există o regulă a cărei sursă este identificatorul grupului de securitate VPN, grup afișat în Figura 3.6. În cadrul acestuia, există o regulă ce permite conectarea SSH, inclusiv la severul VPN, în scopul administrării acestuia. De asemenea, o altă regulă importantă este cea de tip Custom UDP care permite traficul pe portul implicit al VPN Wireguard, 51820, și asigură că serverul VPN poate accepta conexiuni de la alți clienți, după cum se poate observa și în fișierul de configurare 3.17. O altă regulă din grupul de securitate din Figura 3.6 este cea care permite traficul dintre 2 instanțe EC2: sever VPN - client VPN.

The screenshot shows the details of a security group named "WIREGUARD".

Details:

- Security group name: WIREGUARD
- Owner: 381491852522
- Security group ID: sg-0ae644709a6e087a7
- Description: Allow traffic through VPN server
- VPC ID: vpc-0d4655909042f4ad2
- Inbound rules count: 4 Permission entries
- Outbound rules count: 1 Permission entry

Inbound rules (4):

IP rule ID	IP version	Type	Protocol	Port range	Source	Description
:fdd72eba6	IPv4	SSH	TCP	22	0.0.0.0/0	-
435996e40	-	All traffic	All	All	sg-05c2014147af1cadd / Endpoint2Rules	-
055c7b542	IPv4	Custom UDP	UDP	51820	0.0.0.0/0	-
icf1c8e42e	-	All traffic	All	All	sg-008d4a2e0a5c29018 / Endpoint1Rules	-

Figura 3.6. Grup de securitate pentru server VPN

```
[Interface]
PrivateKey = *****
Address = 10.177.186.5/24
DNS = 9.9.9.9, 149.112.112.112

[Peer]
PublicKey = O4UgYr5kP3GbfKNP1blFFtVQAp6SpChIvCBmL9fgg38=
PresharedKey = *****
Endpoint = 3.227.220.68:51820
AllowedIPs = 0.0.0.0/0, ::0/0
```

Listing 3.17. Configurare VPN Client pe *endpoint*

3.2.2. Alte dificultăți întâmpinate

Pe parcursul implementării soluției s-au întâmpinat o serie de probleme ce au constituit alegerea modului de abordare în vederea implementării unor anumite funcționalități. Așa cum a fost descris și în secțiunea anterioară, s-a observat dificultatea mașinilor virtuale de a comunica între ele. Pentru această situație, s-a ales utilizarea unui VPN pe fiecare mașină pentru a crea o rețea privată în care sunt incluse toate sistemele cu care se lucrează. Prin urmare, rezolvarea a constat în instalarea unui VPN, mai exact Wireguard.

O altă problemă identificată și detaliată anterior a fost eliminarea secvențialității în cadrul procesării alarmelor, dar și asigurarea unor funcționalități care să ruleze concurent pentru a nu bloca firul principal de execuție. Aceasta a fost rezolvată folosind metoda de la linia 24 din Listing 3.2.

3.3. Idei originale, soluții noi

3.3.1. Componența Log SIEM

3.3.1.1. Rol uzual

Instrumentele SIEM reprezintă platforme de monitorizare și gestionare a securității, folosite pentru a colecta, analiza și gestiona evenimentele de securitate într-un mediu de rețea, care pot integra alte sisteme de securitate, precum firewall-urile, IDS-urile și software-ul antivirus [19].

3.3.1.2. Scopul abordării personalizate

Una dintre ideile personalizate folosite în implementarea soluției este cea de utilizare sistem SIEM drept container de transfer pentru alertele generate de IDS Suricata în vederea eliminării dependenței de specificitatea unui anumit instrument SIEM. În capitolul precedent s-au detaliat atât avantajele acestei abordări, cât și dezavantajele sale.

Pentru implementarea soluției la nivel de prototip, însă, se consideră a fi o abordare corespunzătoare. Oferă posibilitatea de a păstra soluția generală în continuare, integrând IDS cu alte tehnici pentru acoperirea unei plaje mai mari de atacuri.

De asemenea, oferă opțiunea de a particulariza proiectul, folosind instrumente SIEM care oferă acces la diverse funcționalități: Wazuh, instrumentul utilizat în soluție, este o platformă *open-source* care poate și integrată în sisteme SIEM mai complexe. IBM QRadar, pe de altă parte, este un instrument mai complex care oferă funcționalități *User Behavior Analytics*.

Implementarea este realizată prin utilizarea tuturor funcționalităților oferite de Wazuh, însă preluarea doar a alertelor generate de IDS Suricata, după cum se observă în configurația *query* pentru *request* trimis către API ELK, conform Listing 3.18.

```

def generate_elk_query(...):
    query = {
        "bool": {
            "must": [
                {
                    "term": {"agent.id": agent_id}
                },
                {
                    "term": {"rule.groups": "suricata"}
                },
                {
                    "range": {
                        "@timestamp": {
                            "gte": time_range_start,
                            "lte": time_range_end
                        }
                    }
                }
            ]
        }
    }
    request = {
        "query": query,
        "size": page_size,
        "sort": [
            {"sort_field": {"order": sort_order}}
        ]
    }
    return request

```

Listing 3.18. Filtrare alerte IDS Suricata

3.3.2. Componenta Command&Control

3.3.2.1. Rol uzual

Un server Command&Control este o metodă pe care atacatorii o folosesc pentru a comunica, prin conexiuni de tip *callback*, cu sistemele compromise. Atacatorii urmăresc persistența pe dispozitive pentru a descoperi date, a instala programe malicioase ori pentru a iniția atacuri de tip *Denial of Service*.

3.3.2.2. Scopul abordării personalizate

O altă idee personalizată este modalitatea de utilizare a componentei Command&Control. Mai exact, aceasta este folosită în scop defensiv, având rolul de a reacționa la atacurile identificate pe sistemele monitorizate și de a preveni închiderea sau alterarea agentului Wazuh care asigură preluarea alertelor. S-a ales această abordare datorită funcționalităților de care dispune un astfel de server și care pot fi extinse și personalizate în funcție de nevoi. Așadar, în loc de comenzi pentru exfiltrarea datelor, de exemplu, se folosesc comenzi cu rol pozitiv pentru asigurarea funcționării corespunzătoare a soluției create în Listing 3.19.

```

CHECK_WAZUH_STATUS = "wazuh systemctl status wazuh-agent"
ACTIVATE_WAZUH_AGENT = "wazuh sudo systemctl start wazuh-agent"
PREVENTION_STOPPING_AGENT_CHECK = "Active: active (running)"
CHECK_BRUTEFORCE_SSH_SERVICE = "Active: inactive (dead)"

```

Listing 3.19. Exemplu de constante pentru verificarea statusului agentului Wazuh

3.3.3. Componența Endpoints

3.3.3.1. Rol uzual

Rolul uzual al componentelor Endpoints este de a asigura un număr variabil de mașini virtuale care pot servi drept sisteme pe care se pot rula diverse programe.

3.3.3.2. Scopul abordării personalizate

Modul în care se folosesc în cadrul proiectului este uzuală. Ideea personalizată apare, însă, în modalitatea în care agenții de pe aceste *endpoints* întrețin comunicarea cu serverele proprii. Acest aspect a fost evidențiat atât în descrierea implementării fișierelor din secțiunile anterioare, cat și în Figura 2.3. Astfel, în loc de implementarea unui singur agent care să aibă ambele roluri, s-au utilizat funcționalitățile agentului Wazuh pentru preluarea corespunzătoare a alarmelor și s-a implementat un mecanism de *backdoor* cu rol de reacție, fiind mai dificil de depistat.

3.4. Comunicarea cu alte sisteme și salvarea/stocarea informațiilor

În privința stocării informațiilor cu care se lucrează în proiect, există două mecanisme care îndeplinește acest rol. Primul dintre ele este coada de mesaje RabbitMQ care asigură persistența datelor pana la consumul lor, iar al doilea îl constituie utilizarea bazei de date SQLite pentru stocarea timpilor importanți în procesarea alertelor: începutul procesării, începutul publicării, momentul consumului. Baza de date conține două tabele simple, unul pentru stocarea tuturor agenților și a informațiilor lor, iar al doilea tabel conține timpii menționați anterior. Aceste date vor fi folosite în 4 în vederea observării timpului mediu de stocare în coadă a alertelor fiecărui agent, timpului mediu de procesare pentru alerte și alte informații considerate importante.

3.5. Funcționarea sistemului

3.5.1. Colectare periodică alerale în componența Log Collector

Colectarea alertelor este evidențiată în fișierul log al IDS Suricata și ieșirea consolei programului în Figura 3.7.

LICENTA — ubuntu@ip-172-31-93-122: ~ -- ssh -i vockey.pem ubuntu@10.177.186.5 — 119x31

```

ubuntu@ip-172-31-93-122:~$ tail -n 8 /var/log/suricata/fast.log
06/30/2024-17:53:44.423486 [*] [1:2003688:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:37692 -> 10.177.186.5:22
06/30/2024-17:53:44.756989 [*] [1:2003688:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:37636 -> 10.177.186.5:22
06/30/2024-17:53:44.756989 [*] [1:2003688:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:37636 -> 10.177.186.5:22
06/30/2024-17:53:44.756989 [*] [1:2003688:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:37636 -> 10.177.186.5:22
06/30/2024-17:53:44.756989 [*] [1:2003688:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:37624 -> 10.177.186.5:22
06/30/2024-17:53:44.756989 [*] [1:2003688:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:37624 -> 10.177.186.5:22
06/30/2024-17:54:39.739935 [*] [1:2003688:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:42208 -> 10.177.186.5:22
06/30/2024-17:54:39.745451 [*] [1:2003688:7] ET SCAN LibSSH Based frequent SSH Connections Likely BruteForce Attack [**] [Classification: Attempted Adminstrator Privilege Gain] [Priority: 1] {TCP} 10.177.186.9:42208 -> 10.177.186.5:22
06/30/2024-17:54:39.887588 [*] [1:2003688:7] ET SCAN LibSSH Based frequent SSH Connections Likely BruteForce Attack [**] [Classification: Attempted Adminstrator Privilege Gain] [Priority: 1] {TCP} 10.177.186.9:42208 -> 10.177.186.5:22

```

Wazuh_Server — main.py

```

while True:
    main.main()

```

Run: main.main

```

    "test_latency": {
        "start_processing": "2024-06-30 20:54:09.713201",
        "start_publishing": "2024-06-30 20:54:09.919608"
    },
    "data": {
        "dest_ip": "10.177.186.5",
        "timestamp": "2024-06-30T17:53:44.756989+0000",
        "dest_port": "22",
        "src_port": "37636",
        "src_ip": "10.177.186.9",
        "proto": "TCP",
        "event_type": "alert",
        "alert": {
            "signature_id": "2003688",
            "action": "allowed",
            "severity": "2",
            "signature": "ET SCAN Potential SSH Scan OUTBOUND",
            "category": "Attempted Information Leak"
        }
    },
    "agent": {
        "id": "002",
        "name": "Endpoint2",
        "ip": "10.177.186.5"
    }
}

```

Figura 3.7. Colectare periodică alerme

3.5.2. Reacție agent Wazuh închis

Prevenția închiderii definitive a agentilor Wazuh în Figura 3.8.

```

ubuntu@ip-172-31-93-122:~$ sudo systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
  Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
  Active: active (running) since Sun 2024-06-30 16:12:12 UTC; 1h 46min ago
    Process: 2730 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
   Tasks: 31 (limit: 1120)
  Memory: 66.9M
    CPU: 20.188s
  CGroup: /system.slice/wazuh-agent.service
          ├─2745 /var/ossec/bin/wazuh-execd...
          ├─2765 /var/ossec/bin/wazuh-agentsd...
          ├─2782 /var/ossec/bin/wazuh-syscheckd...
          ├─2796 /var/ossec/bin/wazuh-logcollector...
          └─2808 /var/ossec/bin/wazuh-modulesd...

Jun 30 16:12:05 ip-172-31-93-122 systemd[1]: Starting Wazuh agent...
Jun 30 16:12:05 ip-172-31-93-122 env[2730]: Starting Wazuh v4.7.3...
Jun 30 16:12:06 ip-172-31-93-122 env[2730]: Started wazuh-execd...
Jun 30 16:12:07 ip-172-31-93-122 env[2730]: Started wazuh-agentsd...
Jun 30 16:12:08 ip-172-31-93-122 env[2730]: Started wazuh-syscheckd...
Jun 30 16:12:09 ip-172-31-93-122 env[2730]: Started wazuh-logcollector...
Jun 30 16:12:10 ip-172-31-93-122 env[2730]: Started wazuh-modulesd...
Jun 30 16:12:11 ip-172-31-93-122 env[2730]: Completed...
Jun 30 16:12:12 ip-172-31-93-122 env[2730]: Started wazuh-agent.
Jun 30 16:12:12 ip-172-31-93-122 systemd[1]: Started Wazuh agent.

ubuntu@ip-172-31-93-122:~$ sudo systemctl stop wazuh-agent
ubuntu@ip-172-31-93-122:~$ sudo systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
  Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
  Active: inactive (dead) since Sun 2024-06-30 17:58:27 UTC; 19s ago
    Process: 2730 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
   Tasks: 0 (limit: 1120)
  Memory: 0B
    CPU: 0.301s

Jun 30 17:58:26 ip-172-31-93-122 systemd[1]: Stopping Wazuh agent...
Jun 30 17:58:26 ip-172-31-93-122 env[4145]: Killing wazuh-modulesd...
Jun 30 17:58:26 ip-172-31-93-122 env[4145]: Killing wazuh-logcollector...
Jun 30 17:58:26 ip-172-31-93-122 env[4145]: Killing wazuh-syscheckd...
Jun 30 17:58:27 ip-172-31-93-122 env[4145]: Killing wazuh-agentsd...
Jun 30 17:58:27 ip-172-31-93-122 env[4145]: Wazuh v4.7.3 Stopped
Jun 30 17:58:27 ip-172-31-93-122 systemd[1]: wazuh-agent.service: Deactivated successfully.
Jun 30 17:58:27 ip-172-31-93-122 systemd[1]: wazuh-agent.service: Consumed 20.301s CPU time.

● wazuh-agent.service - Wazuh agent
  Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
  Active: activating (start) since Sun 2024-06-30 17:59:14 UTC; 2s ago
  Control PID: 4269 (wazuh-control)
  Main PID: 4269
  Memory: 77ms
    CPU: 77ms
  CGroup: /system.slice/wazuh-agent.service
          └─4209 /bin/sh /var/ossec/bin/wazuh-control start
            ├─4233 /var/ossec/bin/wazuh-execd...
            ├─4243 /var/ossec/bin/wazuh-agentsd...
            ├─4244 sleep 1

Jun 30 17:59:14 ip-172-31-93-122 systemd[1]: Starting Wazuh agent...
Jun 30 17:59:14 ip-172-31-93-122 env[4209]: Starting Wazuh v4.7.3...
Jun 30 17:59:15 ip-172-31-93-122 env[4209]: Started wazuh-execd...
ubuntu@ip-172-31-93-122:~$ 
```

```

C2_Server - constants.py
Project: C2_Server
Run: main
Wait 29 /30 seconds
Wait 30 /30 seconds
Wait 31 /30 seconds

[+] [check_wazuh_agents()] INFO: Check Wazuh Agents Status
[+] INFO: Waiting for c2_agent response...
AGENT ['10.177.186.5', 52492] has responded:
*****
o wazuh-agent.service - Wazuh agent
  Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
  Active: active (running) since Sun 2024-06-30 17:58:27 UTC; 46s ago
  Process: 2730 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, s
  Process: 4145 ExecStop=/usr/bin/env /var/ossec/bin/wazuh-control stop (code=exited, sta
  CPU: 20.301s

Jun 30 17:58:26 ip-172-31-93-122 systemd[1]: Stopping Wazuh agent...
Jun 30 17:58:26 ip-172-31-93-122 env[4145]: Killing wazuh-modulesd...
Jun 30 17:58:26 ip-172-31-93-122 env[4145]: Killing wazuh-logcollector...
Jun 30 17:58:26 ip-172-31-93-122 env[4145]: Killing wazuh-syscheckd...
Jun 30 17:58:27 ip-172-31-93-122 env[4145]: Killing wazuh-agentsd...
Jun 30 17:58:27 ip-172-31-93-122 env[4145]: Wazuh v4.7.3 Stopped
Jun 30 17:58:27 ip-172-31-93-122 systemd[1]: wazuh-agent.service: Deactivated successfully.
Jun 30 17:58:27 ip-172-31-93-122 systemd[1]: wazuh-agent.service: Consumed 20.301s CPU time.

*****
[+] [check_wazuh_agents()] INFO: Agent <'10.177.186.5', 52492> seems to be inactive
[+] INFO: Waiting for c2_agent response...
[+] INFO: Waiting for c2_agent response...
[+] [check_wazuh_agents()] INFO: Agent <'10.177.186.5', 52492> successfully activated.
***** 
```

Figura 3.8. Prevenție închidere agent Wazuh

3.5.3. Atac Brute Force SSH

Reacție specifică în funcție de semnătura alertei primite în Figura 3.9.

```

ubuntu@ip-172-31-93-122:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Drop-In: /usr/lib/systemd/system/ssh.service.d
            └── ec2-instance-connect.conf
  Active: active (running) since Sun 2024-06-30 18:03:50 UTC; 5s ago
    Docs: man:sshd(8)
          man:sshd_config(5)
  Process: 5116 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
  Main PID: 5117 sshd[1]
  Tasks: 1 (limit: 1120)
  Memory: 2.9M
    CPU: 23ms
  CGroup: /system.slice/ssh.service
          └─5117 sshd[1]: /usr/sbin/sshd -D -o AuthorizedKeysCommand /usr/share/ec2-instance-connect/eic_run_authoriz

Jun 30 18:03:50 ip-172-31-93-122 systemd[1]: Starting OpenBSD Secure Shell server...
Jun 30 18:03:50 ip-172-31-93-122 sshd[5117]: Server listening on 0.0.0.0 port 22.
Jun 30 18:03:50 ip-172-31-93-122 sshd[5117]: Server listening on :: port 22.
Jun 30 18:03:50 ip-172-31-93-122 systemd[1]: Started OpenBSD Secure Shell server.

line 1-19/19

ubuntu@ip-172-31-93-122:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
ubuntu@ip-172-31-93-122:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Drop-In: /usr/lib/systemd/system/ssh.service.d
            └── ec2-instance-connect.conf
  Active: inactive (dead) since Sun 2024-06-30 18:05:26 UTC; 28s ago
    Docs: man:sshd(8)
          man:sshd_config(5)
  Process: 5116 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
  Process: 5117 ExecStart=/usr/sbin/sshd -D -o AuthorizedKeysCommand /usr/share/ec2-instance-connect/eic_run_authoriz
  Main PID: 5117 (code=exited, status=0/SUCCESS)
  CPU: 270ms

Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: Received signal 15; terminating.
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]:
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: error: maximum authentication attempts exceeded for invalid user 3d from 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:26 ip-172-31-93-122 sshd[5120]: Disconnecting invalid user 3d 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: PAM 5 more authentication failures; logname= u
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: PAM service(sshd) ignoring max retries; 6 > 3
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: error: maximum authentication attempts exceeded for invalid user 3d from 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:26 ip-172-31-93-122 sshd[5120]: Disconnecting invalid user 3d 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: PAM 5 more authentication failures; logname= u
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: PAM service(sshd) ignoring max retries; 6 > 3
Jun 30 18:05:26 ip-172-31-93-122 sshd[5120]: Disconnecting invalid user 3d 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:26 ip-172-31-93-122 sshd[5120]: Received signal 15; terminating.
Jun 30 18:05:26 ip-172-31-93-122 systemd[1]: Stopping OpenBSD Secure Shell server...
Jun 30 18:05:26 ip-172-31-93-122 systemd[1]: ssh.service: Deactivated successfully.

line 1-22/22 (END)

ubuntu@ip-172-31-93-122:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
  56 3368 DROP   all  --  *   * 10.177.186.9  0.0.0.0/0
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
  0   0 DROP   all  --  *   * 0.0.0.0/0 10.177.186.9 
```

```

C2_Server - constants.py
Project: C2_Server
Run: main
[!] INFO : SSH Brute Force Detected
[+] INFO: Waiting for c2_agent response...
C2 Agent has responded:
*****
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
  0   0 DROP   all  --  *   * 10.177.186.9  0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
  0   0 DROP   all  --  *   * 0.0.0.0/0 10.177.186.9

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
  0   0 DROP   all  --  *   * 0.0.0.0/0 10.177.186.9
o ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Drop-In: /usr/lib/systemd/system/ssh.service.d
            └── ec2-instance-connect.conf
  Active: active (running) since Sun 2024-06-30 18:05:26 UTC; 26ms ago
    Docs: man:sshd(8)
          man:sshd_config(5)
  Process: 5117 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
  Process: 5117 ExecStart=/usr/sbin/sshd -D -o AuthorizedKeysCommand /usr/share/ec2-instance-connect/eic_run_authoriz
  Main PID: 5117 (code=exited, status=0/SUCCESS)
  CPU: 270ms

Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: Received signal 15; terminating.
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]:
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: error: maximum authentication attempts exceeded for invalid user 3d from 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:26 ip-172-31-93-122 sshd[5120]: Disconnecting invalid user 3d 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: PAM 5 more authentication failures; logname= u
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: PAM service(sshd) ignoring max retries; 6 > 3
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: error: maximum authentication attempts exceeded for invalid user 3d from 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:26 ip-172-31-93-122 sshd[5120]: Disconnecting invalid user 3d 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: PAM 5 more authentication failures; logname= u
Jun 30 18:05:19 ip-172-31-93-122 sshd[5120]: PAM service(sshd) ignoring max retries; 6 > 3
Jun 30 18:05:26 ip-172-31-93-122 sshd[5120]: Disconnecting invalid user 3d 10.177.186.9 port 57484: Too many authentication failures; logname= u
Jun 30 18:05:26 ip-172-31-93-122 sshd[5120]: Received signal 15; terminating.
Jun 30 18:05:26 ip-172-31-93-122 systemd[1]: Stopping OpenBSD Secure Shell server...
Jun 30 18:05:26 ip-172-31-93-122 systemd[1]: ssh.service: Deactivated successfully.

line 1-22/22 (END)

Externally added files can be added to Git // View File... (28.06.2024, 09:51) 224:8 LF UTF-8 4 spaces Python 3.11 C2_Server 
```

Figura 3.9. Reacție atac SSH Brute Force

Capitolul 4. Testarea aplicației și rezultate experimentale

4.1. Punerea în funcțiune/lansarea aplicației

Punerea în funcțiune a proiectului presupune o serie de pași care pregătesc mediul de lucru și pornesc programele necesare. Pentru început, trebuie pornit mediul de laborator care oferă acces la consola AWS EC2 și mașina virtuală pe care rulează serverul Wazuh care va fi detaliată ulterior. Este important să fie verificată starea instanțelor create astfel încât să se evite situațiile în care se opresc brusc și se deconectează și blochează consola. Starea activă a instanțelor este evidențiată în Figura 4.1.

Instances (6) Info								
Instance state (client) != terminated			All states			Actions		
<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
<input type="checkbox"/>	Endpoint2	i-03e4a93bbfd1f0af	Running	t2.micro	Initializing	View alarms +	us-east-1a	ec2-54-88-96-169.com...
<input type="checkbox"/>	Endpoint1	i-03c05869453352fc4	Running	t2.micro	Initializing	View alarms +	us-east-1a	ec2-35-170-202-193.co...
<input type="checkbox"/>	Pentest_Server	i-0a3b1b644c8dba27c	Running	t2.micro	Initializing	View alarms +	us-east-1a	ec2-52-87-223-145.co...
<input type="checkbox"/>	Endpoint3	i-077df455ad8b87359	Running	t2.micro	Initializing	View alarms +	us-east-1c	ec2-34-230-34-91.com...
<input type="checkbox"/>	VPNServer	i-02bdb3b3b6d0f16fb	Running	t4g.nano	Initializing	View alarms +	us-east-1b	ec2-3-227-220-68.com...
<input type="checkbox"/>	Endpoint4	i-0738bccd01c5901c6	Running	t2.micro	Initializing	View alarms +	us-east-1a	ec2-3-86-194-120.com...

Figura 4.1. Stare activă a instanțelor EC2

Pentru a putea realiza conexiunea SSH cu sistemele monitorizate, se realizează, în prima fază, conectarea SSH la serverul VPN cu adresa statică Elastic IP configurată la crearea instanței. A fost aleasă această abordare datorită persistenței și stabilității adresei la fiecare repornire a instanței. După verificarea disponibilității serviciului VPN, se activează clientul VPN de pe computerul principal și se realizează conexiunile la instancele EC2. Se verifică, se asemenea, starea serviciilor care vor fi folosite în rularea prototipului, după cum se observă în Figura 4.2.

```
Last LogIn: Sun Jun 30 18:18:38 2024 [from 10.177.186.2]
[ubuntu@ip-172-0-1-1:~]$ sudo systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
   Loaded: loaded (/usr/lib/systemd/system/wazuh-agent.service; enabled; preset: enabled)
   Active: active (running) since Sun 2024-06-30 14:13:29 UTC; 3h 58min ago
     Process: 597 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
    Tasks: 31 (limit: 1128)
   Memory: 25.4M (peak: 225.0M)
      CPU: 20.699s
     CGroup: /sys/fs/cgroup/system.slice/wazuh-agent.service
             └─ 597 /var/ossec/bin/wazuh-execd
                 ├─ 644 /var/ossec/bin/wazuh-agent
                 ├─ 682 /var/ossec/bin/wazuh-syscheckd
                 ├─ 699 /var/ossec/bin/wazuh-logcollector
                 ├─ 748 /var/ossec/bin/wazuh-modulesd
                 ├─ 768 /var/ossec/bin/wazuh-dulesd
Jun 30 14:13:21 ip-172-31-35-88 systemd[1]: Starting wazuh-agent.service - Wazuh agent...
Jun 30 14:13:22 ip-172-31-35-88 env[597]: Starting Wazuh v.7.3...
Jun 30 14:13:23 ip-172-31-35-88 env[597]: Started wazuh-execd...
Jun 30 14:13:24 ip-172-31-35-88 env[597]: Started wazuh-agentd...
Jun 30 14:13:24 ip-172-31-35-88 env[597]: Started wazuh-syscheckd...
Jun 30 14:13:24 ip-172-31-35-88 env[597]: Started wazuh-logcollector...
Jun 30 14:13:27 ip-172-31-35-88 env[597]: Started wazuh-modulesd...
Jun 30 14:13:29 ip-172-31-35-88 env[597]: Completed...
Jun 30 14:13:29 ip-172-31-35-88 systemd[1]: Started wazuh-agent.service - Wazuh agent.
[ubuntu@ip-172-0-1-1:~]$
```

```
Last LogIn: Sun Jun 30 18:18:38 2024 [from 10.177.186.2]
[ubuntu@ip-172-0-1-1:~]$ sudo systemctl status suricata.service
● suricata.service - Suricata Network IPS
   Loaded: loaded (/usr/lib/systemd/system/suricata.service; enabled; preset: enabled)
   Active: active (running) since Sun 2024-06-30 14:13:22 UTC; 3h 58min ago
     Docs: man:suricata(8)
           https://suricata.io/documentation/
   Process: 571 ExecStart=/usr/bin/suricata -D --af-packet -c /etc/suricata/suricata.yaml --pidfile /run/suricata.pid
   Main PID: 576 (Suricata-Main)
      Tasks: 433 (peak: 445.4M)
     Memory: 413.1M
        CPU: 1min 36.631s
```

Figura 4.2. Verificare stare activă pentru servicii

Punerea în funcțiune a sistemelor monitorizate a fost realizată. Acum atenția este îndreptată spre mașina virtuală locală ce a fost deschisă odată cu mediul de laborator. Tot la deschidere s-a realizat și autentificarea. În cadrul acestelui se verifică serviciile stivei ELK, precum este Logstash. Rezultatul trebuie să fie similar cu ceea ce se observă în terminal în Figura 3.3 din Capitolul 3.

Mediul virtual este funcțional conform imaginilor. Acum se pot deschide programele componentelor Log Collector, respectiv Command&Control. În Figura 4.3 se observă funcționarea acestora, cu mențiunea că alertele afișate în consolă provin de la un atac *SSH Brute Force* realizat în prealabil.

The screenshot shows two PyCharm project windows side-by-side. The left window, titled 'Wazuh_Server - log_elk_collector.py', contains Python code for a log collector. The right window, titled 'C2_Server - consumer_mq.py', contains Python code for a command-and-control server. Both windows show the code, file navigation, and run configurations.

Figura 4.3. Rulare din IDE PyCharm a Log Collector și Command&Control

4.2. Testarea sistemului (hardware/software)

4.2.1. Planul urmat în realizarea testării

Scopul lucrării a fost de a realiza implementarea arhitecturii proiectului din Capitolul 2 la nivel de prototip funcțional pentru a observa fezabilitatea unei astfel de integrări de tehnici și tehnologii. Odată ce implementarea a fost terminată, s-a dorit reluarea observării rezultatelor astfel încât să se concluzioneze avantajele și dezavantajele acesteia. Scopul testării este de a se asigura un set de rezultate valide ce dovedesc funcționalitatea principală a soluției: reacția la atacuri identificate pe un set de sisteme monitorizate. Testele se vor axa pe validarea corectitudinii principalelor idei personalizate, dar și pe funcționarea generală corespunzătoare.

4.2.2. Testare pentru simplificarea formatului alertelor

4.2.2.1. Scopul testului

Testul a vizat observarea structurii mai multor tipuri de alerte generate de IDS Suricata și stabilirea unui format standard simplificat în care s-au reținut doar datele considerate necesare. Alertele prestatibile conțin o multitudine de date într-o varietate de tipuri de structurare a obiectelor JSON, iar acest lucru reprezintă procesarea, în acest moment inutilă, a unor date.

4.2.2.2. Modalitatea de testare

Modalitatea de testare este cea manuală, prin metoda observației, testul începând cu analiza structurii alertelor. S-au observat datele care sunt comune și s-a stabilit nivelul lor de importanță, apoi s-a creat o constantă cu rol de *template* pe baza căreia s-au simplificat alertele. Mediul de testare îl reprezintă: fișierele IDS Suricata care conțin declararea alertelor, fișierele de jurnalizare precum eve.json pentru observarea structurilor, precum și codul ce constituie implementarea procesului de simplificare, prezent în Log Collector.

4.2.2.3. Instrumente folosite

S-au folosit IDE PyCharm pentru implementarea codului, agent Wazuh pentru colectarea și transferul alertelor către server și API ELK pentru preluarea alertelor spre simplificare.

4.2.3. Testare pentru prevenția închiderii agentului Wazuh

4.2.3.1. Scopul testului

Soluția propusă poate fi împărțită conceptual în 2 părți: cea de colectare a alertelor și cea de reacție la acestea. Partea de reacție este implementată ca un mecanism de *backdoor* și rulează ca un proces în *background*. Partea de colectare este implementată folosind agentul instrumentului Wazuh pentru colectare. Wazuh este o platformă *open-source*, cunoscută, a cărei documentație este publică. Vulnerabilitățile sale pot fi studiate, ceea ce înseamnă că, în cazul unui atac care vizează alterarea de date, agentul Wazuh poate fi vizat pentru a fi închis sau modificat. Scopul testului este de a observa funcționarea corectă în situația în care un agent SIEM este închis: repornirea acestuia.

4.2.3.2. Modalitatea de testare

Testul a fost realizat în contextul rulării automatizate a proiectului, prin oprirea manuală sau ștergerea agentului Wazuh și observarea reactivării sale automate sau a generării alertei în consolă.

4.2.3.3. Descrierea cazurilor de test

Testul presupune urmărirea a 2 cazuri: cel în care doar se închide serviciul agentului Wazuh și cel în care agentul este șters, alterat. În primul caz se reacționează automat cu simpla lui deschidere, iar în al doilea caz se semnalează o eroare în fișierul de jurnalizare al componentei Command&Control. Al doilea caz poate fi modificat pentru a urmări posibilele schimbări ale fișierelor serviciului, istoricul închiderii sistemului prin monitorizarea fișierelor de jurnalizare de sistem, dar și a serviciului.

4.2.3.4. Instrumente folosite

Ca și instrumente s-au folosit IDE PyCharm pentru implementarea codului, agent Wazuh pentru testarea activării sale automate și comenzi specifice.

4.2.4. Testare pentru reacția diferențiată la diferite tipuri de alarme

4.2.4.1. Scopul testului

O reacție automatizată presupune lipsa nevoii de intervenție manuală. Totuși, aceste reacții trebuie stabilite în consecință cu tipul de atac. Scopul testului este de a observa identificarea corectă a atacului de către IDS Suricata, identificarea reacției potrivite, primirea răspunsului corespunzător de la agent C2 în urma execuției comenzielor primite și observarea atacului opriți în urma reacției luate.

4.2.4.2. Modalitatea de testare

Testul a fost realizat în contextul rulării automatizate a proiectului prin simularea unui atac de tip *SSH Brute Force* cu ajutorul instrumentului Hydra. Se pornește atacul cu o comandă simplă, de tipul: `hydra -L file\ usernames.txt -P file\ pass.txt ssh://IP -t 8`. Se observă în consolă identificarea atacului, trimitera comenzi, răspunsul venit de la agent. Se analizează răspunsul pentru a identifica textul așteptat și se stabilește dacă s-a executat reacția. În scurt timp se observă și în consola instanței de pe care a fost executat atacul că acesta se oprește.

4.2.4.3. Descrierea cazurilor de test

Cazurile de test constau în observarea reacției la diferite tipuri de atacuri. În contextul unui prototip, reacțiile distincte sunt limitate. Din acest motiv, restul atacurilor care nu au fost tratate diferențiat sunt înregistrate într-un fișier spre analiza lor.

4.2.4.4. Instrumente folosite

Hydra este unul utilizat în atacuri specialize pe *Brute Force* și este preferat datorită flexibilității sale de utilizare. Permite suport pentru protocoale precum HTTP/HTTPS, FTP, SSH. Funcționează pe baza încercărilor de tipul nume utilizator - parolă.

4.2.5. Testare pentru observarea timpilor de procesare al alertelor

4.2.5.1. Scopul testului

În cadrul proiectării unei soluții de securitate, latențele mari în timpul rulării implementării sunt un indicator că metoda trebuie îmbunătățită. Scopul testării timpilor înregistrării în punctele importante ale soluției, cum ar fi momentul începerii procesării, al publicării alertei, și al consumului reprezintă un indicator de bază pentru a oferi o evaluare a soluției.

4.2.5.2. Modalitatea de testare

Testele din această secțiune pot fi realizate separat de rularea proiectului, întrucât toți acești timpi sunt stocați într-o bază de date SQLite atunci când alerta ajunge în componenta Command&Control. Cu ajutorul unui *script* Python simplu, se pot calcula informațiile cele mai importante, cum ar fi: timpul mediu dintre publicare și consumare per agent, timpul mediu de procesare al alertelor per agent, durata medie de procesare până la publicare a unei alerte.

4.2.5.3. Instrumente folosite

Ca instrumente s-au utilizat biblioteci specifice Python în vederea calculării acestor cerințe.

4.3. Aspecte legate de fiabilitate/securitate

Una dintre masurile legate de fiabilitate poate fi considerată redundanța monitorizării grupului de instanțe. Mai exact, existența unui număr mare de agenți care colectează date de pe mai multe sisteme asigură faptul că, în cazul unui atac, se identifică acesta și pot fi luate măsuri pentru restul sistemelor computaționale. Un alt mecanism de fiabilitate îl constituie generarea automată de alerte în privința oricărui comportament suspect identificat.

Una dintre diversele modalități de îmbunătățire a fiabilității ar fi crearea unei componente separate pentru asigurarea actualizărilor. Prototipul nu utilizează mecanisme avansate de învățare automată, cum este utilizat în experimentul din Articolul [45] pentru a asigura adaptarea continuă la modificări ale atacurilor și apariția de atacuri noi. Din acest motiv, se poate crea o componentă separată cu rol de actualizare regulată a tehnologiilor folosite. În acest fel se asigură o măsură minimă de fiabilitate în cazul proiectului.

Prototipul presupune utilizarea unor tehnici și tehnologii de asigurare a continuității serviciilor EDR. Din acest motiv, accentul s-a aplicat pe modelarea unei arhitecturi care să utilizeze toate instrumentele detaliate anterior într-un mod avantajos. Proiectul necesită măsuri de securitate în privința asigurării criptării datelor în tranzit și în repaus. O alta metodă de securitate constituie crearea unui mecanism de autentificare și autorizare pentru accesul la programele soluției în vede-rea mentenanței lor.

4.4. Rezultate experimentale

4.4.1. Rezultate pentru simplificarea formatului alertelor

4.4.1.1. Prezentare rezultat

Rezultatele constau, după cum s-a detaliat și în secțiunea de testare, în simplificarea corectă a alertelor. Pentru o privire de ansamblu, s-a evidențiat o alertă neprocesată în Listing 4.1. Acesta

conține o serie de date care, într-un context mai avansat al implementării soluției, ar constitui note de diferențiere pentru identificarea atacurilor.

```
[  
  {  
    "data": {  
      "src_ip": "10.177.186.9",  
      "flow": {  
        "start": "2024-06-24T05:26:19.079929+0000",  
        "src_ip": "10.177.186.9",  
        "dest_ip": "10.177.186.8",  
        "pkts_toserver": "5",  
        "dest_port": "22",  
        "src_port": "35324",  
      },  
      "in_iface": "wg0",  
      "pkt_src": "wire/pcap",  
      "dest_ip": "10.177.186.8",  
      "dest_port": "22",  
      "timestamp": "2024-06-24T05:26:23.179618+0000",  
      "proto": "TCP",  
      "event_type": "alert",  
      "direction": "to_server",  
      "alert": {  
        "action": "allowed",  
        "severity": "2",  
        "rev": "20",  
        "category": "Attempted Information Leak",  
        "signature_id": "2001219",  
        "gid": "1",  
        "signature": "ET SCAN Potential SSH Scan",  
        "metadata": {  
          "created_at": ["2010_07_30"],  
          "updated_at": ["2019_07_26"]  
        }  
      },  
      "flow_id": "906244108495485.000000",  
      "src_port": "35324",  
    },  
    "agent": {  
      "id": "003",  
      "name": "Endpoint3",  
      "ip": "10.177.186.8"  
    }  
  }, {..}  
]  
]
```

Listing 4.1. *Log Collector* - alertă neprocesată

Într-un aşa caz, trebuie modificat doar şablonul stabilit pentru acceptarea noilor date. Însă, în contextul actual, se simplifică alerta și se ajunge la rezultatul din Listing 4.2.

```
{  
  "test_latency": {  
    "start_processing": "2024-06-24 08:26:47.027988",  
    "start_publishing": "2024-06-24 08:26:47.058488"  
  },  
  "data": {  
    "dest_ip": "10.177.186.8",  
    "timestamp": "2024-06-24T05:26:23.179618+0000",  
    "dest_port": "22",  
    "src_port": "35324",  
    "src_ip": "10.177.186.9",  
  }  
}
```

```

    "proto": "TCP",
    "event_type": "alert",
    "alert": {
        "signature_id": "2001219",
        "action": "allowed",
        "severity": "2",
        "signature": "ET SCAN Potential SSH Scan",
        "category": "Attempted Information Leak"
    }
},
"agent": {
    "id": "003",
    "name": "Endpoint3",
    "ip": "10.177.186.8"
}
}
}

```

Listing 4.2. Log Collector - alertă procesată

4.4.1.2. Modalități de îmbunătățire rezultat

Dacă scopul este de a obține o simplificare a alertei, rezultatul este unul potrivit. Cu toate acestea, se poate îmbunătăți implementarea care produce acest rezultat. Se pot utiliza biblioteci precum jsonpath-ng pentru a extrage datele folosind expresii JSONPath. O altă bibliotecă utilă este pandas care poate fi folosită pentru normalizarea și filtrarea datelor din JSON într-un DataFrame.

4.4.2. Rezultate pentru preventia încăderii agentului Wazuh

4.4.2.1. Prezentare rezultat

Rezultatul constă în activarea automată a agentului Wazuh. Statusul corect al unui agent ca acesta să ruleze și să asigure preluarea continuă a datelor și alertelor trebuie să fie ca în Figura 4.4.

```

[ubuntu@ip-172-31-93-122:~$ sudo systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
  Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
  Active: active (running) since Sun 2024-06-30 14:13:39 UTC; 1h 47min ago
    Process: 382 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
      Tasks: 31 (limit: 1120)
     Memory: 75.8M
        CPU: 19.508s
       CGroup: /system.slice/wazuh-agent.service
               ├─574 /var/ossec/bin/wazuh-execd
               ├─700 /var/ossec/bin/wazuh-agentd
               ├─733 /var/ossec/bin/wazuh-syscheckd
               ├─784 /var/ossec/bin/wazuh-logcollector
               └─859 /var/ossec/bin/wazuh-modulesd

Jun 30 14:13:30 ip-172-31-93-122 systemd[1]: Starting Wazuh agent...
Jun 30 14:13:31 ip-172-31-93-122 env[382]: Starting Wazuh v4.7.3...
Jun 30 14:13:32 ip-172-31-93-122 env[382]: Started wazuh-execd...
Jun 30 14:13:34 ip-172-31-93-122 env[382]: Started wazuh-agentd...
Jun 30 14:13:35 ip-172-31-93-122 env[382]: Started wazuh-syscheckd...
Jun 30 14:13:36 ip-172-31-93-122 env[382]: Started wazuh-logcollector...
Jun 30 14:13:37 ip-172-31-93-122 env[382]: Started wazuh-modulesd...
Jun 30 14:13:39 ip-172-31-93-122 env[382]: Completed.
Jun 30 14:13:39 ip-172-31-93-122 systemd[1]: Started Wazuh agent.
ubuntu@ip-172-31-93-122:~$ 

```

Figura 4.4. Status activ agent Wazuh

Componenta Command&Control identifică această stare și iterează prin lista de agenti în cadrul mecanismului de *heartbeat* detaliat în Capitolul 3 - Listing 3.10. În cazul încăderii agentului Wazuh fie în urma unui atac, fie în urma unei probleme tehnice, se identifică datele acestuia și se trimit o comandă de reactivare a lui, urmând ca mai apoi să verifice rezultatul execuției comenzi.

Scenariul reactivării este disponibil în Anexa 4, iar semnalarea neregulii în cazul unui agent este evidențiată în Listing 4.3.

```
[+++] [check_wazuh_agents()] INFO: Agent <('10.177.186.5', 53876)> seems to be inactive
[+++] INFO: Waiting for c2_agent response...
[+++] INFO: Waiting for c2_agent response...
[+++] [check_wazuh_agents()] INFO: Agent <('10.177.186.5', 53876)> successfully activated.
```

Listing 4.3. Detectare agent Wazuh - consolă

4.4.2.2. Modalități de îmbunătățire rezultat

Scenariul acoperă cazul când agentul este închis, nu și când acesta este alterat. O abordare mai complexă a verificării periodice a integrității agentul ar presupune o serie de verificări suplimentare, precum: analiza fișierelor sale de jurnalizare și a sistemului, analiza datei ultimei modificări în cadrul fișierelor sale, analiza conexiunilor suspecte pe care le-ar putea iniția, verificarea integrității fișierelor prin compararea valorilor *hash* calculate anterior cu cele noi. De asemenea, aceste verificări trebuie realizate și în cazul IDS Suricata.

4.4.3. Rezultate pentru reacția diferențiată la diferite tipuri de alarme

4.4.3.1. Prezentare rezultat

Rezultatul constă în oprirea unui atac, exemplificat fiind *SSH Brute Force*. Atacul este realizat folosind instrumentul Hydra, detaliu specificat și în secțiunile anterioare. Scenariul constă în rularea atacului și identificarea apariției alarmelor în fișierul de jurnalizare al IDS Suricata (Figura 4.5).

The screenshot shows two terminal windows. The top window displays the output of the 'ip a' command on an Ubuntu system, listing network interfaces (lo, eth0, wg0) with their respective configurations. The bottom window shows the 'tail -n 0 -f /var/log/suricata/fast.log' command running, which logs various network events. A green box highlights several entries related to an SSH scan from 10.177.186.9 to 10.177.186.5, indicating an attempted information leak and privilege gain. The bottom window also shows the Hydra tool being used to attack the host at 10.177.186.5 with usernames from 'usernames.txt' and passwords from 'passlist.txt'.

```
LICENTA — ubuntu@ip-172-31-93-122: ~ — ssh -i vockey.pem ubuntu@10.177.186.5 — 119x31
[ubuntu@ip-172-31-93-122:~]$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default qlen 1000
    link/ether 12:66:fd:80:52:25 brd ff:ff:ff:ff:ff:ff
    inet 172.31.93.122/20 metric 100 brd 172.31.95.255 scope global dynamic eth0
        valid_lft 3331sec preferred_lft 3331sec
    inet6 fe80::1066:fdff:fe80:5225/64 scope link
        valid_lft forever preferred_lft forever
3: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 8921 qdisc noqueue state UNKNOWN group default qlen 1000
    link/none
    inet 10.177.186.5/24 scope global wg0
        valid_lft forever preferred_lft forever
[ubuntu@ip-172-31-93-122:~]$ tail -n 0 -f /var/log/suricata/fast.log
06/30/2024-16:48:15.639602 [**] [1:2003068:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:58800 -> 10.177.186.5:22
06/30/2024-16:48:15.642248 [**] [1:2003068:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:58832 -> 10.177.186.5:22
06/30/2024-16:48:15.823614 [**] [1:2006546:9] ET SCAN LibSSH Based Frequent SSH Connections Likely BruteForce Attack [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.177.186.9:58822 -> 10.177.186.5:22
06/30/2024-16:49:01.604984 [**] [1:2001219:20] ET SCAN Potential SSH Scan [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:58432 -> 10.177.186.5:22
06/30/2024-16:49:01.604984 [**] [1:2003068:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:58432 -> 10.177.186.5:22
06/30/2024-16:49:01.787617 [**] [1:2006546:9] ET SCAN LibSSH Based Frequent SSH Connections Likely BruteForce Attack [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.177.186.9:58432 -> 10.177.186.5:22
[ubuntu@ip-172-31-86-11:~]$ ssh -i vockey.pem ubuntu@10.177.186.9 — 119x29
...ssh -i vockey.pem ubuntu@10.177.186.9 ...ubuntu@ip-172-31-86-11: ~ — bash ...untu@ip-172-31-86-11: ~ — bash +
[ubuntu@ip-172-31-86-11:~]$ hydra -L usernames.txt -P passlist.txt ssh://10.177.186.5 -t 8
Hydra v9.5 (c) 2023 by van Hauser/IHC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-06-30 16:48:05
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 8 tasks per 1 server, overall 8 tasks, 10677719992 login tries (l:81476/p:1310542), ~13347214999 tries per task
[DATA] attacking ssh://10.177.186.5:22/
```

Figura 4.5. Detectare *SSH Brute Force* - fast.log

Înainte ca serverul Command&Control să înceapă următoarea sesiune de consum alerte, se poate observa în Figura 4.6 faptul că serviciul SSH este activ, deci se poate efectua un atac asupra lui, iar în tabela de IP blocate nu există vreo înregistrare.

```
● ● ● LICENTA — ubuntu@ip-172-31-93-122: ~ — ssh -i vockey.pem ubuntu@10.177.186.5 — 119x65
[ubuntu@ip-172-31-93-122:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Drop-In: /usr/lib/systemd/system/ssh.service.d
            └─ec2-instance-connect.conf
    Active: active (running) since Sun 2024-06-30 16:53:13 UTC; 5s ago
      Docs: man:sshd(8)
             man:sshd_config(5)
   Process: 3829 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 3830 (sshd)
     Tasks: 1 (limit: 1120)
    Memory: 1.7M
       CPU: 22ms
      CGroup: /system.slice/ssh.service
              └─3830 "sshd: /usr/sbin/sshd -D -o AuthorizedKeysCommand /usr/share/ec2-instance-connect/eic_run_authoriz[

Jun 30 16:53:13 ip-172-31-93-122 systemd[1]: Starting OpenBSD Secure Shell server...
Jun 30 16:53:13 ip-172-31-93-122 sshd[3830]: Server listening on 0.0.0.0 port 22.
Jun 30 16:53:13 ip-172-31-93-122 sshd[3830]: Server listening on :: port 22.
Jun 30 16:53:13 ip-172-31-93-122 systemd[1]: Started OpenBSD Secure Shell server.
lines 1-19 (END)
[ubuntu@ip-172-31-93-122:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target prot opt in     out     source          destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target prot opt in     out     source          destination
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target prot opt in     out     source          destination
ubuntu@ip-172-31-93-122:~$ ]
```

Figura 4.6. Status înainte de SSH Brute Force

Se execută comenziile trimise către agentul C2 și se observă imediat pe sistemul monitorizat noul status (Figura 4.7).

```
● ● ● LICENTA — ubuntu@ip-172-31-93-122: ~ — ssh -i vockey.pem ubuntu@10.177.186.5 — 119x64
[ubuntu@ip-172-31-93-122:~$ tail -n 0 -f /var/log/suricata/fast.log
06/30/2024-16:57:30.204465 [**] [1:2001219:20] ET SCAN Potential SSH Scan [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:52372 -> 10.177.186.5:22
06/30/2024-16:57:30.204465 [**] [1:2003068:7] ET SCAN Potential SSH Scan OUTBOUND [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.177.186.9:52372 -> 10.177.186.5:22
06/30/2024-16:57:30.361240 [**] [1:2006546:9] ET SCAN LibSSH Based Frequent SSH Connections Likely BruteForce Attack
**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.177.186.9:52366 -> 10.177.186.5:22
^C
ubuntu@ip-172-31-93-122:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Drop-In: /usr/lib/systemd/system/ssh.service.d
            └─ec2-instance-connect.conf
    Active: inactive (dead) since Sun 2024-06-30 16:57:46 UTC; 20s ago
      Docs: man:sshd(8)
             man:sshd_config(5)
   Process: 3829 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Process: 3830 ExecStart=/usr/sbin/sshd -D -o AuthorizedKeysCommand /usr/share/ec2-instance-connect/eic_run_authori
   Main PID: 3830 (code=exited, status=0/SUCCESS)
     CPU: 270ms

Jun 30 16:57:46 ip-172-31-93-122 sshd[3858]:
Jun 30 16:57:46 ip-172-31-93-122 sshd[3858]:
Jun 30 16:57:46 ip-172-31-93-122 sshd[3857]: error: maximum authentication attempts exceeded for invalid user 3d from
Jun 30 16:57:46 ip-172-31-93-122 sshd[3857]: Disconnecting invalid user 3d 10.177.186.9 port 52376: Too many authentic
Jun 30 16:57:46 ip-172-31-93-122 sshd[3857]:
Jun 30 16:57:46 ip-172-31-93-122 sshd[3857]: error: maximum authentication attempts exceeded for invalid user 3d from
Jun 30 16:57:46 ip-172-31-93-122 sshd[3853]: Disconnecting invalid user 3d 10.177.186.9 port 52346: Too many authentic
Jun 30 16:57:46 ip-172-31-93-122 sshd[3853]:
Jun 30 16:57:46 ip-172-31-93-122 sshd[3853]:
lines 1-22/22 (END)
ubuntu@ip-172-31-93-122:~$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target prot opt in     out     source          destination
    0     0 DROP      all -- *      *      10.177.186.9      0.0.0.0/0
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target prot opt in     out     source          destination
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target prot opt in     out     source          destination
    64   8192 DROP      all -- *      *      0.0.0.0/0      10.177.186.9
    0     0 DROP      all -- *      *      0.0.0.0/0      10.177.186.9
ubuntu@ip-172-31-93-122:~$ ]
```

Figura 4.7. Status după SSH Brute Force

4.4.3.2. Modalități de îmbunătățire rezultat

Abordarea curentă pentru această funcționalitate este una de bază, nefolosind mecanisme care pot îmbunătăți această clasificare a alertelor în vederea stabilirii mai rapid a unei reacții potrivite. O abordare mai complexă implică realizarea unei componente separate care folosește tehnici de învățare automată pentru a clasifica în mod corespunzător alertele și a optimiza timpul de stabilire a reacției.

4.4.4. Rezultate pentru observarea timpilor de procesare a alertelor

4.4.4.1. Prezentare rezultat

Durata medie de procesare a alertelor a fost înregistrată ca fiind aproximativ 0.06 secunde, ceea ce sugerează un timp acceptabil în contextul unui prototip. Agenții au timpi de procesare ușor diferiți, dar toți se încadrează între 0.05 ÷ 0.08 secunde, ceea ce indică o consistență în alternarea task-urilor fiecărui agent. Cu ajutorul acestor metrii se poate observa dacă un agent prezintă un timp mai mare, ceea ce înseamnă că în procesul de monitorizare au existat un număr de警 mai mare decât de obicei.

4.4.4.2. Modalități de îmbunătățire rezultat

Rezultatul referitor la timpii obținuți poate fi îmbunătățit prin înglobarea tuturor adăugirilor necesare discutate în secțiunile anterioare și în Capitolul 3.

4.5. Utilizarea sistemului

Gradul mare de generalitate a soluției permite folosirea acesteia într-o varietate de contexte economice, asigurând un mecanism automatizat de protejare a resurselor software deținute. Lucrarea propune, astfel, asigurarea securității în orice arie profesională, soluția putând fi implementată atât pentru sistemele personale conectate la rețea, cat și pentru protejarea digitală a companiilor sau instituțiilor. Arhitectura modulară oferă tuturor utilizatorilor posibilitatea să-și extindă numărul de sisteme protejate, fără a fi nevoie de o monitorizare umană a securității fiecărui sistem în parte.

Concluzii

Gradul în care s-a realizat tema propusă

Obiectivele principale ale lucrării au fost de a propune o soluție care să crească nivelul de securitate al sistemelor monitorizate, să elimine situațiile de reacție manuală în cazul atacurilor și să poată fi folosită în cadrul mai multor platforme software. Soluția a fost dezvoltată până la stadiul unui prototip funcțional, care îndeplinește obiectivele propuse, dar al cărui nivel de implementare este încă unul de bază. Aceasta tratează cazuri specifice, fiind prioritară realizarea legăturilor dintre componente și explorarea funcționalităților tehniciilor și tehnologiilor folosite în detrimentul unui număr mai mare de cazuri în care aplicația să fie testată.

Soluția dezvoltată în cadrul lucrării de diplomă oferă o serie de funcționalități care acoperă îndeplinirea obiectivelor detaliate în Capitolul 1. Configurările de pe sistemele monitorizate sunt realizate corect, astfel încât IDS generează alerte în mod corespunzător, iar agentul Wazuh le preia și le transferă managerului Wazuh pentru a putea fi accesibile prin API ELK. Componenta *Log Collector*, descrisă în Capitolul 2, simplifică alertele și le publică în coada de mesaje, urmărind îndeplinirea obiectivului de independentă. Componenta *Command&Control* preia alertele din coadă și verifică semnătura lor pentru a alege o comandă potrivită spre a o trimite proprietului agent. Agentul execută comanda și returnează răspunsul. Această parte urmărește eliminarea acțiunii umane.

Prin integrarea colectării și reacției, rezultă un sistem de tip EDR care îndeplinește obiectivul general menționat, cel de a crește nivelul de securitate pe *endpoints*. Așadar, soluția s-a realizat într-un grad mulțumitor din perspectiva utilizării tuturor tehnologiilor dorite și într-un grad acceptabil din maniera complexității în implementare.

Evidențierea concisă a contribuțiilor/soluțiilor personale

Soluțiile personale în dezvoltarea proiectului au constat în modalitatea de proiectare a arhitecturii și în valorificarea funcționalităților unor tehnologii în alte contexte față de utilizarea lor uzuală.

Una dintre ideile personalizate este cea de utilizare a unui sistem SIEM drept container de transfer pentru alertele generate de IDS în vederea eliminării dependenței de specificitatea unui instrument SIEM. Acest lucru înseamnă generalizare și posibilitate de personalizare ulterioară a proiectului, dar și lipsa acoperirii mai multor tipuri de atacuri care nu fac parte din tipul *System Intrusion*.

O altă idee personală este modalitatea de utilizare a unui server de comandă și control: folosirea sa în scop defensiv, având rolul de a reacționa la atacuri spre neutralizarea acestora. Mecanismul de *backdoor* implementat are avantajul separării de partea de colectare și de a nu fi în atenția posibililor atacatori, dar dezavantajul de a fi confundat cu un atac de tip C&C.

Canalele de comunicare distințe pentru partea de colectare și partea de reacție reprezintă o altă idee personală. Acestea au scopul de a elibera orice legătură aparentă între componente, spre a evita posibila neutralizare a agentului C2. În cazul închiderii agentului Wazuh, serverul C2 sesizează și-l activează.

Comparatie cu alte proiecte similare

Există pe piață o gamă variată de produse care asigură securitatea sistemului fără a necesita intervenția utilizatorului. Unele dintre acestea au fost evidențiate în Capitolul 1, folosind abordări similare cu soluția propusă. De exemplu, Cyphon poate fi văzut ca un ecosistem vast și adaptabil, în timp ce soluția poate fi mai focalizată și specializată doar pe anumite tipuri de atacuri.

Metasploit Framework reprezintă un instrument care utilizează C&C care dispune de mecanisme complexe de evitare a detectării de către sistemele de securitate, prin comparație cu propria componentă care implementează un server de comandă și control cu funcționalități specifice de bază. Compania Stamus Network este unul dintre cei mai mari utilizatori ai IDS Suricata, publicând cărți pe tema valorificării avansate a funcționalităților Suricata. Prin comparație, soluția utilizează IDS Suricata pentru a genera atacuri conform regulilor predefinite și a testa funcționarea IDS prin reguli personalizate.

Posibile direcții de dezvoltare

Modul în care a fost proiectată soluția permite ca aceasta să fie extinsă. Se poate renunța la procesarea exclusivă a alertelor generate de IDS, se poate înlătări instrumentul SIEM pentru a avea acces și la mai multe funcționalități față de căte oferă Wazuh: IBM QRadar are capacitatea de procesare a unor volume mari de date, utilizează tehnologii *User and Entity Behavior Analytics* care permit analiza mai detaliată a comportamentului utilizatorului și a detecta posibile conduite anormale. De asemenea, serverul de comandă și control poate fi dezvoltat astfel încât să includă mecanisme de evitare a detectării de către sistemele de securitate.

Așa cum a fost prezentat și în Capitolul 2, scopul unui server C2 poate fi și reactiv, mecanisme de persistență, comunicare și execuție de comenzi fiind avantaje pentru controlul acțiunilor asupra sistemelor care necesită intervenție. A fost analizat experimentul descris în [44], unde s-a abordat importanța evitării detectiei bazate pe rețea, aplicând teste atât pentru versiunile criptate, cât și necriptate ale C2-ului pentru a determina eficacitatea acestora în evitarea detectiei. Astfel, o posibilă direcție de dezvoltare ar putea fi utilizarea infrastructurilor C2 într-un context defensiv prin implementarea unui mecanism de evitare a detectiei de către sistemele de securitate.

O altă direcție o poate constitui scalarea proiectului la a avea capacitatea de a monitoriza și gestiona mai multe *endpoints*. De asemenea, integrarea tehnologiilor de Inteligență Artificială constituie o direcție necesară de dezvoltare în vederea creșterii acurateții alarmelor și a antrenării serverului de comandă și control pentru a oferi reacții adecvate la atacuri noi și la variațiuni ale atacurilor existente.

Lecții învățătoare pe parcursul dezvoltării lucrării de diplomă

Principala lecție învățată în procesul alegerii temei, al proiectării soluției pentru tema aleasă și al implementării acesteia este de natură organizatorică, întrucât procesul de creare a unor etape cu rol bine definit și clar este foarte important. Aceasta ajută în procesul de cercetare pe tema aleasă, în organizarea ideilor, obiectivelor dorite.

O altă lecție învățată în urma acestui proces este modul în care să se realizeze extragerea de informații din diverse surse, cum să se utilizeze funcționalitățile instrumentelor în scopurile dorite, atâtă timp cât este fezabil.

În cele din urmă, ultima lecție semnificativă învățată este modul de proiectare al unei soluții astfel încât să disponă de cât mai multă generalitate, să permită scalarea sa ulterioară.

Bibliografie

- [1] D. Papatsaroucha, Y. Nikoloudakis, I. Kefaloukos, E. Pallis, and E. K. Markakis, “A Survey on Human and Personality Vulnerability Assessment in Cyber-security: Challenges, Approaches, and Open Issues,” *CoRR*, vol. abs/2106.09986, 2021. [Online]. Available: <https://arxiv.org/pdf/2106.09986.pdf>
- [2] The Verizon DBIR Team and C. David Hylander and Philippe Langlois and Alex Pinto and Suzanne Widup, “2024 Verizon Data Breach Investigations Report,” VERIZON, Report DBIR, 2024, Ultima accesare: 8.05.2024. [Online]. Available: <https://www.verizon.com/business/resources/Tb74/reports/2024-dbir-data-breach-investigations-report.pdf>
- [3] V. Vassilakis, “Static and Dynamic Analysis of WannaCry Ransomware,” in *IEICE Information and Communication Technology Forum (ICTF)*, 2018. [Online]. Available: https://www.academia.edu/38692588/Static_and_Dynamic_Analysis_of_WannaCry_Ransomware
- [4] CrowdStrike, “CrowdStrike 2024 Global Threat Report,” 2024, accessed: 2024-06-23. [Online]. Available: <https://www.crowdstrike.com/global-threat-report/>
- [5] A. Verma, R. Surendra, B. Reddy, P. Chawla, and K. Soni, “Cyber Security in Digital Sector,” in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 2021, pp. 703–710,
DOI: [10.1109/ICAIS50930.2021.9395933](https://doi.org/10.1109/ICAIS50930.2021.9395933).
- [6] T. Abrahams, S. Ewuga, S. Dawodu, A. Adegbite, and A. Hassan, “A REVIEW OF CYBER-SECURITY STRATEGIES IN MODERN ORGANIZATIONS: EXAMINING THE EVOLUTION AND EFFECTIVENESS OF CYBERSECURITY MEASURES FOR DATA PROTECTION,” *Computer Science & IT Research Journal*, vol. 5, pp. 1–25, 01 2024,
DOI: [10.51594/csitrj.v5i1.699](https://doi.org/10.51594/csitrj.v5i1.699).
- [7] European Parliament, “Fighting Cybercrime: New EU Cybersecurity Laws Explained,” November 3 2022, accessed: 2024-06-23. [Online]. Available: <https://www.europarl.europa.eu/topics/en/article/20221103STO48002/fighting-cybercrime-new-eu-cybersecurity-laws-explained>
- [8] B. Schneier, “Testimony at the U.S. House of Representatives Joint Hearing “Understanding the Role of Connected Devices in Recent Cyber Attacks”,” https://www.schneier.com/essays/archives/2016/11/testimony_at_the_us_.html, 2016, Ultima accesare: 8.02.2024.
- [9] D. Upreti and U. Bhangale, “CSE 534 Project Report Understanding the Mirai Botnet,” <https://www3.cs.stonybrook.edu/~arunab/course/2018b-1.pdf>, 2018, Ultima accesare: 8.02.2024.
- [10] T. Wisanwanichthan and M. Thammawichai, “A Double-Layered Hybrid Approach for Network Intrusion Detection System Using Combined Naive Bayes and SVM,” *IEEE Access*, vol. 9, pp. 138 432–138 450, 2021,
DOI: [10.1109/ACCESS.2021.3118573](https://doi.org/10.1109/ACCESS.2021.3118573).
- [11] N. Patil and R. Kondabala, “Two-Layer Secure Mechanism for Electronic Transactions,” in *2022 International Conference on Recent Trends in Microelectronics, Automation, Computing and Communications Systems (ICMACC)*, 2022, pp. 174–181,
DOI: [10.1109/ICMACC54824.2022.10093478](https://doi.org/10.1109/ICMACC54824.2022.10093478).

- [12] S. Shlaer and S. Mellor, “Recursive design of an application-independent architecture,” *IEEE Software*, vol. 14, no. 1, pp. 61–72, 1997,
DOI: [10.1109/52.566429](https://doi.org/10.1109/52.566429).
- [13] T. M. Ghazal, M. K. Hasan, R. A. Zitar, N. A. Al-Dmour, W. T. Al-Sit, and S. Islam, “Cybers Security Analysis and Measurement Tools Using Machine Learning Approach,” in *2022 1st International Conference on AI in Cybersecurity (ICAIC)*, 2022, pp. 1–4,
DOI: [10.1109/ICAIC53980.2022.9897045](https://doi.org/10.1109/ICAIC53980.2022.9897045).
- [14] “Cyphon Documentation,” <https://cyphon.readthedocs.io/en/latest/index.html>, 2018, Ultima accesare: 8.02.2024.
- [15] F. R. Alzaabi and A. Mehmood, “A Review of Recent Advances, Challenges, and Opportunities in Malicious Insider Threat Detection Using Machine Learning Methods,” *IEEE Access*, vol. 12, pp. 30 907–30 927, 2024,
DOI: [10.1109/ACCESS.2024.3369906](https://doi.org/10.1109/ACCESS.2024.3369906).
- [16] G. Karantzas and C. Patsakis, “An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors,” *Journal of Cybersecurity and Privacy*, vol. 1, no. 3, pp. 387–421, 2021,
DOI: [10.3390/jcp1030021](https://doi.org/10.3390/jcp1030021).
- [17] C. Novo, G. Xavier, and R. Morla, “Tweaking Metasploit to Evade Encrypted C2 Traffic Detection,” *arXiv preprint arXiv:2209.00943*, 2022. [Online]. Available: <https://arxiv.labs.arxiv.org/html/2209.00943>
- [18] S. S. Tirumala, H. Sathu, and A. Sarrafzadeh, “Free and open source intrusion detection systems: A study,” in *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 1, 2015, pp. 205–210,
DOI: [10.1109/ICMLC.2015.7340923](https://doi.org/10.1109/ICMLC.2015.7340923).
- [19] Z. S. Younus and M. Alanezi, “Detect and Mitigate Cyberattacks Using SIEM,” in *2023 16th International Conference on Developments in eSystems Engineering (DeSE)*, 2023, pp. 510–515,
DOI: [10.1109/DeSE60595.2023.10469387](https://doi.org/10.1109/DeSE60595.2023.10469387).
- [20] H. Jadi doleslamy, “Weaknesses, Vulnerabilities And Elusion Strategies Against Intrusion Detection Systems,” in *International Journal of Computer Science & Engineering Survey*, vol. 3, 2012, pp. 15–25,
DOI: [10.5121/ijcses.2012.3402](https://doi.org/10.5121/ijcses.2012.3402).
- [21] S. Abdulrezzak and F. A. Sabir, “Enhancing Intrusion Prevention in Snort System,” in *2023 15th International Conference on Developments in eSystems Engineering (DeSE)*, 2023, pp. 88–93,
DOI: [10.1109/DeSE58274.2023.10099757](https://doi.org/10.1109/DeSE58274.2023.10099757).
- [22] A. Tiwari, S. Saraswat, U. Dixit, and S. Pandey, “Refinements In Zeek Intrusion Detection System,” in *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, 2022, pp. 974–979,
DOI: [10.1109/ICACCS54159.2022.9785047](https://doi.org/10.1109/ICACCS54159.2022.9785047).
- [23] K. Wong, C. Dillabaugh, N. Seddigh, and B. Nandy, “Enhancing Suricata intrusion detection system for cyber security in SCADA networks,” in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017, pp. 1–5,
DOI: [10.1109/CCECE.2017.7946818](https://doi.org/10.1109/CCECE.2017.7946818).

- [24] J. Gardiner, M. Cova, and S. Nagaraja, “Command & Control: Understanding, Denying and Detecting,” University of Birmingham, Tech. Rep., 2014. [Online]. Available: <https://arxiv.org/pdf/1408.1136>
- [25] F. M. Ramos and X. Wang, “A Machine Learning Based Approach to Detect Stealthy Cobalt Strike C &C Activities from Encrypted Network Traffic,” in *Machine Learning for Networking*, É. Renault and P. Mühlethaler, Eds. Cham: Springer Nature Switzerland, 2023, pp. 113–129,
DOI: [10.1007/978-3-031-36183-8_8](https://doi.org/10.1007/978-3-031-36183-8_8).
- [26] C. N. Nguyen, J.-S. Kim, and S. Hwang, “KOHA: Building a Kafka-Based Distributed Queue System on the Fly in a Hadoop Cluster,” in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2016, pp. 48–53,
DOI: [10.1109/FAS-W.2016.23](https://doi.org/10.1109/FAS-W.2016.23).
- [27] A. E. Alami, M. Bahaj, and Y. Khourdifi, “Supply of a key value database redis in-memory by data from a relational database,” in *2018 19th IEEE Mediterranean Electrotechnical Conference (MELECON)*, 2018, pp. 46–51,
DOI: [10.1109/MELCON.2018.8379066](https://doi.org/10.1109/MELCON.2018.8379066).
- [28] A. Pathak and C. Kalaiarasaran, “RabbitMQ Queuing Mechanism of Publish Subscribe model for better Throughput and Response,” in *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2021, pp. 1–7,
DOI: [10.1109/ICECCT52121.2021.9616722](https://doi.org/10.1109/ICECCT52121.2021.9616722).
- [29] A. A. Md Shoeb, R. Hasan, M. Haque, and M. Hu, “A Comparative Study on I/O Performance between Compute and Storage Optimized Instances of Amazon EC2,” in *2014 IEEE 7th International Conference on Cloud Computing*, 2014, pp. 970–971,
DOI: [10.1109/CLOUD.2014.146](https://doi.org/10.1109/CLOUD.2014.146).
- [30] V. Sulov, “On the Essence of Hardware Performance,” *Research Journal of Economics, Business and ICT*, vol. 9, pp. 13–18, 05 2014. [Online]. Available: https://www.academia.edu/63124128/On_the_Essence_of_Hardware_Performance
- [31] V. Mateljan, V. Juricic, and M. Moguljak, “Virtual machines in education,” in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2014, pp. 603–607,
DOI: [10.1109/MIPRO.2014.6859639](https://doi.org/10.1109/MIPRO.2014.6859639).
- [32] J. Singh and N. K. Walia, “A Comprehensive Review of Cloud Computing Virtual Machine Consolidation,” *IEEE Access*, vol. 11, pp. 106190–106209, 2023,
DOI: [10.1109/ACCESS.2023.3314613](https://doi.org/10.1109/ACCESS.2023.3314613).
- [33] N. Naik, “Docker container-based big data processing system in multiple clouds for everyone,” in *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, pp. 1–7,
DOI: [10.1109/ICKII46306.2019.9042621](https://doi.org/10.1109/ICKII46306.2019.9042621).
- [34] J. Smith and R. Nair, “The architecture of virtual machines,” *Computer*, vol. 38, no. 5, pp. 32–38, 2005,
DOI: [10.1109/MC.2005.173](https://doi.org/10.1109/MC.2005.173).
- [35] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, “A Comparative Study of Containers and Virtual Machines in Big Data Environment,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 178–185,
DOI: [10.1109/CLOUD.2018.00030](https://doi.org/10.1109/CLOUD.2018.00030).

- [36] G. Blinowski, A. Ojdowska, and A. Przybyłek, “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation,” *IEEE Access*, vol. 10, pp. 20 357–20 374, 2022, DOI: [10.1109/ACCESS.2022.3152803](https://doi.org/10.1109/ACCESS.2022.3152803).
- [37] J. Ding, I. Balasingham, and P. Bouvry, “Management of Overlay Networks: A Survey,” in *2009 Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 2009, pp. 249–255, DOI: [10.1109/UBICOMM.2009.49](https://doi.org/10.1109/UBICOMM.2009.49).
- [38] A. Moskvichev and M. Dolgachev, “System of Collection and Analysis Event Log from Sources under Control of Windows Operating System,” in *2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*, 2020, pp. 1–5, DOI: [10.1109/FarEastCon50210.2020.9271520](https://doi.org/10.1109/FarEastCon50210.2020.9271520).
- [39] F. Ahmed, U. Jahangir, H. Rahim, K. Ali, and D.-e.-S. Agha, “Centralized Log Management Using Elasticsearch, Logstash and Kibana,” in *2020 International Conference on Information Science and Communication Technology (ICISCT)*, 2020, pp. 1–7, DOI: [10.1109/ICISCT49550.2020.9080053](https://doi.org/10.1109/ICISCT49550.2020.9080053).
- [40] Wazuh, Inc., “Wazuh Documentation,” 2024, accessed: 2024-06-24. [Online]. Available: <https://documentation.wazuh.com/current/integrations-guide/elastic-stack>
- [41] The MITRE Corporation, “Common Event Expression - Architecture Overview,” p. 12, 2010. [Online]. Available: https://cee.mitre.org/docs/CEE_Architecture_Overview-v0.5.pdf
- [42] M. Carvalho, J. M. Bradshaw, L. Bunch, T. Eskridge, P. J. Feltovich, R. R. Hoffman, and D. Kidwell, “Command and Control Requirements for Moving-Target Defense,” *IEEE Intelligent Systems*, vol. 27, no. 3, pp. 79–85, 2012, DOI: [10.1109/MIS.2012.45](https://doi.org/10.1109/MIS.2012.45).
- [43] M. Carvalho, T. C. Eskridge, L. Bunch, A. Dalton, R. Hoffman, J. M. Bradshaw, P. J. Feltovich, D. Kidwell, and T. Shanklin, “MTC2: A command and control framework for moving target defense and cyber resilience,” in *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, 2013, pp. 175–180, DOI: [10.1109/ISRCS.2013.6623772](https://doi.org/10.1109/ISRCS.2013.6623772).
- [44] J. Matza, “Development of an Experimental C2 Utilizing Network Sockets,” Bachelor of Arts Honors Thesis, University at Albany, State University of New York, 2021. [Online]. Available: <https://scholarsarchive.library.albany.edu/honorscollege.business/63/>
- [45] H. Zhang, K.-Y. Lin, W. Chen, and L. Genyuan, “Using Machine Learning techniques to improve Intrusion Detection Accuracy,” in *2019 IEEE 2nd International Conference on Knowledge Innovation and Invention (ICKII)*, 2019, pp. 308–310, DOI: [10.1109/ICKII46306.2019.9042621](https://doi.org/10.1109/ICKII46306.2019.9042621).
- [46] Wazuh, Inc., “Wazuh Documentation,” 2024, accessed: 2024-06-24. [Online]. Available: <https://documentation.wazuh.com/current/deployment-options/virtual-machine/virtual-machine.html>

Anexa 1. Componența C2 - Fișier consumer_mq.py

```

1 def handle_agent_alerts(list_alerts_for_agent):
2     try:
3         current_targets = list_targets()
4         agent_ip = list_alerts_for_agent[0]["agent"]["ip"]    # A list must contain alerts
        ↳ just for 1 agent
5         session = next((session for session, ip in current_targets.items() if ip[0] ==
        ↳ agent_ip), None)
6         target_socket = TARGETS[session]
7         target_ip = IPS[session]
8         for alert in list_alerts_for_agent:
9             connection = initialize_connection()
10            insert_agent(connection, alert['agent']['ip'])
11            insert_timestamp(connection,
12                alert['agent']['ip'],
13                alert['test_latency']['start_processing'],
14                alert['test_latency']['start_publishing'],
15                alert['test_latency']['start_consumption']
16            )
17            if alert['data']['alert']['signature_id'] == '2001219':
18                attacker_ip = alert['data']['src_ip']
19                # action - block ip and stop ssh service - shortly attacker should get:
        ↳ Timeout connecting to ip_victim
20                bruteforce_ssh_reaction = f"""
21                    sudo iptables -A INPUT -s {attacker_ip} -j DROP && \
22                    sudo iptables -A OUTPUT -d {attacker_ip} -j DROP && \
23                    sudo systemctl stop ssh && \
24                    sudo iptables -L -v -n && \
25                    sudo systemctl status ssh &
26                """
27                response = shell(target_socket, target_ip, bruteforce_ssh_reaction)
28                if CHECK_BRUTEFORCE_SSH_SERVICE in response:
29                    logging.info(f"[+] INFO: Successfully stopped Brute Force Attack for
        ↳ {agent_ip}")
30                else:
31                    logging.info(f"[+] INFO: Unsuccessfully stopped Brute Force Attack for
        ↳ {agent_ip}")
32
33            elif alert['data']['alert']['signature_id'] == '2003068':    # ET SCAN
        ↳ Potential SSH Scan OUTBOUND
34                attacker_ip = alert['data']['src_ip']
35                bruteforce_ssh_reaction = f" sudo iptables -A OUTPUT -d {attacker_ip} -j
        ↳ DROP &
36                _ = shell(target_socket, target_ip, bruteforce_ssh_reaction)
37
38            else:
39                # Generic response for others alerts
40                message = f"[{datetime.datetime.now()}] <{agent_ip}> generate :
        ↳ {alert['data']}\\n"
41                with open('alerts_received.txt', 'a') as file:
42                    file.write(message + '\\n')
43            except IntegrityError as e:
44                logging.error(f"IntegrityError occurred: {str(e)}")
45
46            except Exception as e:
47                logging.error(
48                    f"[!] [handle_agent_alerts] ERROR when process alerts ( insert timestamp db,
        ↳ treat alerts): {e}")

```

Listing 4. Codul funcției de reacție

Anexa 2. Componența C2 - Fișier c2.py

```
1 def shell(target, ip, command):
2     def reliable_send(data):
3         try:
4             json_data = json.dumps(data)
5             json_data = json_data.encode('utf-8')
6             target.send(json_data)
7         except Exception as e:
8             logging.error(f"[!] [shell()] ERROR while send message to victim: {e}")
9
10    def reliable_recv():
11        try:
12            data = ""
13            while True:
14                try:
15                    chunk_recv = target.recv(1024).decode()
16                    if not chunk_recv or len(chunk_recv) == 0 or chunk_recv == '':
17                        break
18                    data = data + chunk_recv
19                return json.loads(data)
20            except ValueError as e:
21                continue
22
23        except Exception as e:
24            logging.error(f"[!] [shell()] ERROR while receiving data from target: {e}")
25        return None
26
27    result = None
28    reliable_send(command)
29
30    if command == 'q':
31        # in case of manual stop
32        pass
33
34    elif command == "exit":
35        target.close()
36        TARGETS.remove(target)
37        IPS.remove(ip)
38
39    elif command.strip().startswith("cd") and len(command.strip()) > 2:
40        pass
41
42    elif command[:8] == "download":
43        with open(command[9:], "wb") as file:
44            file_data = reliable_recv()
45            file.write(base64.b64decode(file_data))
46
47    elif command[:6] == "upload":
48        try:
49            with open(command[7:], "rb") as fin:
50                reliable_send(base64.b64encode(fin.read()))
51        except ValueError:
52            failed = "Failed to upload"
53            reliable_send(base64.b64encode(failed))
54    else:
55        result = reliable_recv()
56
57    return result
```

Listing 5. Codul funcției shell() - server C2

Anexa 3. Componența C2 - Fișier agent_c2.py

```

1  def shell():
2      global admin
3      while True:
4          try:
5              command = reliable_recv()
6              print(command)
7              if command == 'q':
8                  continue
9              elif command.strip().startswith("wazuh") and len(command.strip()) >
10                 len("wazuh"):
11                  check_wazuh(command)
12              elif command == "exit":
13                  break
14              elif command == "help":
15                  help_menu()
16              elif command.strip().startswith("cd") and len(command.strip()) > 2:
17                  change_directory(command)
18              elif command.strip().startswith("download") and len(command.strip()) >
19                 len("download "):
20                  download_file(command)
21              elif command.strip().startswith("upload") and len(command.strip()) >
22                 len("upload "):
23                  upload_file(command)
24              elif command.strip().startswith("get") and len(command.strip()) > len("get "):
25                  get_from_url(command)
26              elif command.strip().startswith("screenshot"):
27                  handle_screenshot()
28              elif command.strip().startswith("check"):
29                  check_if_admin()
30
31          elif command.strip().startswith("start"):
32              start_application_on_victim(command)
33
34      else:
35          exec_cmd(command)
36  except (ConnectionResetError, ConnectionAbortedError):
37      connection()

```

Listing 6. Codul funcției shell() - agent C2

Anexa 4. Componenta C2 - Scenariu activare agent Wazuh

```
LICENTA — ubuntu@ip-172-31-93-122: ~ — ssh -i vockey.pem ubuntu@10.177.186.5 — 119x65
[ubuntu@ip-172-31-93-122:~]$ sudo systemctl stop wazuh-agent
[ubuntu@ip-172-31-93-122:~]$ sudo systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
   Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Sun 2024-06-30 16:11:47 UTC; 2s ago
     Process: 2098 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
    Process: 2655 ExecStop=/usr/bin/env /var/ossec/bin/wazuh-control stop (code=exited, status=0/SUCCESS)
      CPU: 12.201s

Jun 30 16:11:46 ip-172-31-93-122 systemd[1]: Stopping Wazuh agent...
Jun 30 16:11:46 ip-172-31-93-122 env[2655]: Killing wazuh-modulesd...
Jun 30 16:11:47 ip-172-31-93-122 env[2655]: Killing wazuh-logcollector...
Jun 30 16:11:47 ip-172-31-93-122 env[2655]: Killing wazuh-syscheckd...
Jun 30 16:11:47 ip-172-31-93-122 env[2655]: Killing wazuh-agentd...
Jun 30 16:11:47 ip-172-31-93-122 env[2655]: Killing wazuh-execd...
Jun 30 16:11:47 ip-172-31-93-122 env[2655]: Wazuh v4.7.3 Stopped
Jun 30 16:11:47 ip-172-31-93-122 systemd[1]: wazuh-agent.service: Deactivated successfully.
Jun 30 16:11:47 ip-172-31-93-122 systemd[1]: Stopped Wazuh agent.
Jun 30 16:11:47 ip-172-31-93-122 systemd[1]: wazuh-agent.service: Consumed 12.201s CPU time.
[ubuntu@ip-172-31-93-122:~]$ sudo systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
   Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
   Active: activating (start) since Sun 2024-06-30 16:12:05 UTC; 1s ago
     PID: 2730 (wazuh-control)
      Tasks: 8 (limit: 1120)
     Memory: 6.0M
        CPU: 73ms
      CGroup: /system.slice/wazuh-agent.service
              └─2730 /bin/sh /var/ossec/bin/wazuh-control start
                  ├─2754 /var/ossec/bin/wazuh-execd
                  ├─2764 sleep 1
                  └─2765 /var/ossec/bin/wazuh-agentd

Jun 30 16:12:05 ip-172-31-93-122 systemd[1]: Starting Wazuh agent...
Jun 30 16:12:05 ip-172-31-93-122 env[2730]: Starting Wazuh v4.7.3...
Jun 30 16:12:06 ip-172-31-93-122 env[2730]: Started wazuh-execd...
[ubuntu@ip-172-31-93-122:~]$ sudo systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
   Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2024-06-30 16:12:12 UTC; 1s ago
     Process: 2730 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
      Tasks: 24 (limit: 1120)
     Memory: 171.1M
        CPU: 5.827s
      CGroup: /system.slice/wazuh-agent.service
              ├─2754 /var/ossec/bin/wazuh-execd
              ├─2765 /var/ossec/bin/wazuh-agentd
              ├─2782 /var/ossec/bin/wazuh-syscheckd
              ├─2796 /var/ossec/bin/wazuh-logcollector
              ├─2806 /var/ossec/bin/wazuh-modulesd
              └─3171 apt list --installed libpam-pwquality

Jun 30 16:12:05 ip-172-31-93-122 systemd[1]: Starting Wazuh agent...
Jun 30 16:12:05 ip-172-31-93-122 env[2730]: Starting Wazuh v4.7.3...
Jun 30 16:12:06 ip-172-31-93-122 env[2730]: Started wazuh-execd...
Jun 30 16:12:07 ip-172-31-93-122 env[2730]: Started wazuh-agentd...
Jun 30 16:12:08 ip-172-31-93-122 env[2730]: Started wazuh-syscheckd...
Jun 30 16:12:09 ip-172-31-93-122 env[2730]: Started wazuh-logcollector...
Jun 30 16:12:10 ip-172-31-93-122 env[2730]: Started wazuh-modulesd...
Jun 30 16:12:12 ip-172-31-93-122 env[2730]: Completed.
Jun 30 16:12:12 ip-172-31-93-122 systemd[1]: Started Wazuh agent.
[ubuntu@ip-172-31-93-122:~]$
```

Figura A.1. Scenariu activare agent Wazuh