UNIVERSITETI POLITEKNIK I TIRANËS
FAKULTETI I TEKNOLOGJISË SË INFORMACIONIT
DEPARTAMENTI I ELEKTRONIKËS DHE INFORMACIONIT

RENALDA KUSHE

Për marrjen e gradës

"Doktor"

Në "Teknologjinë e Informacionit dhe Komunikimit"

Drejtimi Telekomunikacion dhe Inxhinieri Informacioni

DISERTACION

TEKNIKAT E VLERËSIMIT TË MBROJTJES KIBERNETIKE

Udhëheqës shkencorë

Prof.As. ADRIAN SHEHU

TIRANË, 2015

Udhëheqësi shkencor: Prof.As. Adrian SHEHU

Juria

_____, Kryetari i Jurisë së Disertacionit të Doktoratës

_____, Anëtar Jurisë së Disertacionit të Doktoratës

_____, Anëtar Jurisë së Disertacionit të Doktoratës

_____, Anëtar Jurisë së Disertacionit të Doktoratës

_____, Anëtar Jurisë së Disertacionit të Doktoratës

I pranuar nga

_____, Dekan, Fakulteti i Teknologjisë së Informacionit

Miratuar nga

Akademik. Jorgaq KAÇANI, Rektori i UPT

_____, Këshillli i Profesorëve, FTI

...familjes sime...

Mirënjohje…

Dua të shpeh mirënjohjen time të thellë për udhëheqësin shkencor të këtij punimi, Prof.As. Adrian Shehu; për drejtimin e kërkimit tim shkencor në fushën e sigurisë kibernetike, një fushë sa e vështirë, aq dhe e bukur; për ndihmën e madhe që më ka dhënë përmes sugjerimeve dhe vërejtjeve tepër profesionale, të cilat gjithmonë më kanë frymëzuar për të punuar me tej.

Një falenderim i singertë shkon për Prof. As. Algenti Lala dhe për kolegët e mij në Departamentin e Elektronikës dhe Telekomunikacionit, me të cilët kam kënaqësinë të punoj prej gati një dekade, mbështetja e të cilëve nuk ka munguar asnjëherë.

Gjithashtu, do doja t'i shprehja mirënjohjen time, në mënyrë të veçantë Prof.As. Vladi Koliçi për mbështetjen dhe këshillat me vlerë në kuadër të përmirësimit të vazhdueshëm të punimit.

I jam mirënjohës prindërve të mij për gjithçka që unë jam sot. Dua të falenderoj familjen time, Ilirin, Enon dhe Arbin dhe kërkoj falje për kohën e munguar. Ndihem shumë e bekuar nga prezenca juaj në jetën time.

Mbi të gjitha dua të falenderoj Zotin, për gjithçka !

Renalda Kushe

PËRMBAJTJA

LISTA E FIGURAVE

LISTA E TABELAVE

ABSTRACT

The recent reports of ITU (International Telecommunication Union) and ENISA (European Network Information Security Agency) organisations present a critical situation of the rapidly growing threat of cyber attack. Their recommendations emphasize the need for standartisation in cyber defense and the establishment of protective barriers for networks and systems. It was these recommendations the main motivation of this research work. Our motivation is to "rise" defense techniques for networks and systems and to design new improved defense techniques for more reliable and secure communication.

In this work, we focus on IDS systems (Intrusion Detection Systems) as a major technique widely used in various fields for both network based and host based defense. IDSs are categorized into two categories based on detection techniques they use: misuse detection and anomaly detection. These techniques have their advantages and limitations. The main advantage of Misuse IDSs is low number of false positives raports (alerts generated by mistake from classifying a legal activity as an attack), but fail to detect previously unknown intrusions (since it attempts to detect only known intrusions based on predefined intrusion characteristics). Anomaly IDSs have the advantage of detecting novel intrusions (it detects anomaly activity based on the deviation of the process behaviors), but it generates high false positives alerts.

The purpose of this work is to propose a new IDS system which combines both detection techniques: misuse detection (misuse IDSs) and anomaly detection (anomaly IDSs). This new detection system contains some new proposed additional mechanisms that will improve the algorithm's performance. The aim is to overcome the limitations of the above techniques and more specifically to increase the detection capability and reduce the number of false positives alerts. We have named the new proposed system Ad-IDS (which stands for Advance IDS).

First, we provide a comprehensive review of the current IDS systems, which implement both of the detection techniques: misuse IDS and anomaly IDS, respectively the Snort IDS system and the Bro IDS system.

Then we move on to the analysis, the design and the implementation of the Ad-IDS system. Through these life-cycle phases of the Ad-IDS system we present its detection algorithm, a mathematical approach and its structure with all the compound modules.

In the end, we make a comparative experimental evaluation between Ad-IDS and current IDSs and give the suggested conclusions. The evaluation is based on the detection capability, on the system's performance (processing time) and on the number of false positives reports generated.

PËRMBLEDHJE

Raportet e fundit të organizatave si ITU (International Telecommunication Union), apo ENISA (European Network Information Security Agency), paraqesin një situatë aktuale kritike, në lidhje me sigurinë kibernetike. Në rekomandimet e tyre, ato venë theksin në nevojën e standartizimit të sigurisë si dhe të ngritjes së barrierave mbrojtëse ndaj rrjetave dhe sistemeve. Duke u nisur nga këto rekomandime, ka lindur dhe motivimi i këtij punimi. Motivimi ynë është nevoja për mbrojtje të rrjetave dhe sistemeve dhe zhvillimin të teknikave të reja të përmirësuara për detektimin e sulmeve kibernetike.

Në këtë punim, ne jemi fokusuar në sistemet IDS (Intrusion Detection Systems), duke qenë se janë mjaft popullorë dhe të përdorshëm në trajta të ndryshme për mbrojtjen e rrjetave (Network based) dhe për mbrojtjen e sistemeve (Host based). Sistemet IDS ekzistuese kategorizohen sipas teknikave që ato implementojnë në: teknika e detektimit të ndërfutjeve (misuse IDS) dhe teknika e detektimit të anomalive (anomaly IDS). Këto teknika kanë avantazhet dhe problematikat e tyre. Teknika e detektimit të ndërfutjeve, ka avantazh numrin e vogël të alarmeve False Positive të gjeneruar (janë alarme të cilët gjenerohen gabimisht, pavarësisht se në fakt kemi të bëjmë me aktivitet legjitim) dhe problematikë: pa-aftësinë e detektimit të ndërhyrjeve të reja (duke qenë se detektimin e kryen duke u bazuar mbi një bazë të dhënash me modele të ndërhyrjeve të njohura); Teknika e detektimit të anomalive ka avantazh aftësinë detektuese të ndërhyrjeve të reja (sepse e bazon detektimin tek ndryshimi i sjelljes së proceseve) dhe problematikë: gjenerimin e shumtë të alarmeve False Positive.

Qëllimi i këtij punimi është propozimi i një sistemi të ri IDS, i cili në vetvete paraqet një kombinim të dy teknikave kryesore të detektimit: teknika e detektimit të ndërfutjeve (misuse IDS) dhe teknika e detektimit të anomalive (anomaly IDS). IDS-së së propozuar do i shtohen edhe disa mekanizma shtesë të cilat do të përmirësojnë performancën e algoritmit. Qëllimi i krijimit të këtij sistemi është evitimi i problematikave që paraqesin dy teknikat e mësipërme dhe më saktësisht: të rrisim aftësinë detektuese si dhe të ulim numrin e alarmeve të tipit False Positive. Sistemin e ri të propozuar e kemi emërtuar Ad-IDS (Advance IDS).

Fillimisht, në punim trajtohen sistemet IDS ekzistuese, të cilat implementojnë dy teknikat e detektimit të ndërhyrjeve: teknika e detektimit të ndërfutjeve (misuse IDS) dhe teknika e detektimit të anomalive (anomaly IDS), përkatësisht sistemet Snort IDS dhe Bro IDS.

Më pas kalojmë në analizën, projektimin dhe implementimin e sistemit Ad-IDS. Në këto faza të ciklit të jetës së sistemit Ad-IDS japim algoritmin e tij, modelin matematikor si dhe strukturën me modulet përbërëse të tij.

Në pjesën e fundit të punimit jepet vlerësimi eksperimental krahasues i Ad-IDS, në lidhje me IDS ekzistuese si dhe konkluzionet e arritura. Vlerësimi eksperimental është mbështetur në aftësinë detektuese, në performancën e sistemit (koha e procesimit) si dhe në numrin e alarmeve False Positive të gjeneruara.

KAPITULLI 1
HYRJE

1.1 Hyrje

Shtrirja Globale e Internetit dhe përdorimi masiv nga një numër gjithmonë në rritje përdoruesish nxjerr në pah Krimin Kibernetik si një sfidë bashkëkohore të udhëhequr nga një gamë e gjerë faktorësh social-ekonomike-politike.

Sipas statistikave të ITU [1] lidhja në Rrjet dhe shfrytëzimi i shërbimeve të ofruara ka pasur një rritje eksponenciale në vitet e fundit.

Krimi kibernetik paraqet një sfidë për shoqërinë e sotme. Përdorimi i teknologjive të reja të informacionit dhe veçanërisht i Internetit ka marrë një rëndësi të veçantë në jetën e përditshme. Ky fenomen prek jo vetëm aktivitetet e një organizmi qoftë ai shtetëror apo privat, i implikuar në sferën e biznesit apo të një aktiviteti jo-fitimprurës, por edhe njeriun e thjeshtë në aktivitetin e tij të përditshëm, në sferën e tij private apo profesionale. Si çdo teknologji e re e venë në dispozicion të një numri të madh përdoruesish, Interneti paraqet jo vetem të mira dhe përfitime, por në të njejtën kohë dhe një sërë problematikash, ku ajo që spikat më së shumti është **SIGURIA**.

Në Vitin 2011, më tepër se 2.3 bilion njerëz, ekuivalentja e me tepër se 1/3 e popullsise Globale ka lidhje në Internet.

Më tepër se 60% e përdoruesve janë nga vendet në zhvillim dhe 45% e tyre jane te moshës 25 vjeç e poshtë.

Në vitin 2017, parashikohet që përdoruesit e Internetit të shkojnë në 70% të popullsisë Globale. [2]

1.1.1 Sulmet dhe kërcënimet më të spikatura të 2014

Dobësitë dhe të metat e sistemeve dhe rrjetave, kanë qenë gjithmonë një pjesë e rëndësishme e gjithë kornizës së sigurisë, ku sistemet operative dhe pjesët që lidheshin me kërkimin në browser (siguria e aplikacioneve, sidomos atyre web), kanë qenë kritike në mbajtjen e sigurt të informacionit. Gjithësesi, vëmendja u zhvendos gjatë 2014, në mënyrën se si "hakerat" munden të fitojnë akses.

"Data breaches" (sulmi dhe marrja e të dhënave) është ende një çështje e rëndësishme, duke qënë që ky sulm u rrit me më shumë se 23 % në 2014. Sulme të ndryshme si Heartbleed, ShellShcok dhe Poodle (sulme të reja të detektuara në 2014) dhe popullaritetit të tyre në një numër të madh sistemesh operative, e risolli temën në një këndvështrim tjetër. *Në 2014, këndvështirimi i ri është transformuar nga mënyra se si "threat-i" X zbulon një dobësi të një sistemi; në mënyrën se si dobësia Y është përdorur nga këto "threat-e" në këto sulme.*

Mund ta quajmë vitin 2014 si vitin serioz të "hyrjes" në cloud dhe një ngjarje e rëndësishme në mendjen e njerëzeve është ajo e "thyerjes" së sigurisë së Apple iCloud, ku një numër i fotove të njerëzve të famshëm u "hakuan" dhe u shpërndanë online. Dhe menjëherë sapo kjo ishte përfunduar si sulm, një tjetër sulm ndodhi në China iCloud ku u tentua të merreshin të dhënat kredinciale të përdoruesve dhe për të marrë informacion për aktivitetet e tyre. [3]

1.1.2 Impakti në fusha të ndryshme të jetës

Krimi kibernetik shoqërohet gjithmonë me një impakt të fuqishëm social, politik dhe ekonomik. [2][3]

- Impakti social u pa shumë qartë në 2014 me marrjen e fotove nga llogaritë cloud të personave të njohur dhe publikimi i tyre në një numër të madh websitesh. Normalisht ky sulm preku drejtpërdrejtë reputacionin e personave publik të përfshirë në këto foto si dhe theksoji dhe njëherë se askush nuk është i paprekur nga kjo luftë.
- Përsa i përket impaktit politik ai lidhet drejtpërsëdrejti me luftën kibernetike që zhvillohet mes shteteve dhe dëmtimi e marrëdhënieve të tyre diplomatike. Rasti i përfolur mes SHBA, Rusisë dhe Kinës nxorri në pah marrëdhëniet e tendosura mes këtyre shteteve.

- Dhe kur vjen fjala për impaktin ekonomik, shifrat nuk gënjejnë kurrë. Ndalimi i krimit kibernetik –kostoja e të cilit përllogaritet në 445 bilion $, rrjedhimisht do të ketë një ndikim në ekonominë botërore. Kostoja totale që lidhet me marrjen e informacionit personal arrin në 160 bilion $. Gjithashtu humbjet dhe pasojat nga krimi kibernetik përfshin dhe ato të lidhura me pastrimin dixhital dhe elektronik që ndodhin pas një sulmi. Në Itali për shembull kostoja totale llogaritet në 875milion $ ndërkohë procesi i "recovery" arrin në 8.5bilion $. Studimet tregojnë se ekonomia e Internetit gjeneron 2-4 trilion $ dhe pritet që kjo vlerë të pësojë rritje. Në raportet e CSIS (Center for Strategic and International Studies) [41] tregohet se krimi kibernetik nxjerr 15% deri në 20% të vlerës që gjenerohet nga Interneti. Kostoja më e rëndësishme e krimit kibernetik vjen nga pasojat e tij në perfomancën e kompanisë dhe ekonomisë kombëtare, nuk duhet neglizhuar fakti që krimi kibernetik dëmton tregjet, konkurrencën, inovacioni dhe rritjen globale ekonomike. Krimi kibernetik është nje taksë për inovacionin dhe e ngadalëson ritmin e inovacionit global duke reduktuar shpejtësine e kthimit të fitimit te investitorët. Efekti është dhe zhvendosja e punësimit nga punët që krijojnë vlera. Krimi kibernetik ka pasur nje impakt prej 200000 vende pune në SHBA dhe 150000 në vendet e BE. Me kalimin e viteve krimi kibernetik është shndërruar në një industri fitimprurëse por kjo mund të ndryshojë me një bashkëpunim në rritje mes shteteve dhe partnerëve privat.

Me bindje mund të themi se e vetmja gjë konstante në fushën e Sigurisë Kibernetike është ***ndryshimi*** *i vazhdueshem. Kjo mund të shihet qartësisht në vitin 2014, një vit me shumë dobësi të shfrytëzuara. Nqs 2013 u pa si viti i Mega Breach, 2014 u pa dhe u cilësua si viti ku sulmet e një niveli të lartë "profesional" dominuan titujt e gazetave dhe edicioneve të lajmeve.*

1.1.3 Strategjitë Globale, plan-veprimi Europian dhe nevoja për Standartizim

Komisioni Europian publikoi strategjinë e mbrojtjes kibernetike të Bashkimit Europian (EU CSS) ne 4 shkurt 2013 [4][5]. Duke vepruar kështu, komisioni njohu dhe iu përgjigj nevojës për të sjellë komunitete të ndryshme bashkë, për të përmirësuar qasjen ndaj sigurisë kibernetike në të gjithë BE-në dhe ka hedhur themelet për një qasje më të koordinuar. Strategjia e sigurisë kibernetike e BE-së gjithashtu përfshin një propozim për një direktivë në "Network and information security" (NIS), i cili do t'i kërkoj një shteti anëtar (MS) që do të ketë aftësi minimale NIS në vend dhe duhet të bashkëpunojë dhe të shkëmbejë informacion me një rrjet të dedikuar dhe t'i kërkojë sektorit privat të adaptojë NIS. Strategjia përmban pohimet e mëposhtme:

- BE-ja riafirmon rëndësinë e entiteteve komerciale dhe jo qeveritare të përfshira në menaxhimin ditë-pas-dite të standarteve të internetit.
- Në fokus kryesor duhet të jetë në krijimin e iniciativave të cilat mbajnë një risk manaxhimi të duhur dhe adaptojnë standarte sigurie dhe zgjidhje.
- Komisioni do të suportojë zhvillimin e standarteve të sigurisë dhe të ndihmojë me skemën e çertifikimit vullnetar në fushën e "cloud computing".

Në objektivin strategjik të katërt të përcaktuar në raport, komisioni i kërkoi ENISA [5] për të zhvilluar në bashkëpunim me autoritetet kompetente nacionale, palët e interesuara, organet ndërkombëtare dhe europiane të standartizimit dhe qendrës së kërkimit të komisionit europian, udhëzime teknike dhe rekomandime për adaptimin e standarteve NIS dhe praktikave të mira në sektoret privat dhe publik.

Ky është një rekomandim i njohur si ENISA e re, e cila i jep agjensisë një rol proaktiv në këtë fushë. Rregullorja e re e ENISA në këtë fushë ngarkon ENISA-n për të mbështetur zhvillimin dhe standartizimin, duke lehtësuar krijimin dhe duke marrë parasysh standartet europiane dhe ndërkombetare të menaxhimit të riskut dhe për sigurinë e produkteve elektronike, rrjetave dhe shërbimeve. Ka dhe rekomandime për aktorët private dhe publike. Në veçanti komisioni inkurajoi palët e interesuara private dhe publike për të :

- Stimuluar zhvillimin dhe adaptimin e standarteve të sigurise në industri, normat teknike dhe sigurisë nga dizanji , privatësisë nga dizanji parimisht nga prodhuesit e produkteve ICT dhe ofruesit e shërbimeve duke përfshirë ofruesit "cloud" dhe paisjen e brezave të rinj të softuerve dhe harduerve me siguri më të fortë.

- Zhvillimin e standarteve për performancën e kompanive në sigurinë kibernetike dhe përmirësimin e informacionit të disponueshëm për publikun duke zhvilluar etiketat e sigurisë.

Një pjesë e rëndësishme e sigurisë kibernetike është propozimi për një rrjet dhe direktivë për informacionin e sigurise (NIS). Direktiva kërkon nga shtetet anëtare (MS) të suportojnë standartizimin në hapësirën e NIS:

- Duke patur parasysh natyrën globale të problemeve NIS, ka një nevojë për një bashkëpunim ndërkombëtar për të përmirësuar standardet e sigurisë dhe këmbimin e informacionit, dhe për të promovuar një qasje të përbashkët globale ndaj çështjeve NIS.
- Standardizmi i kërkesave të sigurisë është një proces i drejtuar nga tregu. Për të siguruar një aplikim konvergjent të standardeve të sigurisë, Shtetet Anëtare duhet të inkurajojnë përputhjen me standardet e përcaktuara, për të siguruar një nivel të lartë sigurie në nivel BE-je. Për këtë qëllim, mund të jetë e nevojshme të hartojmë standarte të harmonizuara.

Përveç kësaj, neni 16 mbi standardizimin thotë:

"... Shtetet anëtare do të inkurajojnë përdorimin e standarteve dhe / ose specifikimeve për rrjetet dhe sigurinë e informacionit. "

"Komisioni do të hartojë, me anë të zbatimit të akteve një listë të Standardeve të përmendura në paragrafin 1. Lista, do të botohet në fletoren Zyrtare te Bashikimit Europian".

1.1.4 Rekomandimet e Agjensive

Rekomandimet e mëposhtme të përgjithshme për zhvillimin dhe përdorimin e standarteve mund të ndihmojnë shtetet antarë të NATOS në shumë fusha kritike të sigurisë kibernetike dhe mbrojtjes kibernetike. Këto variojnë nga proceset e standartizimit dhe forcimi i rregullave, në përcaktimin e praktikave efiçente për verifikimin e sigurisë në sistemet përkatëse të sigurisë kombëtare, në identifikimin e standarteve për fusha specifike të R&D. [1][4][5] Rekomandimet jane si më poshtë:

- Politikë-bërësit duhet të vazhdojne të inkurajojnë përdorimin e standardeve, dhe të inkurajojnë organizatat e sektorit privat dhe publik për të përfshirë referenca për këto standarde në proceset e prokurimit.
- Qeveritë duhet të përfshijnë standartizimin si pjesë të strategjisë për sigurinë kibernetike nacionale. Theksi duhet të vihet te përmiresimi i koordinimit midis politikës dhe niveleve operacionale dhe rrjitjen e rolit të partnerve publik-privat në procesin e standartizimit.
- Autoritetet rregullatore nacionale duhet të përdorin më shumë standarte si një pikë reference në forcimin e rregullave.
- Institucionet publike të përfshirë në financimin e kërkimit dhe zhvillimin duhet të identifikojnë grupe të qëndrueshme standartesh për fusha të ndryshme kërkimore.
- Organizatat e zhvillimit të standarteve duhet të punojnë së bashku për të identifikuar mënyrat e përshpejtimit të procesit të zhvillimit të standarteve për sigurinë kibernetike. Kjo mund të arrihet me anë të një mekanizmi “fast track”.
- Qeveritë e vendeve duhet të bashkpunojnë për të përcaktuar një skeme çertifikimi që i lejon përdoruesve fundor të verifikojnë që shërbimi apo produkti është në përputhje me standartet e sigurise.
- Mbështetja financiare e qeverive dhe promovimi i zhvillimit të kërkimit shkencor në fushën e sigurisë kibernetike, me qëllim krijimin e teknikave të reja mbrojtëse, si dhe përmirësimi i atyre ekzistuese.

1.2 Punime të deritanishme në sistemet IDS

Sistemet e Detektimit të Ndërhyrjeve IDS (Intrusion Detection System) janë një ndër teknikat kryesore, të përdorura, për mbrojtjen nga sulmet kibernetike. Detektimi i ndërhyrjeve është një fushë kërkimore mjaft e lëvruar, për vite të tëra, me arritje të jashtëzakonshme në drejtim të zbulimit të rasteve të ndërhyrjeve, përmes teknikash të ndryshme. Në një artikull mbi një studim mbi sistemet IDS, Stefan Axelsson [6] përshkruan dhe krahason sistemet e hershme IDS. Këto sisteme fokusoheshin kryesisht në detektimin e ndërhyrjeve që lidhej me veprimtarinë e përdoruesve. Aftësia e tyre detektuese bazohej në regjistrat (logs) e komandave të sistemit operativ (fillimisht janë implementuar në sistemin e operimit UNIX), të thirrura nga përdoruesit individualë.

Bazuar në teknikat e detektimit, IDS-të ndahen në dy kategori: detektimi i ndërfutjeve (misuse detection) dhe detektimi i anomalive (anomaly detection).

Në çështjet në vijim, synojmë të jepim një panoramë të punës kërkimore-shkencore të realizuar në këtë fushë. Kjo panoramë është bazuar në një analizë gjithëpërfshirëse të punimeve kërkimore, të realizuara mbi sistemet IDS duke u bazuar në algoritmat e detektimit të përdorura. Duke u nisur nga këto të fundit, do të paraqesim dhe problematikat dhe kufizimet e hasura.

1.2.1 Sistemet IDS të bazuara në detektimin e ndërfutjes (misuse detection)

Kjo teknikë është qasja më e përhapur që përdoret në botën komerciale të IDS-ve. Ideja themelore e saj është *identifikimi i ndërhyrjeve në burime të ndryshme të të dhënave që monitorohen, duke shfrytëzuar modele të njohura sulmesh*. Për pasojë, IDS-të e bazuara në këtë teknikë detektimi tentojnë të zbulojnë vetëm sulmet me karakteristika të njohura e të paracaktuara [7].

Një sulm mund të shfaqet në trajta të ndryshme, ndaj saktësia e këtyre IDS-ve varet, në mënyrë eksplicite, nga mënyra se si paraprocesohet dhe "shërbehet" në motorin e detektimit të IDS, informacioni i përvetësuar mbi sulmin. Një set modelesh "të mirë-hartuara" e "të mirë-përzgjedhura" mundëson një shkallë të lartë saktësie të sistemeve IDS me detektim të ndërfutjes, duke ulur probabilitetin e dështimeve. Teknika e detektimit të ndërfutjeve bazohet në metodat e mëposhtme:

- **Metoda e bazuar në "vulën e sulmit" (Signature based approach);** Kjo metodë, funksionon në mënyrë të ngjashme me softuerin e një antivirus-i. Në këtë metodë, specifikat mbi analizën e karakteristikave kuptimore të një sulmi përdoren për të krijuar një "vulë" (signature) të tij [8]. Modeli i sulmit ndërtohet në mënyrë të tillë që të mund të detektohet duke përdorur informacionin e auditit të regjistrave të të dhënave, të gjeneruara nga sistemet kompjuterike. Nisur nga sulmet e njohura e të mirë përcaktuara mund të krijohet një bazë të dhënash me modelet identifikuese të tyre, e cila të mund të përdoret në zbulimin e sulmeve nga motori i detektimit i IDS-ve, përmes krahasimit të saj me vargun e të dhënave të auditit ose regjistrave (logs). Sa herë që detektohet një sulm i ri, baza e të dhënave duhet të pasurohet menjëherë me modelin e ri, për të rritur shkallën e saktësisë së detektimit. IDS-të algoritmat e të cilëve bazohen në këtë metodë, jepen si më poshtë:

  Snort [9][10] është një IDS, nga më të njohurit (open source), i bazuar mbi modelet e ndërfutjes. (Snort është IDS në të cilën ne jemi bazuar për të krijuar IDS-në e propozuar në këtë punim). Snort mund të konfigurohet në mënyrat: "thithësi" i paketave (packet sniffing mode); që bën të mundur monitorimin dhe shfaqjen e paketave të trafikut të rrjetit; "rregjistruesi" i paketave (network traffic logger mode); në të cilën Snort hedh në një skedar të dhënat e regjistrit të trafikut në rrjet; IDS mode, ku ndërthuren aftësitë e detektimit dhe parashikimit të ndërhyrjeve, bazuar në specifikat e njohura të sulmeve. (është dhe IPS- Intrusion Prevention System mode e cila është e papërfunduar dhe akoma në zhvillim). Snort përdor një bazë të dhënash, që konsiston në rregulla të përcaktuara nga përdoruesi dhe modelin e algoritmit krahasues të Boyer-Moore të çdo pakete trafiku të rrjetit me database-n [9]. Në ndryshim nga sisteme të tjera IDS të bazuara te modeli, Snort analizon shtresën e aplikimit të rrjetit për të detektuar modele specifike të sulmeve të mirënjohura, si psh. mbingarikimi i memories buffer (buffer overflow), skanimi i portave (port scan), etj. [9]. Nëse konstatohet ngjashmëri, Snort sinjalizon përdoruesin, regjistron informacionin e paketës dhe mund të ndërmarrë veprime të paracaktuara nga përdoruesi si p.sh. rrëzimi (drop) i kësaj pakete. Për të rritur më tej aftësitë detektuese-parashikuese, Snort mund ta kombinojmë me IDS të tjera të hapura, por këto aftësi kufizohen brenda kuadrit të rregullave të ofruara nga database-i i modeleve e për pasojë, sulmet e reja në rrjet nuk mund të detektohen.

Haystack [11] u krijua si një nga versionet më të hershme të IDS-ve, të mbështetur në modele, kryesisht për monitorimin e sistemeve me shumë përdorues, në rrjetet kompjuterike të Forcave Ajrore [6]. Prototipi i parë i sistemit u projektua për të detektuar vetëm gjashtë tipe specifike sulmesh, përkatësisht tentativat për thyerjen e sigurisë (break-ins), sulmet maskaradë, penetrimin e paautorizuar në sistemin e kontrollit të sigurisë, rrjedhjen e të dhënave sensitive, refuzimin e shërbimit të sulmit dhe përdorimet e dyshimta të sistemit [6] [11]. Politikat e sigurisë u shndërruan në rregulla, të cilat u ruajtën në një bazë të dhënash, për krahasimin me këtë të fundit të çdo sesioni të ri të të dhënave të auditit dhe detektimin, në rast shkelje të rregullave të paracaktuara të detektimit të ndërfutjes. Megjithatë, sistemi është një IDS i mbështetur në një profil përdoruesi, jo në kohë reale, me aftësi të limituara [6].

Network Flight Recorder (NRF) [12] është një mjet i fuqishëm i detektimit dhe analizës së ndërhyrjeve, me bazë rrjetin, i përdorur për qëllime komerciale. Ai përdor modelet e disa sulmeve të njohura për të gjeneruar sinjale alarmi në rast detektimi të një ndërhyrjeje potenciale. Ndryshon nga Snort në faktin se NRF është një sistem më i kompletuar monitorimi dhe analizimi i rrjetit. NRF përdor gjuhën e tij specifike, e ashtuquajtur “n-code”, për gjenerimit e modeleve dhe analizën e paketave të rrjetit.

- **Metoda e bazuar në rregulla (rule based)** Shumica e sistemeve detektuese të ndërfutjes, përdorin qasje të bazuar në rregulla të përcaktuara [7]. Sisteme të tilla ndërtohen mbi bazën e një numri cilesh kushtëzimesh (if-then) për teknikat e detektimit. Rregullat krijohen përmes analizës së ndërhyrjeve dhe shndërrimin e tyre në kushte të domosdoshme për t’u përdorur në modulet e ndërfaqes së IDS-ve, për krahasimin me të dhënat e monitorimit (zakonisht logs).

  Një sistem ekspert i detektimit të ndërhyrjeve në kohë reale **IDES (Instruction-Detection Expert System)** [13] është një nga sistemet tipike të detektimit të ndërfutjes mbi bazën e rregullave. Sistemi përdor metoda shumë variantëshe për analiza statistikore të karakteristikave të sjelljes së përdoruesve, përmes të dhënave të auditit të veprimtarisë së tyre. Përmes këtyre statistikave, sistemi harton profilin e një sjelljeje normale në rrjet për secilin grup përdoruesish, në bazë të atributeve të përdoruesve të secilit grup, sipas parimit se përdoruesit e grupit gëzojnë shkallë të njëjtë atributesh. Sistemi përdor një nënsistem statistikor për të monitoruar e krahasuar sjelljen e çdo përdoruesi me historikun e

veprimtarisë së tij në rrjet dhe me një sistem referimi, të bazuar në rregulla, i përbërë nga sjellje të pritshme, të konsideruara normale, për grupin e përdoruesve ku ai ben pjesë. Nëse veprimtaria e përdoruesit nën vëzhgim shmanget me një nivel të caktuar nga sjellja normale, duke thyer ndonjë nga rregullat e sistemit, ajo klasifikohet si ndërhyrje. IDES kishte aftësinë të detektonte raste ndërfutje ose sulmi nga përdorues të autorizuar, të cilët abuzonin me atributet e tyre.

Sistemi P-BEST [14] **(Production-Based Expert System Toolset),** është një sistem i specializuar i bazuar në rregulla, i cili konsiston në një interpretues rregullash dhe një set rutinash të ndryshme. Faktet dhe rregullat rreth përdoruesit interpretohen nga një sistem ekspert zinxhir në P-BEST. Sistemi P-BEST përdoret për detektimin e rasteve të ndërfutjes në kompjutera dhe rrjeta [14]. Sa herë që shtohet një fakt i ri, seti i rregullave duhet të rishikohet e rivlerësohet [13]. Kjo është një procedurë, e cila kërkon kohë [7].

EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [15] përdor gjithashtu sistemin ekspert P-BEST dhe nënsistemin e tij të detektimit të modelit mbi bazë rregullash.

NIDES (Next Generation Intrusion Detection System) [16] është i ndërtuar mbi modelin klient-server, në të cilin të dhënat e disa nyjeve (host-ve) të një rrjeti grumbullohen në një nyje (host) specifike.

"AT&T's ComputerWatch" [17] është një mjet i analizës së të dhënave të auditit (audit log) si dhe një IDS, i cili përdor rregulla të paracaktuara dhe bën krahasimin me to të veprimeve të përdoruesve, të dyshuara për rreziqe sulmesh.

- **Metoda me tranzicion të gjendjes (state transition);** Bazuar në makinat me gjendje të fundme, metoda e tranzicionit të gjendjes përdoret në shumë punime kërkimore mbi detektimin e sulmeve ose ndërfutjeve të kompjuterave [18][19]. Në këtë teknikë, ndërhyrjet konsistojnë në kryerjen e një sërë veprimesh, të cilat veç e veç ose të kombinuara, mund të shkaktojnë tranzicion nga njëra gjendje e sensorit të monitorimit në një tjetër (termi *një sensor monitorimi* i referohet një hosti i cili implementon një IDS të tipit host-based), duke arritur gjendjen përfundimtare të sigurisë së sistemit të monitorimit. STATL (State Transition Analysis Technique Language) [19] është një gjuhë e bazuar në teknikën e

tranzicionit të gjendjes që i përkufizon sulmet si një seri veprimesh të kryera me qëllimin për të depërtuar në një sistem kompjuterik. Specifikimi i STATL për ndërhyrjen përfshin atributet kyçe, të mjaftueshme për ta konsideruar atë të pavaruar nga cilido mjedis apo sistem operimi. Specifikimet e STATL mbi ndërhyrjet mund të implementohen në çfarëdo sistemi operativ shfrytëzimi për identifikimin si të sulmeve me qendër rrjetin, ashtu edhe të atyre me qendër host-in.Teknika e analizës së tranzicionit të gjendjes STAT (State Transition Analysis Technique) [19] është një bashkësi mjetesh për detektimin e rasteve të ndërfutjeve, i cili përdor mekanizmin e tranzicionit të gjendjes për të identifikuar veprimtari ndërhyrjeje në sistemet kompjuterike. STAT përfshin gjuhën STATL për të përcaktuar skenaret e sulmeve përmes atributeve të pavaruara domain të tyre, të një gjuhe abstrakte të nivelit të lartë. Këto përcaktime duhet të përmbahen nga zhvilluesit e sistemit të sigurisë për përmbushjen e nevojave për një mjedis specifik (si një nyje, rrjet specifik ose sistem operativ). Ideja themelore e detektimit të një sulmi ose të një rasti keqpërdorimi është: kur një sistem kompjuterik, i ndodhur në një gjendje fillestare të sigurtë, sulmohet nga një varg veprimesh nga ana e "agresorit" me qëllim depërtimin në të, ai mund të pësojë tranzicion, nëpër disa stade gjendjesh, para se të vendoset gjendja përfundimtare.

Mjeti i analizës së tranzicionit të gjendjes **UNIX USTAT (UNIX State Transition Analysis Tool)** [20] është i pari STAT që analizon auditin e të dhënave të sistemeve të bazuara në UNIX, për identifikimin e ndërfutjeve në kohë reale. USTAT kishte fillimisht aftësinë të analizonte këto të dhëna, vetëm për një host të sistemit UNIX dhe me zhvillimin e aftësive të analizës së të dhënave të auditit për shumë sisteme UNIX, evoluoi në NSTAT [20]. Ky i fundit përdor procesimin e shpërndarë për të grumbulluar regjistrimet e auditit për nyjet (host-et) dhe i proceson ato në një sistem qendror, për detektimin e ndërfutjeve dhe ngjarjeve në një mjedis të shpërndarë.

WinSTAT [18] është gjithashtu një mjet detektues i bazuar në STAT, për nyjet e mjedisit të shfrytëzimit Windows NT.

NetSTAT [19], një sistem dektimi në kohë reale i bazuar te rrjeti, mbështetet gjithashtu në strukturën e STAT. NetSTAT përdor topologjinë e rrjetit si një model hiper grafik dhe përkufizimet e STAT për sulmet me bazë rrjeti, për të zhvilluar konceptin e keqpërdorimevë ne lidhje me konfigurimin specifik të tij

[7]. Përdor si hyrje input regjistrat e trafikut të rrjetit dhe një paraprocesor për të filtruar paketat e rrjetit për ngjarje të rëndësishme të tij dhe për gjenerimin e ngjarjeve abstrakte [7]. USTAT dhe NetSTAT patën performancë të mirë për vlerësimin off-line të sistemit të detektimit të ndërhyrjeve në MIT Lincoln Laboratory në vitin 1998 dhe gjithashtu në Laboratorin Kërkimor të Forcave Ajrore ARFL (Air Force Research Laboratory), me një nivel të lartë të shkallës së vlerësimit.

- **Metoda "Data mining";** Në punimet kërkimore mbi IDS ka mjaft aplikime të "data mining" për detektimin e ndërfutjeve [21] [22]. Në këtë metodë, të dhënat historike të një sistemi të monitoruar duhet të kategorizohen qartazi dhe në mënyrën e duhur, si të pranueshme, jo të pranueshme dhe të etiketuara, në varësi të rrethanave. Sistemi është i trajnuar për të kuptuar çfarë konsiderohet e pranueshme ose e papranueshme për çdo rrjet ose përdorim të sistemit. Nëse kemi ndonjë shmangie, sistemi prodhon një sinjal alarmi për ndërhyrje. Lee dhe Stolfo [21] aplikuan disa teknika "data mining" si rregulli i shoqërimit (association rule) dhe teknika të ndryshme klasifikimi, për të zhvilluar modele automatike detektimi të ndërfutjeve, duke përdorur të dhënat e auditit dhe thirrjet e sistemit.

  Ata projektuan një strukturë detektimi të quajtur **MADAM ID** (Mining Audit Data for Automated Models for Intrusion Detection). Vlerësimi i tyre eksperimental dha rezultate shumë premtuese [22]. Përparësia e kësaj strukture qëndron në faktin se, bazuar në të dhënat e auditit, modelet mund të ndërtohen në mënyrë automatike.

1.2.2 Sistemet IDS të bazuara në detektimin e anomalive (anomaly detection)

Sistemet IDS të cilët përdorin teknikën e detektimit të anomalive, janë sisteme të cilët punën e tyre e mbështesin tek monitorimi i sjelljes së sistemit. *Në rastin kur evidentohen sjellje të cilat paraqesin "devijim" nga "sjellja normale" e sistemit, atëhere në këtë rast raportohet për ndërhyrje.*

Metodat në të cilat bazohet një sistem i detektimit të anomalive jepet si më poshtë:

- **Metoda e Modelimit Statistikor;** Përdorimi i metodave statistikore në detektimin e anomalive është një nga teknikat më të hershme të aplikuara në

fushën e kërkimit mbi sistemet IDS. Në këtë teknikë, sjellja normale e një përdoruesi përcaktohet nga çfarë konsiderohet e pranueshme për politikat e përdorimit të sistemit. Me anë të teknikave të ndryshme të modelimit statistikor (si modelet e identifikimit të shmangies nga korniza e një sjelljeje normale), sjellja e një përdoruesi në rrjet monitorohet dhe, në rast se konstatohet devijim nga pragu i paracaktuar i sjelljeve normale, veprimtaria e këtij përdoruesi do të konsiderohet ndërhyrje.

Bro [23], një sistem i hapur i detektimit të ndërhyrjeve, me bazë rrjetin UNIX, përdor po ashtu qasjen e bazuar në modelin statistikor [24]. Teknikat e tij të detektimit konsistojnë në identifikimin e sulmeve dhe ngjarjeve të njohura, bazuar në modele sulmesh e ngjarjesh të paracaktuara dhe në konstatimin e veprimtarive të pazakonta (tentativa të dështuara për lidhje). Sistemi Bro përbëhet nga tri shtresa të dallueshme, përkatësisht libcap-libraria e kapjes së paketave për filtrimin e tyre, shtresa gjeneruese e ngjarjeve (event engine)-administron paketat tashmë të filtruara të rrjetit dhe interpretuesi i politikave dokumentuese (policy script interpreter)-manaxhon ngjarjet e gjeneruara nga shtresa e dytë [23].

"Qiao et al." [25] aplikuan teknikën e modelimit statistikor, një model HMM (Hidden Markov Model), në thirrjet e sistemit për të detektuar ndërhyrje anormale. Për të përcaktuar tranzicionet e ndryshme të gjendjeve nëpër të cilat kalon një proces special, i bazuar në sistemin UNIX, ata grumbulluan thirrjet e sistemit, specifike për atë proces dhe i aplikuan të dhënat e grumbulluara në modelin HMM. Nëpërmjet këtyre sekuencave të tranzicionit të gjendjeve, ata ndërtuan një bazë të dhënash me sekuenca të konsideruara normale, e cila do të shërbente si etalon krahasimi me të gjitha thirrjet e monitoruara të sistemit, për detektimin e anomalive. Megjithatë, ndërtimi i një database-i të plotë gjendjesh për secilën nga hyrjet do të ishte e pamundur, për shkak se input-et e thirrjeve të sistemit lidhen një për një me gjendjet e sistemit në dalje (output) [25].

- **Metoda "Machine Learning" ;** Metoda "machine learning" është aplikuar me sukses në kërkimet e kryera për sistemet IDS, për detektimin e anomalive. Punimet më të fundit në këtë fushë kërkimore e përdorin këtë teknikë të ndërthurur me "data mining" për rezultate më të mira të detektimit të ndërhyrjeve të reja.

“Ghosh et al” [26] vlerëson tre algoritme të tipit “machine learning” për detektimin e ndërhyrjeve në kohë reale, bazuar në sjelljen e programit në mjedisin e sistemit Sun Solaris. Rezultatet e tyre eksperimentale, të vitit 1999, për vlerësimin e detektimit të ndërhyrjeve në laboratorin Lincoln Laobratory/DARPA ishin mjaft premtuese për secilin nga tre algoritmet, sa i takon shpejtësisë së detektimit ne kohë reale të ndërhyrjeve.

“Fox et al.” propozon një nga sistemet e para IDS, të mbështetur në rrjetin neural, duke modeluar sjelljet e përdoruesve përmes metodës së të mësuarit jo të mbikqyrur. Ata përdorën hartën e vetë-organizuar SOM (Self-Organizing Map) të Kohonen për të mësuar karakteristikat e një përdoruesi të zakonshëm dhe sjelljen e sistemit në një kompjuter me shumë përdorues. Si input-e në hyrje të SOM mund të shërbejnë okupimi i përgjithshëm i CPU, i memories RAM, numri i logimeve të dështuara, gjatësia e sesionit, numri i përgjithshëm i përdoruesve, numri i aksesimeve në skedarin “Help”, disqet e ndryshëm të aksesuar, etj. [7]. Çdo devijim statistikor nga sjellja normale do të identifikohej si sulm. Megjithatë, performanca e këtij sistemi IDS nuk paraqet ndonjë rezultat mbresëlënës.

“Mohajerani et al” [27] përdori logjikën fuzzy dhe rrjetin neural të ndërthurur me sistemin neuro-fuzzy të detektimit të ndërhyrjeve NFIDS (Neuro-fuzzy Intrusion Detection System). Rrjeti neural u përdor për të mësuar rregullat “fuzzy” për çdo tip sulmi të përcaktuar nga administratori i sistemit. Krijuesit e kësaj metodë pretendonin parashikime të sakta në 90.6% të rasteve dhe dështime vetëm në masën 9.4%, për të dhënat e mbledhura gjatë një periudhë 15 ditore në një rrjet të pambrojtur. Megjithatë, ata nuk specifikojnë natyrën dhe sasinë e trafikut të testuar, tipet e ndërhyrjeve të detektuara dhe aftësinë e sistemit për menaxhimin e shpejtë të trafikut (shpejtësinë e reagimit të sistemit). Për më tepër, për shkak të varësisë nga shkalla e njohjes së ndërhyrjeve në rrjet, nga ana e administratorëve të tij, për përcaktimin e rregullave për secilin tip sulmi, saktësia në detektimin e ndërhyrjeve të reja për një sistem si NFIDS mund të mos jetë e lartë.

“Abouzakhar et al.” [28] propozoi një teknikë të re neuro-fuzzy për detektimin e sulmeve në një rrjet të shpërndarë, si mohimi i shërbimit DoS (Denial of Serive). Sistemi i propozuar mund të memorizonte karakteristikat e trafikut të rrjetit, përmes funksioneve të logjikës fuzzy.

"Chavan et al" [29] propozoi një IDS adaptive neuro-fuzzy për rrjetin IP, në të cilën ndërtohet një bazë të dhënash e përbërë nga modelet e ndërhyrjeve, për të plotësuar bazën e të dhënave të Snort. Këto modele "signatures" u zhvilluan duke analizuar protokollet e rrjetit dhe "të mësuarit adaptive" duke u bazuar në kombinimin e Rrjetave Artificiale Neurale dhe logjiskës Fuzzy. Gjatë gjenerimit të modeleve, ideatorët e sistemit morën në konsideratë disa pika të mangëta të bazës së të dhënave të Snort. Duke përdorur regjistrime të rastësishme nga sistemi i vlerësimit DARPA [30], ata i klasifikuan ato në pesë klasa të ndryshme ndërhyrjesh dhe nxorrën numrin minimal të variablave të duhur, më të domosdoshme për secilën klasë, duke përdorur një pemë vendimesh. Rezultatet eksperimentale të dy prej algoritmeve, njëri prej të cilëve me bazë rrejtin ANN (Artificial Neural Network), me 80 nyje të fshehura dhe tjetri me rrjet EFNN (Evolving Fuzzy Neural Network) i tipit Mamdani, ishin mjaft të kënaqeshme, me një shkallë të lartë detektuese. Specifikat mbi sistemin Mamdani të FIS mund të gjenden në. Megjithatë, IDS të tilla kishin performance të mirë në sete shumë të vogla të dhënash të rrjeteve të para-klasifikuara, e cila përkeqësohej me rritjen e numrit të variablave të input-it. Rrjeti MITRE përdor sensorë IDS të cilët gjenerojnë mbi një milion sinjale alarmi në ditë [31], shumë prej të cilave të rreme, që sjell pamundësinë e shyrtimit të tyre nga ana e eksperteve, për konstatimin e rasteve të mundshme të detektimit të ndërhyrjeve. Kërkimi në drejtim të "data mining", për mbrojtjen ndaj sulmeve kibernetike, po shihet me më shumë interes, për detektimin me saktësi të lartë të anomalive të trafikut në rrjete me sasi të mëdha të dhënash [30] [31].

- **Teknika "Outlier-based Data Mining" ;** Detektimi "outlier" është një nga metodat më të përhapura të "data mining", të përdorura në fushën e kërkimit të IDS. Një numër i madh punimesh kërkimore [31,32,33,34] përbëjnë një rezultat mjaft premtues për detektimin e ndërhyrjeve me anë të kësaj teknike. Ideja themelore është përcaktimi i pjesëve të të dhënave, që dallojnë dukshëm nga sasia tjetër e të dhënave, si të dhëna "outlier". Në shpërndarjen statistikore, të bazuar në një shmangie standarde, të dhënat që nuk përmbahen në diapazonin e përcaktuar, konsiderohen të palejuara. Ka disa metoda për përcaktimin e outlier në shkencat kompjuterike dhe fushat statistikore, specifikimet e të cilave mund të gjenden në.

"Ramaswamy et al." [35] dhe **Petrovskiy** [36] argumentuan një larmi algoritmash efektive për teknikën "outlier mining", në sasi të madhe të dhënash. Sistemi Minnesota i detektimit të ndërhyrjeve (MINDS), i përshkruar në [37, 38], implementon, në mënyrë të suksesshme, këto teknika për detektimin e sulmeve kibernetike në rrjetet kompjuetrike. MINDS përdor këto algoritme në module të ndryshme detektimi. Modulet merren me detektimin e llojeve të ndryshme të sulmeve kompjuterike dhe veprimtaritë e ndërhyrjeve në rrjet. Për shembull, skaneri i tij i detektimit identifikon veprimtari të dyshimta në rrjet, detektuesi i anomalive identifikon trafik rrjeti anormal nga ekzaminimi i header-it të paketës dhe përbërësja e detektimit të profilit i lejon personit që analizon rrjetin të përcaktojë sjelljet anormale në trafikun e të dhënave [38]. Në detektuesin e anomalive, MINDS përdor një algoritëm efikas, LOF (local outlier factor). Specifikimet për algoritmin LOF mund të gjenden në [39]. Rrjedha e të dhënave në rrjet grumbullohet për një dritare specifike dhe të dhënat shërbehen, në radhë, në detektor. Ky i fundit, cakton një rezultat LOF (anormaliteti) për secilën nga rrjedhat e të dhënave, bazuar në rrjedhat fqinje të tyre. Më pas ato renditen në bazë të ketij rezultati dhe është detyrë e ekspertëve të rrjetit t'i etiketojnë këto të dhëna si normale ose jo normale. Rezultatet eksperimentale dëshmuan se MINDS detektonte me saktësi të lartë raste të ndryshme anomalie në trafikun e rrjetit, të paarritura më parë nga IDS të tjera. [9][10]

1.2.3 Përfundime

Kërkimet e bëra mbi mbrojtjen ndaj sulmeve kibernetike, krahas përparësive, kanë edhe të meta e kufizime. Të dy drejtimet e spikatura të kërkimeve, si sistemet IDS të cilat implementojnë teknikën e detektimit të ndërfutjeve ashtu dhe ajo e detektimit të anomalive, kanë mungesë të të dhënave reale të sulmit për testim dhe vlefshmëri, mos marrje në konsideratë të shpejtësisë së detektimit, etj.

Kufizimi kryesor i IDS, bazuar në detektimin e ndërfutjeve (misuse detection) është fakti se mund të zbulojnë me saktësi vetëm raste të njohura ndërhyrjesh dhe paaftësia për të detektuar sulme të reja ose të pashfaqura më parë në rrjet. Për më tepër, duhet të disponohen specifikimet paraprake të sulmit, që kërkon ekspertizë njerëzore për analizën manuale të ndërhyrjeve dhe hartimin e specifikave përkatëse. Kjo është një detyrë që kërkon punë voluminoze dhe harxhim të madh kohor për menaxhimin e gjithë sasisë së madhe të të dhënave që duhen analizuar.

Specifikimet mbi ndërhyrjet mund të gjenerohen në mënyrë automatike nëpërmjet disa teknikave të automatizuara, por shumica e sistemeve të detektimit të keqpërdorimit nuk e kanë këtë aftësi.

Problematika të cilat paraqesin IDS, bazuar në teknikën e detektimit të anomalive është paqartësia në detektim; që nënkupton se shpesh si ndërhyrje konsideron dhe pjesë e informacionit legjitim. Megjithatë, kjo teknikë ka avantazhin e detektimit të sulmeve të reja, të pandodhura më parë.

1.3 Motivimi

Ku punim është motivuar nga sa më poshtë:

- **Nevoja për mbrojtje të rrjetave dhe sistemeve;** Në ditët e sotme, informacioni është shumë i vlefshëm dhe ka një rritje të kërkesës për të aksesuar gjithmonë e më shumë informacion. Kjo kërkesë në rritje për informacion, bën që siguria e tij të jetë gjithmonë e më e cënueshme nga kërcënime dhe sulme të ndryshme. Megjithatë, krahas këtyre sulmeve, paralelisht lind nevoja për zhvillimin dhe aplikimin e teknikave të mbrojtjes kibernetike. Pikërisht, të qenurit pjesë e kësaj "skuadre" (white hat – Ethical hackers) [40], më ka motivuar për zhvillimin e këtij punimi.
- **IDS- Teknika më e përdorur për mbrojtjen kibernetike;** Gjatë studimeve dhe punës kërkimore të zhvilluar për disa vite, jam njohur me disa teknika të mbrojtjes kibernetike si psh: pentest, antiviruset, firewall, metoda enkriptimi, vizualizimi i sigurisë (security visualization), metodat e skanimit të cënueshmërive (Nmap, Armitage, Metasploit), IDS, etj. Gjatë kësaj periudhe, kam konstatuar se sistemet IDS janë mjaft popullore dhe mjaft të përdorura në trajta të ndryshme: si Host- based dhe si Network- Based, pra mënyra e përdorimit është si për mbrojtjen e një hosti, por edhe për mbrojtjen e një rrjeti.
- **Përmirësimi i IDS-ve ekzistuese;** Duke u njohur me IDS të ndryshme përsa i përket teknikave që ato implementojnë, jam ndeshur me avantazhet e tyre kundrejt të tjerave, si dhe me problematikat përkatëse. Duke qenë se një problematike e një IDS-je, përbënte një nga avantazhet e një tjetre, lindi ideja dhe motivimi për krijimin e një IDS-je të përmirësuar, e cila të përmbajë avantazhet e IDS-ve ekzistuese (të teknikave të ndryshme), duke mundur kështu të zgjidhim dhe disa prej problematikave ekzistuese.

Ky punim është i motivuar nga ideja e krijimit dhe zhvillimit të sistemeve të reja të detektimit të ndërhyrjeve, për të rritur mbrojtjen kibernetike.

Synojmë që nëpërmjet këtij punimi, të projektojmë dhe implementojmë një sistem të ri të detektimit të ndërhyrjeve IDS, nëpërmjet të cilit të sjellim një përmirësim në aftësinë detektuese të ndërhyrjeve, në performancen e sistemit (kohë më të ulët procesimi) si dhe të ulim numrin e alarmeve False Positive.

Ky punim, është frut i nje analize të detajuar të teknikave ekzistuese, të cilat të ndërthurura së bashku dhe me disa mekanizma shtesë çojnë në një teknikë të re, e cila trashëgon disa nga vetitë më të mira dhe lejon të jetë më e efektshme pothuajse në të gjitha drejtimet, siç do të shihet në përmbajtjen e kësaj teze.

Sistemi i ri i propozuar dhe i implementuar është quajtur Ad-IDS (Advance Intrusion Detection System). Ad-IDS është një sistem i krijuar nga integrimi i testeve që rrjedhin nga dy teknikat ekzistuese si dhe të disa testeve shtesë të propozuar, në një set të vetëm testesh, i cili është pjesë e modulit të testeve të Ad-IDS. Përveç modulit të testeve, pjesë përbërëse e Ad-IDS është edhe moduli kryesor, i cili ka funksion inicializimin e sistemit si dhe bazën e të dhënave ku mbahen raportet e gjeneruara nga Ad-IDS; është edhe moduli i kundër-masave në të cilin përcaktojmë disa veprime të cilat Ad-IDS i ndërmerr në mënyrë automatike, në lidhje me detektimin e disa ndërhyrjeve.

1.4 Qëllimi dhe kontributi

Bazuar në motivimin e mësipërm, qëllimi i punimit është: *të propozojmë një sistem të ri të detektimit të ndërhyrjeve, të quajtur "Advance IDS" (Ad-IDS) dhe përmiresimin e tij nëpërmjet integrimit të teknikave ekzistuese dhe disa testeve shtesë me qëllim rritjen e aftësisë detektuese, uljen e kohës së procesimit dhe uljen e numrit të alarmeve False Positive.*

Kontributi i kësaj teze është teorik dhe i aplikuar. Nga pikëpamja teorike, ne japim një formulim matematikor të ri, nëpërmjet të cilit japim zgjidhje njërës prej problematikave të teknikave ekzistuese. Për më tepër, sistemi Ad-IDS, me implementimin sipas aparatit të ri matematikor, jep performancë shumë interesante, nga pikëpamja e numrit të alarmeve False Positive të gjeneruara. Nga pikëpamja aplikative, kontributi ynë qëndron në propozimin e një sistemi të ri i cili ka aftësi shumë të mira detektuese dhe kohë procesimi të ulët.

1.5 Struktura e punimit

Për të realizuar çështjen e lartpërmendura, punimi është ndarë në kapitujt si më poshtë:

KAPITULLI 1 – HYRJE

Kapitulli i parë trajton : Situatën aktuale të hapësirës kibernetike, duke u nisur nga raporte të organizatave kombëtare në fushën e sigurisë. Rekomandimet e tyre, ndikimi i krimit kibernetik në jetën socale, politike, ekonomike. Punime të deritanishme në zhvillimin e sistemeve të detektimit të ndërhyrjeve. Klasifikimi i tyre, avantazhet dhe problematikat për secilën teknikë. Motivimin, Qëllimin dhe objektivat që synon të realizojë punimi, si dhe Kontributi jonë në këtë fushë.

KAPITULLI 2 – KARAKTERISTIKAT KRYESORE TË SISTEMEVE TË DETEKTIMIT TË NDËRHYRJEVE

Kapitulli 2 trajton dy IDS më tipike për secilën teknikë Snort dhe Bro IDS, përkatësisht implementojnë teknikën e detektimit të ndërfutjeve (misuse detection) dhe detektimit të anomalive (anomaly detection). Arkitekturat e tyre, mënyrat e operimit, avantazhet dhe problematikat për secilën.

KAPITULLI 3 – ANALIZA DHE PROJEKTIMI I Ad-IDS

Në këtë kapitull, përshkruhen dy fazat e para të ciklit të jetës të krijimit të një softueri: analiza dhe projektimi. Në fazën e analizës, paraqiten disa konsiderata për sistemin e propozuar, përshkruhet algoritmi i tij; ndërsa në fazën e projektimit paraqitet struktura e sistemit të propozuar Ad-IDS, e cila përbëhet nga tre module: moduli kryesor, moduli i testeve dhe moduli i kundër-masave.

KAPITULLI 4 – IMPLEMENTIMI DHE TESTIMI I Ad-IDS

Në këtë kapitull, paraqiten dy fazat e tjera të ciklit të jetës; implementimi dhe testimi. Në fazën e implementimit paraqiten në mënyrë të detajuar thirrjet kryesore të sistemit si dhe funksionimi i tij në tërësi. Në këtë fazë përcaktohet dhe modeli matematikor ekzistues, si dhe ai i propozuar. Në fazën e testimit, kryhen një sërë testesh duke u nisur nga disa exploite të sulmit Mohimi i Shërbimit, “Mbirrjedhja” e buferit, Symlink, ndryshimi i “sjelljes” (anomaly).

KAPITULLI 5 – KRAHASIMI I SISTEMIT Ad-IDS ME SISTEME TË TJERË DETEKTIMI

Në këtë kapitull, paraqitet vlerësimi krahasues eksperimental i Ad-IDS me sistemet e tjerë Snort dhe Bro IDS. Ky krahasim është bazuar në aftësinë detektuese, performanca e sistemit, numri i alarmeve False Positive të gjeneruar.

PERFUNDIME DHE OBJEKTIVA TË SË ARDHMES

Së fundmi jepen disa përfundime mbi punën e realizuar dhe të paraqitur në këtë tezë, duke nxjerrë në pah avantazhet dhe problematikat e hasura, të cilat vihen në fokusin e punës në të ardhmen.

KAPITULLI 2

KARAKTERISTIKAT KRYESORE TË SISTEMEVE TË DETEKTIMIT TË NDËRHYRJEVE

2.1 Snort IDS

Snort [41] është një program "open-source", që përdoret si sistem dedektimi të ndërhyrjeve (Intrusion Dedection System). Snort është zhvilluar nga Martin Roesch, themelues i Sourcefire Inc, në Maryland. Snort i ka fillesat në vitet 1998-1999, por në këtë kohë ai ishte një program, i cili funksiononte shumë thjeshtë dhe nuk kishte një eficensë si një sistem dedektimi të ndërhyrjeve. Pavarsisht problemeve dhe vështërsive të fillimit, Snort u zhvillua si një program për të kryer funksionin e një sistemi dedektimi të ndërhyrjeve. Në vitin 1999 ishte i vlefshëm versioni 0.98, ndërsa versioni 1.0 u hodh në treg në 4/28/99. Por, versioni i parë i qëndrueshëm me karakteristikat e një IDS u lançua si Snort versioni 2.0 në 4/6/2003. Ky program ka patur një zhvillim të madh dhe të shpejtë dhe versionet më të fundit që janë implementuar janë versioni 2.4.5. dhe së fundi 2.9.

2.1.1 Teknika e detektimit

Teknika e detektimit të cilën përdor Snort është teknika e detektimit të ndërfutjeve (misuse detection). Kjo do të thotë se ai mbështetet në një bashkësi rregullash (rule set), e cila është përkufizuar më parë dhe NUK mund të njohë apo të detektojë sulme të reja të pakonfiguruara. Fatkeqësisht, Snort nuk ka aftësi të detektojë sulme apo ndërhyrje të reja.

2.1.2 Mënyrat e operimit

Mënyrat e operimit të Snort jepen si më poshtë:

- **"Thithësi" i paketave (Packet Sniffer Mode);** thjesht lexon paketat nga rrjeti (Interneti) dhe më pas i shfaq në formën e një "stream-i" të vazhdueshëm. Pra, në këtë mënyrë thjesht "printon" të gjitha paketat që merr nga libraria libpcap në një dritare. Kjo, ka qenë trajta fillestare e krijimit dhe operimit të Snort.

  ./snort- v: printon header TCP/IP në ekran.

./snort-vd: printon të dhënat e aplikacionit.

./snort-vde: printon përmbajtjen e shtresës së dalalinkut

- **"Rregjistruesi" i paketave (Packet Logger Mode);** rregjistron paketat në disk. Pra, "logon" gjithë trafikun tek një file të dhënë.

  .snort-dev-1./log: ruan paketat në direktoritë e specifikuara.

  .snort-1./log-b: "Binary logs" Dosjet që kanë përmbajtje binare

- **Sistem i Detektimit të Ndërhyrjeve (IDS Mode);** gjëndja më normale e funksionimit të Snort; funksioni i krijimit të tij (i përcaktuar në '99 me krijimin e versionit 1.0). Lejon Snort të analizojë proceset, duke u mbështetur në bashkësinë e rregullave (rule set).

- **Sistemi i Parandalimit të Ndërhyrjeve (IPS Mode- Intrusion Prevention System);** merr paketat nga iptables dhe jo nga libpcap. Më pas këto paketa i bën dropp/pass në varësi të tipeve të rregullave specifikë (drop, reject, sdrop etj). Kjo trajtë e funksionimit të Snort është ende në fazë eksperimentale dhe nuk mund të shprehemi se Snort është një IPS i mirëfilltë. (Sistemet IPS paraqesin një fushë të kërkimit shkencor në zhvillim e sipër).

2.1.3 Arkitektura e Snort

Figura e mëposhtme, paraqet arkitekturën e Snort.

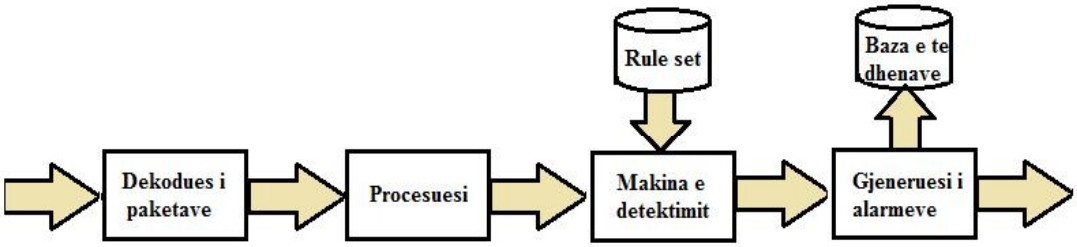

Figura 2.1 Arkitektura e Snort [41]

Komponentet e Snort janë :

1- Dekoduesi i paketave (Packet Decoder ose Packet Sniffer)

2- Procesuesi (Preprocesser)

3- Makina e detektimit (Detection engine)

4- Gjeneruesi i alarmeve (Output engine ose Alert Generation)

5- Baza e të dhënave

6- "Rule set" (bashkësia e rregullave)

Në mënyrë më të detajuar këto komponente jepen si më poshtë:

1. Dekoduesi i paketave (Packet Decoder ose Packet Sniffer) është një softuer (por mund të jetë edhe paisje harduer), e cila përdoret për të kapur paketat në rrjet. Për të realizuar këtë funksion, Snort përdor librarinë libpcap. "Kapësi " i paketave (sniffer-i) përdoret për:

- Analizën e trafikut të rrjetit;
- Analizën e performancës dhe ndërtimin e kampioneve;
- Përgjimi për tekste fjalë kyçe apo të dhëna të tjera "sensitive".

Ndërfaqet e rrjetit ku mund të përdoret, mund të jenë Ethernet, SLIP, PPP etj.

Në fillimet e Snort, kjo komponente e tij luante rol kryesor duke qenë se funksioni kryesor i tij ishte pikërisht "kapja" dhe analizimi i trafikut të rrjetit.

2. Procesuesi (Preprocesser). Procesorët janë shumë të rëndësishëm për një IDS, pasi përgatisin paketat që do të analizohen më pas nga motori dedektimit. Funksioni i këtij komponenti është modifikimi ose rregullimi i të dhënave të paketave përpara se ato të kalojnë në motorin e dedektimit. Snort përmban dy tipe procesuesish:

- Kontrollin e gjendjes;

- Ri-asemblimin e stream-ës.

Procesuesi i kontrollit të gjendjes kontrollon paketat për protokollet që ato përdorin dhe funksionet e tyre (psh. RPC, HTTP apo portscanner). Ndërsa procesuesi i riasemblimit të stream-ës, merret me përcaktimin e paketave të cilat i përkasin të njëjtës stream. Ky procesues është shumë i rëndësishëm sepse një nga sulmet më të njohura sot dhe më të aplikuara është ai i Fragmentimit të paketave. Fragmentimi i paketave ndosh kur një sasi e madhe të dhënash transferohet në një host, atëherë paketa fragmentohet në njësi më të vogla, në përputhje me formatin e ndërfaqes së rrjetit. Më pas destinacionit i duhet të bëjë riasemblimin e këtyre njësive në paketën origjinale. Sulmi konsiston në përdorimin e fragementeve për të fshehur etiketat në njësi më të vogla, në mënyrë që të mashtrojnë IDS me lehtësi. Megjithatë Snort është një IDS, e cila e realizon shumë mirë detektimin e këtij sulmi, pikërisht në sajë të funksionit të këtij procesuesi.

3. **Makina e detektimit** (Detection engine), është pjesa më kryesore në IDS dhe njëkohësisht pjesa më kritike e Snort në lidhje me parametrin kohë, i cili varet nga:

- Numri i rregullave;
- Arkitektura e makinës mbi të cilën po ekzekutohet;
- Lidhja me rrjetin etj.

Ai merr të dhënat nga procesori dhe i kontrollon mbi bazën e një bashkësie rregullash (rule set). Nëse rregullat përputhen me të dhënat në paketë atëherë ato gjenerojnë një alarm. Këto rregulla grupohen sipas kategorive: Trojan, worm, viruse, mbirrjedhja e buferit, aksesi në aplikacione të ndryshme etj.

4. **Gjeneruesi i Alarmeve** (Output engine ose Alert Generation) Nëse të dhënat që vijnë nga procesori përputhen me një nga rregullat në makinën e dedektimit atëherë gjenerohet një alarm. Këto alarme mund të dërgohen në një file nëpërmjet një lidhjeje me Internetin ose mesazheve SNMP traps. Gjithashtu, alarmet mund të ruhen në një database të tipit MySQL dhe Postgres (baza të dhënash falas). Alarmet në bazën e të dhënave dërgohen nëpërmjet protokolleve SNMP . Një formë tjetër për të dërguar alarmet është e-mail, për të lajmëruar një administrator sistemi në kohë reale. Në këtë mënyrë shmanget nevoja për të monitoruar gjatë gjithë kohës rrjetin.

5. **Baza e të dhënave**; raportet e gjeneruara nga moduli i Gjeneruesit të Alarmeve, ruhen në një bazë të dhënash. Duke qenë se Snort është "Falas" (dhe mund të shkarkohet nga www.snort.org), atij i atashohen baza të dhënash gjithashtu "Falas" si MySQL apo Postgres.

6. **"Rule set"**; bashkësi rregullash, në thelb është një bazë të dhënash në të cilën ruhen modelet e ndërhyrjeve të njohura. Aktiviteti i ndërhyrësve shoqërohet me "gjurmë" që në gjuhën e Snort njihen si modele të sulmit "attack patterns". Rregullat e Snort ngrihen mbi bazën e këtyre "patterns".

2.1.4 Sintaksa e një rregulli

Në inicializim, Snort lexon të gjitha rregullat, i analizon ato dhe i ruan në strukturatë dhënash të veçanta në memorjen kryesore. Të gjitha rregullat e Snort duhen të shkruhen në një rresht të vetëm. Me anën e karakterit '\' në fund të rreshtit, kalohet në një rresht të ri.

Sintaksa e rregullave të Snort është e mirë-organizuar. Çdo rregull përbëhet nga 2 pjesë:

```
alert tcp any any -> 192.168.1.0/24 111 \
  (content:"|00 01 86 a5|"; msg:"mountd access";)
```

Figura 2.2. Shembull i një Snort rule

1. **Header i rregullit** (rule header)- është një përkufizim statik që duhet të jetë i pranishëm në çdo rregull. Headeri përcakton ku mund të vijnë paketat e kapura, çfarë protokolli ka kjo paketë. Një header përbëhet nga fushat e mëposhtme:

```
Rule_action Protocol Source_ip Source_port \
Direction Destination_ip Destination_port
```

- **Rule-action** - veprimi që duhet të bëhet nqs rregulli ndeshet (në rastin e Fig.2.2 është alert- gjenero një alarm)
- **Protocol**- Protokollet që suportohen janë:ICMP,TCP,IP,UDP. (në rastin e fig, është tcp)

- **Source-IP**-IP burim nga vjen paketa (në rastin e fig. është any, që do të thotë nga çdo nyje burim)
- **Source port**- portën burim nga vjen paketa. (në rastin e fig, any)
- **Direction**:''=>'' ose '' <>" operatori shigjetë i tregon Snort që informacioni i burimit është në anën e majtë, ndërsa informacioni i destinacionit është në anën e djathtë. Operatori"<>", i tregon Snort që duhet të ekzaminojë të dy drejtimet.
- **Destination-IP**- adresa IP e destinacionit (në rastin e fig. është 192.168.1.0/24)
- **Destination-port**- porten ku duhet te shkoje paketa.(në rastin e fig. është 111)

2. Opsionet e rregullit (Rule Options)- është një përkufizim varibël dhe jo gjithmonë i pranishëm. Snort, ka më shumë se 50 opsione të vlefshme për një rregull, të cilat shërbejnë për funksione të ndryshme.

```
( <rule_option_name>: <rule_option>; ...)
( msg:"Sasser worm found!"; content:"|0f0f12|"; )
```

Opsionet e rregullit mund t'ju referohen fushave të protokolleve ose protokolleve të veçanta të rrjetit, mesazheve që janë drejtuar administratorit ose përmbajtjeve që mund të përmbajë paketa për të trigeruar një veprim. Në rastin e mësipërm kemi:

- **rule_option_name** - emri i opsionit të rregullit (në rastin tonë është msg.)
- **rule option**- në rastin tonë është gjetja e një worm-i me emrin Sasser (Sasser worm found)
- **content**- opsioni i përmbajtjes

Përcakton se çfarë përmbajtje duhet të ketë paketa''keqdashëse'' për të trigeruar veprimin që është përcaktuar më parë në header-in e rregullit. Opsioni i përmbajtjes automatikisht shikon për hyrje ASCII. Nqs kërkimi pas modeleve të sulmit, gjithashtu konsiston në permbajtje binare, kjo përmbajtje duhet të shkruhet mes simboleve (''|'') në shkrim hekzadecimal në mënyre që të dallohet nga përmbajtja ASCII.

Psh content:"|0102|/etc/passwd";

Ky rresht kap paketa që përmbajnë bytet 0x0.1,0x0.2 të ndjekura nga stringu "/etc/passwd''. Opsioni i përmbajtjes mund të marrë këto vlera:

Opsioni	Kuptimi
depth	Përcakton sa byte të paketës së kapur, duhet të "skanojë"
offset	Përcakton se nga cila pjesë e paketës duhet të nisë "skanimin" e paketës
nocase	Përcakton "skanim" të pavarur nëse gërmat janë shypi apo jo (no case sensitive)
session	"printable" or "all": "printable" ruan vetëm tekstin, ndërsa "all" përveç tekstit ruan edhe të dhënat binare në log file
uricontent	Kërkon një stringë të dhënë vetëm në argumentin URL (për këtë arsye paketa duhet të jetë paketë HTTP)

Tabela 2.1 Opsionet e rregullit në Snort

2.1.5 Konfigurimi i rregullave

Konfigurimi i Snort IDS, i rregullave, si dhe të veprimeve të tij realizohet në një file të tipit .conf. (snort.conf) Në këtë file bëhen konfigurimet e mëposhtme:

- **përcaktimi dhe përdorimi i variablave**; Do jetë një humbje e madhe në kohë për të rregulluar adresat e paketave IP nqs diçka ndryshon në rrjet. Snort e suporton përdorimin e variablave. Sintaksa për variabla të përcaktuara është:

```
var <variable_name> <variable_value>
```

- **përfshirjet**; Snort mund të përfshije file që përmbajnë pjesë të konfigurimit të tij. Sintaksa është:

```
include: <include_path/name>
```

- **konfigurimi i përgjithshëm**; Sintaksa për të shtuar një opsion konfigurimi në filen e konfigurimit të Snort është:

```
config <directive> [: value]
```

Tabela e mëpshtme përshkruan disa nga opsionet të konfigurimit të Snort.

Kuptohet që duke qenë se rregullat e Snort janë të konfigurueshme, ato janë fleksibël dhe na lejojnë shtimin e rregullave të reja. Në fakt, sa më e madhe makina e detektimi (numër të madh rregullash), aq më e madhe është edhe aftësia detektuese e Snort.

Emri	Shpjegimi
alertfile	Përcakton se ku duhet të shkruhen alarmet e detektuara
checksum_mode	Përcakton se checksum të cilave paketave duhet të kontrollohen. Opsionet që mund të përdoren janë: all, none, noicmp, noip, notcp dhenoudp
chroot	I përcaktojmë Snort-it direktorinë root
daemon	E ekzekuton Snortin si një proces i cili duhet të testohet
decode_arp	I thotë Snort-it të dekodojë paketa ARP
decode_data_link	I thotë Snort-it të dekodojë shtresën data-link
interface	Përcakton se cilën ndërfaqe NIC, duhet të marre Snort në konsideratë për të kapur paketa. (në rastin kur kemi më shumë se një NIC)
logdir	Përcakton direktorinë log
stateful	Paketat të cilat kanë një TTL më të ulët se TTL e përcaktuar, i bëhen “drop”

Tabela 2.2 Konfigurimi i rregullave në Snort

2.2 Bro IDS

Bro është një sistem i detektimit të ndëryrjeve mjaft i njohur në llojin e vet (detektimi i anomalive). Bro ka një operim eficient në rrjetat me trafik të lartë dhe përballimin e sulmeve ndaj rrjetit, duke përdorur një qasje "të bazuar në ngjarje" (event driven).

Bro e ka origjinën si një sistem kërkimi shkencor. U projektua dhe zhvillua nga Vern Paxcon në qendrën kërkimore në Berkeley. E filloi projektin në 1995 dhe që atëherë Bro është në zhvillim të vazhdueshëm.

2.2.1 Teknika e detektimit

Bro është një IDS e cila bazohet në teknikën e detektimit të bazuar në sjelljen ose ngjarje (behavior or event), pra teknikën e detektimit të anomalive (anomaly detection) [23].

Aktiviteti i marrë në shqyrtim, është abstraguar në ngjarje të cilat kalojnë dhe trajtohen nga bërthama (pjesa qendrore) e Bro. Në këtë pjesën qëndrore (kernel) administratori përcakton kufizimet specifike të mjedisit ku po ekzekutohet, në mënyrë që të detektohet sjellja normale nga sjellja me anomali.

Si shumica e NIDS Bro bazohet në një paketë monitorimi e cila dëgjon për paketa. Në mënyrë më të detajuar arkitektura e Bro jepet në paragrafet më poshtë.

2.2.2 Mënyra e operimit

Bro koniston në tre pjesë kryesore:

- Kapja e paketave;
- Mekanizmi i ngjarjeve ose sjelljes (event);
- Shtresa e cila definon kufizimet specifike të mjedisit bazuar në scriptin që jep përdoruesi (policy layer) .

Mënyra e operimit të Bro konsiston në përcaktimin e ngjarjeve normale të sitemit dhe ndërveprimin me administratorin i cili jep miratimin përfundimtar mbi definimin final të një sjellje normale (e cila bëhet në shtresën e politikave- policy layer).

2.2.3 Arkitektura e Bro

Qëllimet e projektimit e paraqitura më lart, çojnë në arkitekturën e shtesave të treguar në Figurën 2.3. Detyra e shtresës më të ulët,eshte *kapja e paketave*. Ajo i dërgon ato në bërthamën e Bro-së, makina e ngjarjeve "event engine" , e cila kryen përpunimin e tyre. Makina gjeneron një stream te eventeve të cilat janë kaluar për në shtresën e Interpretuesit (Policy layer). Kjo shtresë vlerëson ngjarjet duke u nisur nga përputhshmëria me "historikun e sjelljes".

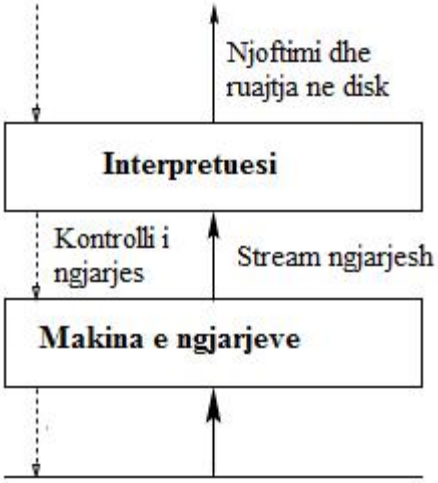

Figura 2.3 Arkitektura e Bro IDS

Në shtresën e Interpretuesit (policy layer), përdoruesi mund të përcaktojë një manaxhues të ngjarjeve (event handler) përkatës i cili specifikon veprimet që duhet të kryhet.Për funksionimin e tij, manaxhuesi i ngjarjeve nuk ka për të inspektuar paketat të përfshirë në vetvete, por punon në përmbajtjen e informacionit të tyre në nivel të lartë.

2.2.4 Kapësi i paketave

Shtresa më e ulët e Bro është Kapësi i paketave (packet capture) që kalon paketat aktuale të rrjetit te Makina e ngjarjeve (event engine). Për këtë detyrë, Bro përdor librarinë *libpcap* [Pca] e cila siguron një ndërfaqe të pavarur të platformës në teknologjitë e ndryshme të rrjetit. Libpcap implementon shprehje filtrimi të vogla por të fuqishme që cilat mund të përdoren për të reduktuar sasinë e paketave të analizuara. Për

shembull, në qoftë se ne nuk duam të analizuar trafikun në kanalet dhënave FTP, ne thjesht mund të injorojë të gjitha paketave në portën TCP 20. Duke implementuar këtë ide, Bro ndërton një shprehje filtrimi dinamike gjatë ekzekutimit.Filtri përzgjedh vetëm ato paketa për analizim të mëtejshëm të cilat janë vërtet të nevojshme për qëllimet aktuale. Meqënëse libpcap ndërfaq direkt në nivelin kernel paketat filtrohen në shumë sisteme operative (si FreeBSD dhe Linux).

2.2.5 Makina e ngjarjeve

Bërthama e Bro është motorri i tij i ngjarjeve-event engine. Kjo shtresë kryen të gjitha punët e nivelit të ulët për të analizuar paketat të rrjetit. Event engine merr paketat e para IP nga packet capture , I klasifikon sipas lidhjes, rimbledh të dhënat TCP streams, dhe deshifron protokollet e shtresës se aplikimit. Sa herë që ajo ballafaqohet me diçka potencialisht të përshtatshme me policy layer , ajo gjeneron një event. Meqënëse përballet me sasi të mëdha të trafikut, event engine duhet të jetë shumë efikas. Njëkohësisht, për t'i bërë ballë sulmeve kundër vetë Bro, ajo duhet të jetë aq robust sa të jetë e mundur..

Event engine përbëhet nga disa analizues. Secili prej tyre është përgjegjës për një detyrë të mirëpërcaktuar për shembull deshifrimin e një protokolli të veçantë, kryerjen njohjes se nënshkrimit, ose identifikimin e backdoors. Secili mund të ngrejë një grup të eventeve specifike të analizuesve. Zakonisht, një analizues është i shoqëruar nga një script i parazgjedhur i cili zbaton një politikë të përgjithshme e rregullueshme në mjedisin lokal.

2.2.6 Interpretuesi

Interpretuesi ose Shtresa e politikave (Policy layer), interpreton “sjelljen”, apo historikun e saj duke u nisur nga Makina e Ngjarjeve (event engine). Është pikërisht ky modul përbërës i arkitekturës Bro IDS, i cili gjeneron raportet duke u nisur nga interpretimi që i bëhet një ngjarje (event). Për të realizuar këtë funksion, ai përdor manaxhuesin e ngjarjeve, i cili bën një procesim në nivel të lartë.

2.3 Disa përfundime

Nga analiza e detajuar teorike dhe e aplikuar e Snort IDS dhe Bro IDS, kemi arritur në disa përfundime si më poshtë:

- Snort është një mënyrë relative për të detektuar ndërhyrje në system. Administratoret janë të vetëdijshëm për aktivitetin e detektueshëm nga Snort (duke u nisur nga konfigurimi i rule set).
- Snort nuk suporton sulme anormale por detekton vetëm sulme të njohura më parë.
- Snort mund të implementohet në platforma të ndryshme softuer dhe harduer.
- Bro IDS është një IDS e cila implementon teknikën e detektimit të anomalive dhe si e tillë është e aftë të detektojë edhe sulme të reja.
- Është e thjeshtë për tu implementuar dhe e lehtë për tu konfiguruar.
- Limitimi i Bro konsiston në faktin se është e varur nga sistemi i operimit ku implementohet (pasi është krijuar për sisteme operimi UNIX ose Linux).

KAPITULLI 3
ANALIZA DHE PROJEKTIMI I SISTEMIT Ad-IDS

3.1 Analiza e sistemit të propozuar

Ideja themelore e këtij punimi është të projektojmë një sistem detektimi të ndërhyrjeve Ad-IDS. Për të krijuar strukturën e Ad-IDS jemi bazuar më së shumti në Snort IDS (duke qenë se është "open source"). Gjithashtu set-it të testeve ekzistuese të Snort (të cilat janë të tipit "të detektimit të ndërfutjes"), do t'i shtohen edhe testet e Bro IDS (të cilat janë teste të tipit "të detektimit të anomalive"). Përveç set-it të testeve të Snort dhe Bro, ne kemi konceptuar edhe disa teste shtesë (dhe në vijim do i referohemi me këtë emërtim "teste shtesë"), të cilat ne mendojmë se do të përmirësojnë performancën e Ad-IDS, duke parashikuar disa raste të veçanta të historikut të "sjelljes" së një procesi. Të gjitha testet e sipër-përmendura do të jenë pjesë e Modulit të Testeve [3.5]. Të gjitha proceset e sistemit të operimit të testuar, do të vendosen "përballë" Modulit të Testeve.

Çdo test i kaluar do të vlerësohet nga një mekanizëm i përcaktimit të pikëve [kap. 4]. Në fund të të gjitha testeve, çdo proces ka një shumë pikësh të caktuara. Duke numëruar këto pikë, Ad-IDS-ja mund të përpiqet të përcaktojë nëse një proces është i parrezikshëm, paraqet një aktivitet të dyshimtë apo një aktivitet të rrezikshëm. Në çdo rast Ad-IDS-ja i raporton veprimet e kryera tek një bazë të dhënash. Baza e të dhënave është pjesë e modulit kryesor [paragrafi 3.4].

Gjithashtu, duke ju bazuar nivelit të shumatores së pikëve, Ad-IDS duhet të jetë e aftë të realizojë disa kundër-veprime automatike. Këto veprime janë pjesë e Modulit të

Kundër-masave [paragrafi 3.6]. Kuptohet se përveç veprimeve të realizuara në mënyrë automtaike nga Ad-IDS, administratori i Ad-IDS luan një rol shumë të rëndësishëm në gjykimin dhe veprimin e proceseve të Rrezikshme, e sidomos atyre që paraqesin aktivitet të Dyshimtë.

3.2 Metodologjia e përdorur

Metodologjia e përdorur bazohet në sa më poshtë:

1. **Klasifikimi i Alarmeve;** duke u nisur nga IDS-të ekzistuese alarmet klasifikohen si më poshtë:

- **FALSE POSITIVE** (Type 1 errror); kur një test gabimisht raporton për ndërhyrje, ndërkohë që kemi të bëjmë me aktivitet legjitim;
- **TRUE POSITIVE**; kur vërtet ka ndodhur një ndërhyrje;
- **FALSE NEGATIVE**; kur nuk ka patur asnjë alarm të gjeneruar nga IDS-ja, pavarësisht se ka patur ndërhyrje;
- **TRUE NEGATIVE**; kur nuk ka patur ndërhyrje dhe nuk është gjeneruar alarm.

Pra, siç mund të vihet re nga klasifikimi, alarmet "FALSE" janë ata më problematikët, ndërsa alarmet "TRUE" janë alarme të cilët duhet të gjenerohen.

Metodologjia jonë bazohet në klasifikimin e alarmeve (proceseve), si më poshtë:

- **I Parrezikshëm (kategoria 1);** Në këtë rast kemi të bëjmë me alarmin True Negative;
- **I Dyshimtë (kategoria 2);** Në këtë rast kemi të bëjmë me alarmin False Positive;
- **I Rrezikshëm (kategoria 3);** Në këtë kategori, janë futur alarmet True Positive dhe False Negative.

2. **Në rastin kur procesi konsiderohet kategoria (2) ose (3), krijohet një model ("pattern") i tij dhe i shtohet Bazës së të dhënave** (Rritet Baza e të dhënave); Duke u nisur nga set-i testeve mbi "sjelljen" e procesit, Ad-IDS është e aftë të detektojë ndërhyrje të reja, të panjohura më parë. Si ndërhyrje konsiderohen jo vetëm proceset të Rrezikshme (3), por edhe ato të Dyshimta (2). Integrimi i

testeve të tipit të detektimit të ndërfutjeve me tipin e detektimit të anomalive, në një set testesh, na sjell këto avantazhe:

- Zgjidh problematikën e teknikës së detektimit të ndërfutjeve, në lidhje me mos detektimin e ndërhyrjeve të panjohura. Pra, Ad-IDS me këtë set testesh të integruar, është e aftë të detektojë jo vetëm sulme të njohura, por edhe sulme të reja;
- Krijimi i një modeli mbi një sulm të ri dhe shtimi i tij në bazën e të dhënave, na rrit Bazën e të dhënave të sulmeve të njohura.

3. **Në rastin kur procesi kategorizohet (1), atë e konsiderojmë si True Negative (Ulet numri i alarmeve False Positive);**
4. **Nocioni i pikëve**; Për të gjithë testet e kaluara procesi merr një vlerësim që ne e quajmë ”pikë” [kap.4]. Në lidhje me pikët që fiton një proces duhet të përcaktojmë dy gjëra:

- Sa pikë do të fitojë procesi për teste të veçanta;
- Çfarë niveli i pikëve do të konsiderohet një ndërhyrje.

3.3 Projektimi i sistemit Ad-IDS

Faza e projektimit në “ciklin e jetës” së krijimit të një softueri, është mjaft e rëndësishme sepse në këtë fazë, krijohet struktura kryesore e softuerit. Gjithsesi, para se të projektojmë modulet kryesore të Ad-IDS, duhet të kemi parasysh disa konsiderata për të, si dhe algoritmin mbi të cilin do të bazohemi për krijimin e tij.

3.3.1 Disa konsiderata mbi sistemin e propozuar

Disa konsiderata për projektimin e IDS dhe vetitë thelbësore të tij janë:

- IDS duhet të përmbajë një modul kryesor; ky modul duhet të ketë funksion inicializues dhe konfigurues për IDS-në;
- IDS duhet të përmbajë një set testesh; në bazë të tyre do të bëhet kategorizimi i proceseve si të rrezikshëm, i dyshimtë, i parrezikshëm;
- Testet duhet të jenë fleksibël; testet duhet të jenë të manaxhueshme dhe të konfigurueshme nga administratori i IDS;

- IDS duhet të përmbajë një set kundërmasash; janë veprime të cilat do të kryhen pasi procesi kalon nëpër një test. Të gjitha këto veprime do të pasqyrohen në një dokument të tipit *log file;*
- IDS duhet të gëzojë vetinë e përshkallëzimit (skalabilitetin); ky sistem duhet të lejojë shtimin e testeve të reja ose heqjen e testeve ekzistuese (të cilat mund të mendohen të pavlefshme);
- IDS duhet të jetë fleksibël; ky sistem duhet të lejojë manaxhimin e kundërmasava nga një administrator i IDS; kundërmasa të cilat mund të gjenerohen automatikisht nga sistemi ose të kryehen manualisht nga administratori;
- IDS duhet të përmbajë një bazë të dhënash; kjo bazë të dhënash duhet të mbajë raportet në lidhje me testet e kryera.

3.3.2 Algoritmi i Ad-IDS

Bllokskema logjike e IDS-së së propozuar jepet si më poshtë. Me ngjyrë të kuqe jepen modulet kryesore të IDS-së; moduli i Testeve; moduli Kryesor (i cili paraqitet nga baza e të dhënave); moduli i Kundër-Masave. Çdo proces kalon të gjitha testet e propozuara. Nqs shuma e pikëve e tejkalon limitin e pikëve të përcaktuar i cili identifikon një ndërhyrje, atëhere duhet të gjenerohet një alert, i cili do të aktivizojë kundër-masën përkatëse, qoftë kjo automatike nga vetë sistemi, ose manuale të ndërmarrë nga administratori i IDS-së. Në të dyja rastet, kundër-masa e ndërmarrë duhet të raportohet në bazën e të dhënave për procesin përkatës, duke gjeneruar një raport (log file).

Në rastin kur procesi paraqet një nivel të ulët pikësh, larg nivelit të përcaktuar si ndërhyrje, atëhere në këtë rast nuk kemi të bëjmë me ndërhyrje, megjithatë procesi duhet të raportohet në bazën e të dhënave në kategorinë "i Parrezikshëm".

Në rastin kur pikët e mbledhura nga testet, janë në një diapazon afër vlerave të përcaktuar si limit për të identifikuar një ndërhyrje [kap.4], në këtë rast kemi të bëjmë me një aktivitet "të Dyshimtë". Në këtë rast, ky proces raportohet në bazën e të dhënave dhe për të gjenerohet një log file (një raport). Administratori i IDS-së, pasi analizon procesin, vendos mbi veprimin që do të ndërmarrë. Në rastin kur kemi të bëjmë me alarm të tipit False Positive, procesi konsiderohet "i PaRrezikshëm".

Në rastin kur procesi në fjalë, paraqet një ndërhyrje, atëhere përcaktohet një "patern" i tij dhe ai i shtohet modulit të Testeve, në mënyrë që në momentin kur ky proces t'i "nënshtrohet" sërisht testeve, ai të detektohet si ndërhyrje dhe të kategorizohet si proces "i Rrezikshëm". Në këtë mënyrë ulet numri i alarmeve False Positive të gjeneruar dhe në të njëjtën kohë pasurohet numri i testeve të aplikuara mbi proceset, duke rritur në këtë mënyrë performancën e IDS-së së propozuar.

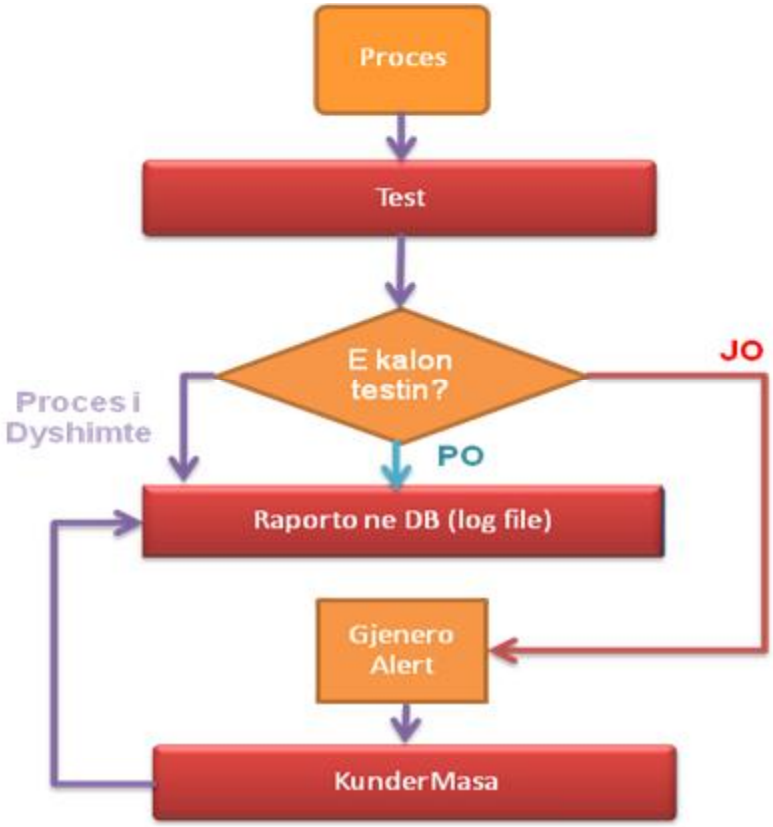

Figura 3.1 Algoritmi Ad-IDS

3.3.3 Projektimi i sistemit Ad-IDS

Sistemi Ad-IDS përbëhet nga tre module kryesore:

1. Moduli kryesor; Përgjegjës për inicializimin e sistemit, si dhe të konfigurimit të tij. Pjese e këtij moduli është menduar edhe baza e të dhënave, e cila manaxhon raportet e gjeneruara, si pasojë e testeve të kryera.

2. Moduli i testeve; Ky modul paraqet një set testesh të cilat aplikohen mbi proceset që do të testohen. Kalimi apo jo i testit përcaktohet nga një nivel i pikëve. Përcaktimi i nivelit të pikëve duhet të jetë lehtësisht i konfigurueshëm.
3. Moduli i kundërmasave; Ky modul paraqet disa veprime të cilat do të kryhen në rastin kur procesi konsiderohet "i rrezikshëm", pra atëhere ku është kaluar niveli i caktuar i pikëve.

Sistemi Ad-IDS ekzekutohet me cikle (në formë cirkulare), duke performuar të gjithë testet mbi të gjithë proceset, çdo sekondë. Pasi ekzekutohen të gjithë testet dhe gjenerohen raportet e tyre, sistemi thërret të gjithë veprimet e paracaktuara për të krahasuar rezultatet. Pas çdo testi, përdoruesi (administratori i sistemit), mund të konfigurojë pikët e përftuara si dhe nivelin e nevojshëm të tyre. Kodi burim i sistemit Ad-IDS të propozuar, jepet në Shtojcën A.

3.4 Moduli kryesor

Përgjegjës për inicializmin e sistemit. Pjesë e këtij moduli është menduar edhe baza e të dhënave, e cila manaxhon raportet e gjeneruara, si pasojë e testeve të kryera.

3.4.1 Baza e të dhënave

Baza e të dhënave përmban të gjithë informacionin e mbledhur rreth sjelljes së procesit, në trajtë e një raporti. Ky informacion konsiston në: kohën kur është zhvilluar testi (data dhe ora), emri dhe PID (Process ID), Test ID, pikë-vlerat (niveli i pikë-vlerave të "kapura" nga procesi pas përfundimit të testit), si dhe parametrin TTL (Time To Live).

Është e kuptueshme që informacioni rreth aktivitetit të proceseve të dyshimta, është i rëndësishëm brenda një kohe të limituar. Psh. Informacioni "procesi ishte duke ekzekutuar një program SUID 4 javë më parë", nuk ka vlerë. Kjo është dhe arsyeja e vendosjes së parametrit TTL (Time To Live) në sistemin IDS. Ky parametër është i konfigurueshëm nga përdoruesi, pra mund të ndryshohet sipas kërkesave dhe qëllimeve të tij. Për qëllimet e testit tonë, ne e vendosim këtë parametër për 60 sekonda, sepse ne besojmë se tentativat tipike për ndërfutuje mund të përftohen për një diapazon kohe të tillë. Sistemi gjithashtu krijon një raport tekst (log file) të kompletuar, i cili përmban të gjithë informacionin rreth proceseve të testuar.

Koha kur u zhvillua testi	Emri i Procesit	PID	Test ID	Pikët	TTL (1)	Klasifikimi (2)
Data/ ora						
(1) TTL-Time To Live . Koha gjatë së cilës merret në konsideratë një proces. (2) Klasifikimi. Kemi tre kategori të klasifikimit të proceseve: I Rrezikshëm, I Dyshimtë, I Parrezikshëm						

Tabela 3.1 Formati i Bazës së të dhënave

3.4.2 Thirrjet e sistemit (syscalls)

Thirrjet e sistemit (syscalls), janë metoda të cilat thirren nga procese të ndryshëm për të realizuar funksione të ndryshme. Thirrjet e sistemit manaxhohen nga kerneli i sistemit të operimit. Thirrjet e sistemit janë shumë të rëndësishme në vlerësimin e sistemit Ad-IDS; kjo, sepse në bazë të tyre përcaktohet historiku i sjelljes së proceseve të ndryshme. Si pjesë e këtij historiku, është dhe implementimi i thirrjeve të sistemit të cilat realizojnë një funksion të caktuar. Në momentin që një proces i testuar nga Ad-IDS ka një "devijim" nga ky historik, atëhere në bazë të testeve të mbështetura në algoritmin e detektimit të anomalive, ky "devijim" detektohet si një ndërhyrje. Prandaj "kapja" e këtyre thirrjeve të sistemit dhe vlerësimi i tyre është shumë i rëndësishëm në Ad-IDS. Pjesa e "kapjes" së thirrjeve të sistemit, është pjesë e modulit kryesor. Historiku i implementimit të thirrjeve të sistemit, mbahet në bazën e të dhënave të modulit kryesor të Ad-IDS.

3.4.3 Manaxhimi i "fork( )s"

Një çështje e cila duhet te merret në konsideratë gjatë projektimit të IDS-së është manaxhimi i rasteve kur procesi implementon fork(). Në sistemet e operimit, fork() është një "sistem call", e cila implementohet në kernel. Kjo metodë krijon një proces kopje të vetvetes, ku procesi i krijuar konsiderohet proces fëmijë e atij që e krijoi. *Arsyeja e implementimit të fork() është për të mundësuar vetinë e "multitasking" për sistemin e operimit (procese të cilat ekzekutojnë programe, duhet të krijojnë procese "klone" për të ekzekutuar programe të tjera).* Procesi prind dhe fëmijë kanë të njëjtën hapësirë adresimi, por vende të ndryshme në memorie. Metoda fork(), nuk merr

argumenta, por ajo na kthen një integer ID e procesit. PID e procesit përftohet nga *metoda getpid().* Në sistemet e operimin, proceset krijojnë një pemë, ku *procesi init (me PID=1),* është në rrënjë të saj.

Mund të ndodhë që një proces "malicios" mund të bëjë fork() dhe në këtë mënyrë të gjithë proceset fëmijë të gjeneruar, të jenë pjesë e një aktiviteti të Dyshimtë. Për këtë arsye sistemi IDS, ka mundësi të mos detektojë këtë aktivitet. Për ta detektuar, i duhet të ekzaminojë të tërë grupin e proceseve (process tree).

Për të zgjidhur këtë çështje, ne propozojmë këtë mekanizëm:

- Kur një proces implementon një fork(), të gjitha raportet (të dhënat) e tij kopjohen tek fëmija e tij. Prandaj të dy proceset shfaqin të njëjtën histori.
- Kur një test raporton një veprim të dyshimtë të procesit, ky raport kopjohet edhe tek procesi prind. Megjithatë, pikët e raportit prind, është e modifikuar në varësi të numrit të proceseve fëmijë që ka procesi prind. Për shembull, nëse procesi A ka 10 fëmijë, dhe një nga ata paraqet një veprim të dyshimtë, raporti i procesit fëmijë është gjithashtu i kopjuar në Bazën e të dhënave të procesit A, por pikë – vlerat e procesit A (procesi prind) është pjesëtuar me 10. Në rast se procesi ka vetëm një fëmijë, procesi prind e mban të gjithë përgjegjsinë për veprimet e procesit fëmijë dhe përmban të gjitha pikë-vlerat e tij.
- Kur një proces thërret setuid() për të përcaktuar privilegjet e tij, lidhja e tij me procesin prind fshihet. Kjo bëhet për dy arsye: 1- në mënyrë që procesi prind të mos mbajë përgjegjësitë e procesit fëmijë me privilegje të reja (procesi fëmijë mund të performojë aktivitete të dyshimta); 2- në këtë rast gjenerimi i raporteve do të kryheshin vetëm për procesin fëmijë (i cili tashmë është një proces i pavarur, i cili mund të implementojë nga ana e tij fork() dhe të krijojë procese fëmijë nga ana e tij), duke mënjanuar gjenerimin e raporteve "të pamerituara" për procesin prind.

3.4.4 Veprimet e përsëritshme

Kur një proces është duke kryer disa veprime në mënyrë të përsëritshme, ne nuk duam që sistemi IDS të gjenerojë sasi të mëdha të raporteve. (sepse mund të ndodhë që një proces të kryejë disa veprime në mënyrë ciklike; psh. Të kalojë nga njëra gjendje në tjetra në mënyrë ciklike). Në një rast të tillë IDS-ja mund të jëtë shumë e prirur për të gjeneruar alarme të tipit "false positive". Alarmet e tipit "false positive", kategorizohen në kategorinë "Aktivitet i Dyshimtë"(në këtë rast aktiviteti legjitim i procesit mund të

konsiderohet si një sulm i mohimit të shërbimit “denial of service”). Për të zgjidhur këtë çështje (pra duke ulur në këtë mënyrë alarmet e tipit “false posistive”), ne propozojmë mekanizmin e mëposhtëm:

- Kur një proces me një PID të dhënë paraqet një aktivitet të dyshimtë dhe tashmë ka një raport që e përshkruan këtë lloj aktiviteti në Bazën e të dhënave, raporti i ri i gjeneruar për përshkrimin e një aktiviteti të dyshimtë, nuk i shtohet Bazës së të dhënave, por thjesht i mbivendoset ose azhornon raportin ekzistues për të njëjtën PID. Kjo do të thotë që parametri TTL e tij është vendosur në kohën e tashme + TTL ekzistuese, si dhe pikë – vlera e tij është modifikuar në varësi nga pikë – vlera korrente e testit (e cila mund të ishte ndryshuar nga administratori i IDS-së).

3.5 Moduli i Testeve

Ekzistojnë shumë tipe të ndryshme problemesh lidhur me sigurinë dhe ndërhyrjet kompjuterike. Për qëllimet tona të testimeve të IDS-së së propozuar, ne kemi zgjedhur tre sulme tipike: “mbirrjedhja” e buferit (buffer overflow) dhe sulmet ”symlink” (symlink attack). Këta sulme (exploit-et e tyre) janë marrë të gatshëm nga baza e të dhënave e PSSED (Packet Storm Security Exploit Database http://packetstormsecurity.org/assess/exploits/), si dhe nga baza e të dhënave http://www.exploit-db.com/.

Duke integruar karakteristikat e këtyre sulmeve, në veprime individuale të performuara nga proceset, përftojmë një tërësi testesh të cilët do të jenë në gjendje të kapin këto lloj ndërhyrjesh. Gjatë analizës së exploit-eve të sulmeve të shkarkuara nga interneti, vumë re se:

- shumica e tyre kanë një mungesë në dokumentacionin shoqërues dhe komenteve;
- janë të dizenjuara për një version të caktuar të sistemit të operimit (zakonisht në Linux);
- jo të gjithë exploit-et punojnë korrektësisht.

Exploit-et e përdorur jepen në tabelat e mëposhtme. Kodi burim, si dhe dokumentacioni shoqërues e exploit-eve të përdorur, mund të gjenden në Shtojcën B.

3.5.1 Përshkrimi i exploit-eve

Në tabelën e mëposhtme jepen në mënyrë të përmbledhur exploit-et e përdoura:

“Mbirrjedhja” e buferit exploit	
TerminatorX	overflow buferit idbuff[256] në funksionit load_tt_part()

Tabela 3.2 Exploiti i “Mbirrjedhja” e buferit

Symlink exploit	
xine	shkruan në /tmp/xine-bugreport pa kontrolluar per symlink.

Tabela 3.3 Exploiti i “symbolic link”

Mohimi i Shërbimit (DoS)	
forkmembomb	Procesi bën fork() në cikle

Tabela 3.4 Exploiti i Mohimit të Shërbimit

TerminatorX. Ky exploit është tipik i mbirrjedhjes së buferit Ky exploit i bën “overflow” buferit idbuff[256] në funksionit load_tt_part(). Ky exploit thërret një “root-shell” duke ekzekutuar programin terminatorx. Programi terminatorx është një sintetizator audio në kohë reale. Duke qenë se programi terminatorx ekzekutohet në një makinë lokale, ky exploit është i tipit “Local root exploit”.

Xine 0.9.23 Ky program përdoret për të "mbishkruar" file të tjera, në rastin kur ai ekzekuton xine-bugreport, i cili shkruan në /tmp/xine-bugreport pa kontrolluar për symlink.

Forkmembomb ; është një exploit i cili gjeneron procese fëmijë nëpërmjet implementimit të thirrjes fork(), me qëllim shfrytëzimin e memorjes së makinës.

3.5.2 "Mbirrjedhja" e buferit (buffer overflow)

Sulmet "buffer overflow" jane tipikë për programet e shkruara në gjuhën C. Ky sulm bazohet në mbingarkimin e një buferi (p.sh. vektor) në memorje, me një sasi e madhe të dhënash, të cilat shkaktojnë mbishkrimin e zonave të tjera të memorjes së programit dhe mund të bëjnë që programi të ekzekutojë kode të gabuara (me qëllim të keq) të përgatitura nga sulmuesi.

Ne vendosëm të implementojmë testet e mëposhtme për këtë lloj sulmi:

- Procesi thërret funksionin execve() me EIP në stivë ose hapësirën ku janë adresuar të dhënat. EIP (Extended Instruction Pointer) këtu vendoset adresa e instrukionit të rradhës që do të ekzekutohet.Thirrja execve() na kthen një pointer mbi emrin e filet që do të ekzekutohet;
- Procesi ekzekuton një "interactiv shell". Një "interactiv shell", është një shell me të cilin përdoruesi ndërvepron, duke shkruajtur komanda dhe duke marë përgjigje.
- Procesi ekzekuton një program SUID ose SGID. (SUID-Set User ID dhe SGID-Set Group ID) ndryshojnë privilegjet e procesit. Kur SUID është vendosur me privilegjet e administratorit të sistemit (owner privilages), proceset që janë në ekzekutim kanë akses të plote në burimet e sistemit.

3.5.3 Sulmet "symlink" (symbolic link)

Sulmet symlink bazohen në ndërveprimin e dy proceseve. Procesi i parë është i privilegjuar (ka përparësi) dhe shkruan mbi një file (ose ekzekuton një file) në një direktori të shkrueshme edhe nga përdorues të tjerë. Në rast se ai (procesi i parë) nuk dikton që file-i përbën një "symlink", procesi i dytë (i ekzekutuar nga një sulmues) mund të krijojë një "symlink" duke bërë që file-i të shënjojë (pointojë) tek një tjetër file, i cili mund të mbajë të dhëna sensitive (p.sh. /etc/passwd). Në këtë mënyrë file-i me të dhëna sensitive mund të mbishkruhet duke ndryshuar të dhënat që mbante ose mund të ekzekutohet madje edhe pse sulmuesi nuk ka privilegje për ta bërë atë.

Ne vendosëm të implementojmë testet e mëposhtme për këtë lloj sulmi:

- Procesi krijon një link në direktoritë e dyshimta (p.sh. /tmp);
- Procesi i ekzekutuar si "non-root" krijon një link në file-t "root";
- Procesi që është duke u ekzekutuar si "root", hap link-e të tjera përdoruesish për shkrim;
- Procesi vazhdon të krijojë një sasi të madhe linqesh, ose vazhdon të thërrasë funksionin unlink() vazhdimisht.

3.5.4 Sulmi i Mohimit të shërbimit (Denial of Service)

Ky lloj sulmi është i bazuar në mohimin e aksesit në shërbimet standarte të sistemit. Ai mund të ketë forma të ndryshme, por ne fokusohemi mbi një DoS (Denial of Service) të thjeshtë të shkaktuar nga një proces që konsumon burimet kompjuterike.

Ne vendosëm të implementojmë një test që mund të mbikqyrë parametra të caktuara të procesit çdo sekondë dhe krahason vlerat aktuale me vlerat e marra nga sekondi i fundit. Në fund të sekondit aktual, vlerat e vjetra janë "harruar", vlerat aktuale rregjistrohen si të vjetra dhe procesi përsëritet. Testimet janë si vijon:

- Procesi i përdorimit të CPU (CPU usage) është mbi 50% ose mbi 90%;
- Procesi i përdorimit të memorjes (memory usage) rritet 10x ose 100x;
- Numri i proceseve fork()s rritet 10x ose 100x.

3.5.5 Testimi i anomalive në sjelljet e procesit

Ne gjithashtu kemi implementuar një test që mund të përshkruajë sjelljen normale të një procesi në terma të gjata. Ne shohim thirrjet e sistemit (syscalls) që kryen procesi dhe pastaj krijojmë një bazë të dhënash duke përshkruar se cila nga thirrjet e sistemit mundet drejtëpërdrejtë të ndjekë thirrjet e tjera. Thirrjet e sistemit janë metoda të cilat implementohen në kernel. Këto metoda implementohen në funksion të proceseve të cilat i kanë thirrur këto metoda, të cilët nga ana e tyre implementojne edhe metoda të tjera, në funksion të punës që këto procese kryejnë. Pra, në bazën e të dhënave, ruhet le të themi një historik i implementimit të thirrjeve të sistemit në funksion të proceseve të veçanta, si dhe të thirrjeve (metodave) të tjera, të cilat ndjekin këtë proces. Ky

"historik", përfaqëson një sjellje normale të procesit. Nqs një proces paraqet implementimin e thirrjeve të tjera, jashtë këtij "historiku", kjo paraqet një ndërhyrje e cila duhet raportuar. Mund të ndodhë që ky devijim nga sjellja normale e një procesi, të mos jetë një ndërhyrje. Në këtë rast kemi të bëjmë me alarm False Negative. (kjo është dhe një e metë e teknikës së detektimit të anomalive).

Ky test mund të aplikohet tek çfarëdo lloj ndërhyrjesh, përderisa ai nuk kontrollon për një "patern" të veçantë të ndërhyrjes.

3.5.6 Teste Shtesë

Përveç testeve të gjeneruara duke u nisur nga karakteristikat e sulmeve, ne shtuam dhe disa teste "shtesë", duke besuar se këto teste mund të ndikojnë pozitivisht në performancën e IDS. Këto teste jepen si më poshtë:

- Procesi bën pjesë në grupin "wheel" (GID =0 Group ID);
- Procesi UID nuk është i njëjtë me ekuivalentin e tij EUID (Equivalent UID, një proces SUID).

Ideja e krijimit të testeve shtesë, lindi nga problematika e gjenerimit të alarmeve False Positive. Këto alarme gjenerohen kur procese legjitimi, klasifikohen si të Rrezikshë. Kjo ndodh sepse këto proces mund të jenë duke ekzekutur ndonjë nga testet e përcaktuara. Nëpërmjet testeve shtesë, ne synojmë që këto procese t'i kategorizojmë si të Dyshimta.

3.6 Moduli i kundër-masave

Kundërmasat gjenerohen në bazë të nivelit të pikëve "të fituara" nga procesi, pasi ai ka kaluar testet e përcaktuara në Modulin e Testeve. Disa nga kundër-masat e parashikuara nga Ad-IDS, jepen si më poshtë:

- Gjeneron raportin e procesit;
- "Vret" procesin, duke përfshirë fëmijët e tij (implementimi i thirrjes kill());
- Administratori i IDS-së.

KAPITULLI 4
IMPLEMENTIMI DHE TESTIMI I SISTEMIT Ad-IDS

4.1 Implementimi i sistemit Ad-IDS

Në fazën e implementimit kemi realizuar "ngarkimin" e sistemit Ad-IDS mbi sistemin e operimit të një makine, kompilimin dhe konfigurimin e tij. Më me detaj, këto veprime jepen në paragrafet pasardhëse.

4.1.1 Ambjenti i zhvillimit

Sistemi IDS është zhvilluar dhe testuar mbi sistemin e operimit UNIX FreeBSD, arkitekura i386. Arsyet pse është përdorur ky system operimi jepen si më poshtë:

- FreeBSD është një system i thjeshtë me kernel modular, i cili e bën më të lehtë implementimin e sistemit IDS;
- Është sistem me burim të hapur (open source) dhe mund të gjendet lehtësisht në rrjet;
- Mund të gjenden me shumicë forume dhe dokumentacione mbështetëse për zgjidhjen e problematikave të ndryshme;
- Shumica e "exploits" të sulmeve janë të krijuara për UNIX (duke qenë se shumica e serverave janë UNIX), prandaj e gjykuam më të lehtë implementimin e IDS-së në këtë sistem, sesa përshtatja e exploit-eve për sisteme të tjera operimi.

Kompilatori i përdorur për të ndërtuar sistemin është kompilatori gcc, i cili instalohet si pjesë e paketave shtesë të FreeBSD. Moduli kryesor është pjesa kryesore e sistemit IDS. Ky modul duhet të "montohet" (i bëhet "mount") në direktorinë /usr/src/.

Për të ekzekutuar Ad-IDS, është e nevojshme dhe instalimi i disa librarive shtesë si më poshtë:

- GTK+ (mekanizëm i nevojshëm për X window);
- glib;

- libglade;
- libxml (për gjenerimin e raporteve të tipit xml).

Duke qenë se këto librari nuk janë të pranishme gjatë instalimit të sistemit të operimit, ato duhet të instalohen duke përdorur sistemin port/package, ose duke përdorur komandën portinstall (duke supozuar se programi portupgrade është instaluar dhe konfiguruar).

4.1.2 Kompilimi dhe ngarkimi

Për të ngarkuar modulin IDS është përgatitur një skript (shell script). Për ngarkimin e modulit IDS thjesht ekzekutohet komanda ./load në direktorinë burim, e cila ngarkon skriptin e përgatitur. Ky skript, kryen veprimet e mëposhtme:

- Fshin të gjithë rezultatet e kompilimeve të mëparshme (make clean);
- Kompilon modulin (gjeneron një file të tipit .ko (file i kompiluar));
- Ngarkon modulin në kernel (make load);
- Vendos të gjithë konfigurimet e parametrave sysctl në skedarin e konfigurimit të Ad-IDS.

Moduli mund të ndalojë ekzekutimit duke përdorur komandën "make unload". Në momentin që performohet komanda make unload, sistemi duhet të presë derisa të gjithë proceset të përfundojnë implementimin e thirrjeve të sistemit të nisura gjatë kohës kur moduli ishte i ngarkuar. Ndalimi i modulit papritur mund të shkaktojë dështimin e sistemit, sepse kerneli i cili luan rolin e "manaxhuesit" të thirrjeve të sistemit (syscalls), imponon ekzekutimin e kodeve të modulit IDS, të cilët tashmë janë "ngarkuar".

4.1.3 Konfigurimi

Konfigurimi i sistemit IDS është i përcaktuar në skedarin e konfigurimit (ids.conf), i cili mban:

- parametrat sysctl; këto parametra marrin ose vendosin gjendjen e kernelit;
- variablat e konfigurimit të testeve [4.1.5];
- variablat e konfigurimit të kundërmasave [4.1.6].

Sistemi IDS ka këto parametra konfigurimi sysctl:

- ids.debug; Gjeneron mesazhet e gabimeve (debugging mesages);
- ids.ttl; Përcaktimi i parametrit Time to live. (vlerën e tij e kemi vendosur 60 sekonda) [3.4.1];
- ids.timezone; Caktimi i kohës lokale, e përfaqësuar nga orët e vendosura nga GMT. Ky parametër është përdorur për të printuar informacione mbi datën/orën e kryerjes së çdo veprimi mbi sistem. (Ky variabël është vendosur timezone = 2, sepse ora lokale e Shqipërisë është GMT+2).

Formati i skedarit të konfigurimit, jepet si më poshtë (Shtojca A1.0):

```
# parametrat sysctl

ids.debug=1
ids.ttl=60
ids.timezone=2

# parametrat sysctl te testeve te "sjelljes" se proceseve

# parametrat sysctl te testeve bazuar ne sulmin mbirrjedhja e buferit

# parametrat sysctl te testeve bazuar ne sulmin symlink

# parametrat sysctl te testeve bazuar ne sulmin mohimi i sherbimit

# parametrat sysctl te testeve shtese te propozuar
```

Në skedarin e konfigurimit bëhet vlerësimi i çdo testi me shumën e pikëve përkatëse.

4.1.4 Moduli kryesor

Moduli kryesor duhet të përmbushë funksionet e mëposhtme:

1. Inicializimi i sistemit Ad-IDS;
2. "kapja" e thirrjeve të sistemit (syscalls);
3. Krijimi dhe manaxhimi i bazës së të dhënave, e cila do të mbajë raportet rreth proceseve të marra në shqyrtim nga Ad-IDS.

Kodi burim i modulit kryesor jepet i detajuar në Shtojcën A1.

1. Inicializimi i sistemit Ad-IDS [Shtojca A1.1]

Moduli kryesor shërben si pika hyrëse e Ad-IDS. Ngarkimi i Ad-IDS realizohet nëpërmjet një funksioni ids_loader(), nëpërmjet të cilit parashikohen katër gjendje të Ad-IDS-së:

- Ngarkimi i modulit IDS;
- Moduli IDS nuk është ngarkuar (Ndalimi i modulit);
- Modulit IDS i bëhet "shut down";
- Gjenerimi i gabimeve nga manazhuesi i thirrjeve të sistemit;

Në file duhet të përcaktohet dhe një makro DECLARE_MODULE që është e nevojshme për çdo kernel UNIX për tu startuar.

2. "Kapja" e thirrjeve të sistemit syscalls [Shtojca A1.2]

Me "kapjen" e thirrjeve të sistemit, nënkuptojmë evidentimin e herëve dhe PID të cilat implementojnë një syscall të caktuar. Funksioni më i rëndësishëm për kapjen e thirrjeve të sistemit është funksioni "universal" my_syscalls(), i cili "kap" të gjitha thirrjet e sistemit të implementuara nga procese të ndryshëm. Gjithashtu, kemi të përcaktuar dhe dy funksione: një funksion i cili na kthen numrin e herëve të implementimit të një thirrje sistemi të caktuar dhe një funksion i cili na kthen PID-në e proceseve të cilat impementojnë një thirrje sistemi të caktuar.

Të gjitha thirrjet e sistemit mund të gjenden në direktorinë /sys/kern. [43] Të gjitha syscalls të "kapura", mbahen në një bazë të dhënash të quajtur syscall_db. Kjo

bazë të dhënash është një listë e lidhur (linked-list), e cila na lejon kryerjen e veprimeve si kërkimin, futjen dhe largimin e elementëve nga kjo listë (përkatësisht veprimet janë search, insert dhe remove).

Në bazën e të dhënave, mbahet numri i herëve të implementimit të një thirrje sistemi nga një proces, si dhe PID-në e proceseve që implementojnë një thirrje sistemi të caktuar. (kjo përfaqëson atë që ne e quajmë "historia e sjelljes" së një procesi).

3. Krijimi i bazës së të dhënave të Ad-IDS [Shtojca A1.3]

Baza e të dhënave e Ad-IDS është përgjegjëse për gjenerimin e raporteve të proceseve. Kjo bazë të dhënash menaxhohet si një listë e lidhur (linked list) e strukturave pid_info të përcaktuara në direktorinë include/pid_info.h. Në skedarin pid_info.c, i cili përcakton bazën e të dhënave, duhet të përcaktojmë disa funksione si më poshtë:

- Funksioni update_pid_info() përditëson bazën e të dhënave për një proces të vetëm, rinumëron pikët totale të tij dhe gjeneron një riparaqitje të azhornuar të raporteve. (KUJDES!!! Tek përditësimi i raportit të një procesi, fillimisht duhet shqyrtuar nëse ai proces është fëmijë e një procesi prind; dhe nqs rezulton i tillë, atëhere duhet të përditësohet edhe raporti i procesit prind [3.4.3] , (nqs ky proces nuk ka prind, atëherë përditësohet vetem raporti i tij);
- Funksionet add_process_info() dhe remove_process_info() janë funksione të cilat realizojnë shtimin dhe fshirjen e një procesi në bazën e të dhënave;
- Funksionet lock_pid_info() dhe unlock_pid_info() janë funksione të cilat na lejojnë të aksesojmë bazën e të dhënave dhe të rrisim sigurinë e saj;
- Funksionet init_pid_info() dhe destroy_pid_info() thirren nga funksioni load() në filen inicializues;
- Funksioni get_pid_info() kthen një raport të procesit për një PID të dhënë;
- Funksioni add_report() shton një raport të ri në bazën e të dhënave të proceseve. Përdor një makro ADD_REPORT() të përcaktuar në direktorinë include/pid_info.h.

4.1.5 Moduli i testeve

Moduli i testeve duhet të përmbushë funksionet e mëposhtme:

1. inicializimi i testeve për çdo proces;
2. implementimi i testeve duke u bazuar në algoritmin e detektimit të anomalive (testet e bazuara tek "sjellja" e procesit).
3. implementimi i testeve duke u bazuar në algoritmin e detektimit të ndërfutjeve; testet në lidhje me sulmin e mbirrjedhjes së buferit (buffer overflow) [3.5.2];
4. implementimi i testeve duke u bazuar në algoritmin e detektimit të ndërfutjeve; testet në lidhje me sulmin symlink (symbolic link) [3.5.3];
5. implementimi i testeve duke u bazuar në algoritmin e detektimit të ndërfutjeve; testet në lidhje me sulmin mohimit të shërbimit (denial of service) [3.5.4];
6. implementimi i testeve shtesë [3.5.4].

Kodi burim i detajuar i modulit të testeve jepet në Shtojcën A2.

1. **Inicializimi i testeve për çdo proces [Shtojca A2.1]**

Për të inicializuar testet është përdorur funksioni perform_test(), i cili implementohet për çdo proces. Ky funksion ka për detyrë të "lokalizojë" procesin në bazën e të dhënave, të gjejë PID e tij (check_pid()), të gjenerojë raportin e tij (report()), të azhornojë raportin e tij (update_pid_info()). Kur të gjithë testet dhe veprimet janë përfunduar, funksioni planifikon 1 sekondë kohë vonesë (timeout) për të ekzekutuar përsëri vetëveten, duke përdorur funksionin callout_reset().

2. Testet bazuar tek "sjellja" e procesit [Shtojca A2.2]

Ky test implementon mekanizmin e detektimit të anomalive [2.2]**.** Për çdo proces ai krijon një bazë të dhënash të thirrjeve të sistemit që ky proces implementon (syscall_sequence në strukturën process_database) [4.1.4][Shtojca A1.2].

Kur një proces implementon këtë numër të syscalls, IDS-ja ndërton një strukturë informacioni, në të cilën syscalls janë të rradhitura sipas rradhës së performancës së tyre. Kjo strukturë është ndërtuar në funksionin build_database(). Funksioni check_pid() thirret çdo sekondë, dhe krahason gjendjen aktuale të sekuencave syscall të procesit me informacionin në bazën e të dhënave. Numri i "mospërshtatjeve" me database-n numërohet në përqindje (%) dhe nëse ai është më i madh se 0, një raport vendoset në database-n e raporteve të procesit (pid_info).

3. Testet e bazuar tek sulmi i mbirrjedhjes së buferit [Shtojca A2.3]

Ky test implementon mekanizmin e detektimit të ndërfutjeve, për sulmin e mbirrjedhjes së buferit (buffer overflow). Ky test manaxhon thirrjen e sistemit execve(). Testet e kryera jepen nga parametrat e mëposhtëm sysctl:

- `test.bo.exec_EIP` - thirrjen e execve() me EIP
- `test.bo.exec_SUID` - ekzekutimin e një programi SUID
- `test.bo.exec_SGUID` - ekzekutimin e një programi SGID
- `test.bo.exec_shell` - për të ekzekutuar një shell interaktiv

4. Testet e bazuar tek sulmi symlink [Shtojca A2.4]

Ky test implementon mekanizmin e detektimit të ndërfutjeve, për sulmin e symlink. Ky test manaxhon thirrjet e sistemit link(), symlink() dhe unlink(). Testet e kryera jepen nëpërmjet variablave të mëposhtme sysctl:

- `test.slink.dir` – lista e direktorive të dyshimta (psh. /temp, /nologin etj.)

- `test.slink.exec_symlink` - për të ekzekutuar një"symlink" në një direktori të dyshimtë.
- `test.slink.exec_unlink` - për të bërë unlink
- `test.slink.link_root` - për proceset që nuk po ekzekutohen si root duke krijuar një lidhje në filet root.

5. Testet e bazuar tek sulmi i mohimit të shërbimit [Shtojca A2.5]

Ky test implementon mekanizmin e detektimit të ndërfutjeve, për sulmin e mohimit të shërbimit. Ky test analizon disa parametra të procesit çdo sekondë dhe sipas ndryshimit të këtyre parametrave, ai vendos pikët. Testet e kryera jepen nëpërmjet variablave të mëposhtme sysctl:

- `test.dos.cpu_perq` - procesi i përdorimit të CPU është mbi 50%
- `test.dos.mem_x10` - procesi i përdorimit të memorjes rritet 10x
- `test.dos.mem_x100` - procesi i përdorimit të memorjes rritet 100x
- `test.dos.fork_x10` - procesi i degëzimeve fork() rritet 10x
- `test.dos.fork_x100` - procesi i degëzimeve fork() rritet 100x

6. Testet shtesë [Shtojca A2.6]

Testet shtesë të propozuar, jepen si më poshtë. Secili prej tyre vlerësohet me 0.5 pikë. Këto teste monitorojnë autorizimin e procesit (process permission). Testet e kryera jepen nëpërmjet variablave të mëposhtme sysctl:

- `test.sh.EUID` - procesi ekzekutohet me: EUID = 0

- `test.sh.group_0` - procesi ekzekutohet si një element i grupit 0

4.1.6 Moduli i kundër-masave

Moduli i kundër-masave, duhet të përmbajë funksione të cilat paraqesin veprime automatik, që duhet të ndërmerren në momentin e detektimit të një ndërhyrje. Ky modul përmban funksionet e mëposhtme:

1. të përfundojë një proces (t'i bëjë kill() këtij procesi...KUJDES!!! jo close() ose shutdown());
2. të njoftojë me email administratorin e Ad-IDS, për ndërhyrjet e detektuara.

Për proceset të cilët paraqesin një aktivitet "të Dyshimtë", është përcaktuar që administratori i Ad-IDS, të ndërhyjë manualisht për të zgjidhur situata të ndryshme.

Moduli i kundër-masave automatike të gjeneruara nga Ad-IDS, është në fokusin e punimeve në të ardhmen, së bashku me disa përmirësime të mundshme të sistemit Ad-IDS.

Kodi burim i modulit të kundër-masave, jepet në Shtojcën A.3.

1. **Kill() një proces [Shtojca A3.1]**

Në momentin kur një proces është përcaktuar si "i Rrezikshëm", atëhere sistemit Ad-IDS në mënyrë automatike e asgjeson ose e"vret" këtë proces duke implementuar funksionin kill(). KUJDES!!! Ad-IDS duhet të shqyrtojë edhe fëmijët e këtij procesi, në mënyrë që të asgjesohen dhe ato.

2. **Njofto me email [Shtojca A3.2]**

Për të realizuar njoftimin me email të administratorit është përdorur një server SMTP të specifikuar, i cili dergon mesazhe një klienti SMTP. Për të realizuar komunikimin klient/server duhet të përcaktohen parametrat sysctl të mëposhtëm:

- adresën IP të serverit SMTP;
- porta e serverit SMTP (në default është 25);
- e-mail i adresës marrëse;
- koha kufi për të përsëritur raportet e-mail;
- minimumi i pikëve të procesit për t'u aktivizuar njoftimi me email.

4.2 Testimi i sistemit Ad-IDS

Sistemi Ad-IDS, do të vendoset "përballë" sulmeve të mëposhtme:

Test 1 : Sulmi Mohimi i Shërbimit

Test 2 : Sulmi i Mbirrjedhjes së buferit

Test 3: Sulmi Symlink

Test 4: Ndryshimi i "sjelljes"

Test 5: Testet Shtesë

Metodologjia e përdorur për të realizuar këto teste jepet në paragrafin e mëposhtëm.

4.2.1 Metodologjia e përdorur

Për të gjithë testet e kaluara procesi merr një vlerësim që ne e quajmë "pikë".

Në përfundim të të gjitha testeve, procesi ka një shumatore të të gjitha pikëve "të fituara". Kjo shumatore llogaritet si më poshtë formula (1).

$$(\sum_{p \in P} p) \quad (1)$$

P = {shuma e të gjithë pikëve}

Në varësi të nivelit të pikëve të fituara, procesi përcaktohet si i Rrezikshëm, i Dyshimtë apo i Parrezikshëm. Duhet të përcaktojmë dy gjëra:

- Sa pikë do të fitojë procesi për teste të veçanta;
- Çfarë niveli i pikëve do të konsiderohet një ndërhyrje.

Pikët për teste të ndryshme vendosen manualisht në skedarin e konfigurimit .config. [Shtojcës A1.0]. Ato janë të konfigurueshme dhe mund të ndryshohen në varësi "të sjelljes" së Ad-IDS.

- Vendosja e pikëve për teste të veçanta është bërë në këtë mënyrë: Themi se një proces "e kalon" testin në rastin kur ai nuk është duke ekzekutuar ndonjë nga kriteret e tij (këto kritere janë të përcaktuara për çdo tip sulmi)[referoju modulit të sulmeve kap3]. Në këtë rast, pra kur procesi nuk është duke ekzekutuar ndonjë nga kriteret e testit, vlerësimi i tij bëhet me 1 pikë. Vlerësimi me 1 pikë, bëhet për arsyen që, gjithsesi të gjenerohet një raport mbi këtë proces. Në rastin kur procesi është duke ekzekutuar ndonjë nga kriteret e testit, atëhere vlerësimi i tij bëhet me 100 pikë. Në rastin e testimit të sjelljes së procesit (test [syscall_db]), procesi do vlerësohet me 50 pikë. Ky vlerësim i tillë bëhet për arsye që procesi të karakterizohet si i Dyshimtë, në mënyrë që të trajtohet nga administratori i Ad-IDS. (administratori vendos nëse do krijojë model të ri apo jo për këtë proces).
- **Niveli i pikëve**: Në bazë të nivelit të pikëve, proceset do kategorizohen si më poshtë:

- **I Parrezikshëm**; një proces konsiderohet i parrezikshëm kur niveli i pikëve të tij është në këtë diapazon vlerash:

  niveli minimal = 1 pikë

  niveli maksimal = numri i kritereve të paekzekutuara x 1 pikë (në rastin tonë është 14)

- **I Rrezikshëm;** një proces konsiderohet i rrezikshëm kur niveli i pikëve të tij është në këtë diapazon vlerash:

niveli minimal = 100 pikë (ka ekzekutuar të paktën një kriter të një testi)

niveli maksimal = numri i kritereve të ekzekutuara x 100 pikë (në rastin tonë është 14000)

- **I Dyshimtë;** një proces konsiderohet i dyshimtë kur niveli i pikëve të tij është në këtë diapazon vlerash:

  niveli minimal = niveli max i Parrezikshëm + 1 pikë (në rastin tonë është 15)

  niveli maksimal = niveli min i Rrezikshëm – 1 pikë (në rastin tonë është 99)

SHËNIM 1: Numri total i kritereve të testeve, nuk përfshin kriteret e testeve Shtesë, të cilët kanë tjetër koefiçent pikësh.

Metodologjia e përdorur për llogaritjen e pikëve të një procesi, paraqet një problematikë. Ekzistojnë disa kritere, të cilat kanë mundësi të na rrisin "piket" e procesit, ndërkohë që këto procese janë legjitime. Ne e zgjidhëm këtë çështje duke vendosur kritere Shtesë si më poshtë. [3.5.6]. Kemi krijuar dy kritere për testet shtesë dhe të dyja kanë të bëjnë me autorizimin e proceseve (permissions).

$$\left(\sum_{p \in P} p\right) \cdot \prod_{s \in S} s \qquad (2)$$

P = { shuma e të gjithë pikëve}

S = {shuma e të gjithë koeficentëve Shtesë}

Këto kritere, reflektohen ne setin e testeve Shtesë [referoju kapitullit 3, kap 4], pjesë e Modulit të Testeve. Formula (2), paraqet mënyrën e propozuar për llogaritjen e pikëve të proceseve, për sistemin Ad-IDS. Nqs procesi i cili është duke implementuar një nga kriteret e përcakutara në testet Shtesë, atëhere ai vlerësohet me 0.5 pikë. Ky vlerësim bëhet i tille sepse, duke u shumëzuar me shumën e pikëve totale (formula (2)), niveli i pikëve totale të procesit do të përgjysmohet (sepse shumëzohet me 1/2), duke synuar në këtë mënyrë që sistemi të paktën të kategorizohet si I Dyshimtë. (më pas të mund të merret në shqyrtim nga administratori i sistemit). Administratori i sistemit, nga

ana e tij, mund të vendosë për të krijuar ose jo, një model (pattern) të tij për ta vendosur në bazën e të dhënave të sjelljeve të proceseve, duke evituar kështu alarmet e tipit False Positive.

SHËNIM 2: Në rastin kur një proces ekzekuton vetëm një kriter dhe ai është kriter i testeve shtesë, atëhere procesi vlerësohet me 0.5 pikë (nuk bën pjesë në asnjë nga diapazonet e përcaktuara më sipër). Në këtë rast të veçantë, procesi kategorizohet si i Parrezikshëm.

4.2.2 Testi 1- Sulmi i Mohimit të shërbimit

Sulmi i Mohimit të shërbimit është një "forkmebomb" (fork memory bomb), i cili nëpërmjet thirrjes fork() krijon procese fëmijë, të cilëve ju alokohet nga sistemi një hapësirë në memorje [44]. Me rritjen e numrit të proceseve fëmijë, e gjithë memorja okupohet nga këta procese dhe si rrjedhim proceset "legjitime" e kanë të pamundur procesimin e tyre nëpërmjet burimeve të makinës (ndosh mohimi i shërbimit). Si përfundim, pasi okupohet e gjithë memorja, kemi të bëjmë me një bllokim të sistemit "proces dump".

Ekzekutimi i këtij sulmi jepet si më poshtë:

```
renalda#

renalda# pwd

/usr/src/ad-ids/test/denial_of_service/
renalda# cc forkmembomb.c
renalda# ./
renalda# mv a.out forkmembomb_attack
renalda# ls -al
total 14
drwxr-xr-x  2 root  wheel   512 Apr  20 10:39 .
drwxr-xr-x  3 root  wheel   512 May 25 16:22 ..
-rw-r--r--  1 root  wheel   273 May 25 16:22 README
```

```
-rw-r--r--  1 root  wheel   790 May 25 16:22 forkmembomb.c
-rwxr-xr-x  1 root  wheel  5039 Apr  20 10:39 forkmembomb_attack
renalda# ./forkmembomb_attack
forkbomb_attack in malloc(): error: allocation failed
Abort (core dumped)
renalda# ls -al
total 51694 2
505M  forkmembomb_attack.core
```

Pasi kompilohet file-i forkbomb.c, sistemi gjeneron një file a.out. Përmbajtjen e a.out (sulmi i kompiluar), e kemi "hedhur" në file-n forkmebomb_attack (nëpërmjet komandës mv a.out forkmebomb_attack). Ekzekutimi i sulmit realizohet nëpërmjet komandës ./forkmebomb_attack.

Raporti i gjeneruar nga Ad-IDS, pas ekzekutimit të sulmit të Mohimit të Shërbimit, jepet si më poshtë:

```
Procesi 13635 [a.out]
 - total 300 pike
 - Mon Ap  20 2015 10:39:04
* Mon April  20 2015 10:39:09, 100 pike [test.dos.mem_x10]
   Perdorimi i memorjes (KB) rritet 10x
* Mon April  20 2015 10:39:05, 100 pike [test.dos.fork_x10]
   Implementimi i fork() rritet 10x
* Mon April  20 2015 10:39:13, 100 pike [test.dos.mem_x100]
   Perdorimi i memorjes (KB) rritet 100x
```

Nga ekzekutimi i komandës top, mund të shohim rezultatet e mëposhtme:

```
renalda# top
last pid: 13692;
75 processes:  2 running, 73 sleeping
CPU states:  0.0% user,  0.0% nice,  0.7% system,  0.3% interrupt, 99.0% idle
Mem: 521M Active, 229M Inact, 91M Wired, 8K Cache, 111M Buf, 155M Free
Swap: 2023M Total, 2023M Free

  PID USERNAME PRI NICE   SIZE    RES STATE    TIME   WCPU    CPU COMMAND
  405 root      96    0  3500K  2720K select   0:00   0.00%  0.00% sendmail
  253 root      96    0  1312K   880K select   0:04   0.00%  0.00% syslogd
  426 root       8    0  1336K  1032K nanslp   0:02   0.00%  0.00% cron
  352 root      96    0  1236K   764K select   0:01   0.00%  0.00% usbd
  688 root      96    0  3488K  2792K select   0:00   0.00%  0.00% sshd
  408 smmsp     20    0  3400K  2664K pause    0:00   0.00%  0.00% sendmail
13450 root      96    0  6220K  3116K select   0:00   0.00%  0.00% sshd
  637 root       5    0  2308K  1816K ttyin    0:00   0.00%  0.00% csh
13452 root       5    0  2304K  1872K ttyin    0:00   0.00%  0.00% csh
  636 root       8    0  1648K  1408K wait     0:00   0.00%  0.00% login
13631 root      96    0  2288K  1544K RUN      0:00   0.00%  0.00% top
13625 root      96    0  6220K  3116K RUN      0:00   0.00%  0.00% sshd
  450 root      96    0  1228K   708K select   0:00   0.00%  0.00% moused
13627 root      20    0  2260K  1788K pause    0:00   0.00%  0.00% csh
  480 root       5    0  1276K   944K ttyin    0:00   0.00%  0.00% getty
  481 root       5    0  1276K   944K ttyin    0:00   0.00%  0.00% getty
  485 root       5    0  1276K   944K ttyin    0:00   0.00%  0.00% getty
  483 root       5    0  1276K   944K ttyin    0:00  0.00%  0.00% getty
  482 root       5    0  1276K   944K ttyin    0:00   0.00%  0.00% getty
  484 root       5    0  1276K   944K ttyin    0:00   0.00%  0.00% getty
  479 root       5    0  1276K   944K ttyin    0:00   0.00%  0.00% getty
13635 root       8    0  1264K   596K nanslp   10:39:05  0.00%  0.00% forkmembomb_attack
  161 root      20    0  1180K   644K pause    0:00   0.00%  0.00% adjkerntz
13656 root       8    0 32604K   720K nanslp   10:39:12 0.00%  0.00% forkmembomb_attack
13683 root       8    0   444M  1348K nanslp   10:39:21 0.00%  0.00% forkmembomb_attack
13657 root       8    0 48248K   744K nanslp   10:39:12 0.00%  0.00% forkmembomb_attack
13682 root       8    0   429M  1324K nanslp   10:39:20 0.00%  0.00% forkmembomb_attack
13674 root       8    0   307M  1140K nanslp   10:39:17 0.00%  0.00% forkmembomb_attack
13687 root       8    0   505M  1440K nanslp   10:39:22 0.00%  0.00% forkmembomb_attack
13668 root       8    0   215M  1000K nanslp   10:39:15 0.00%  0.00% forkmembomb_attack

renalda#
```

Figura 4.1 Rezultatet pas ekzekutimit të exploitit forkmembomb

Duke ekzekutuar komandën grep | forkmembomb_attack, filtrojmë të gjitha proceset që lidhen me këtë sulm.

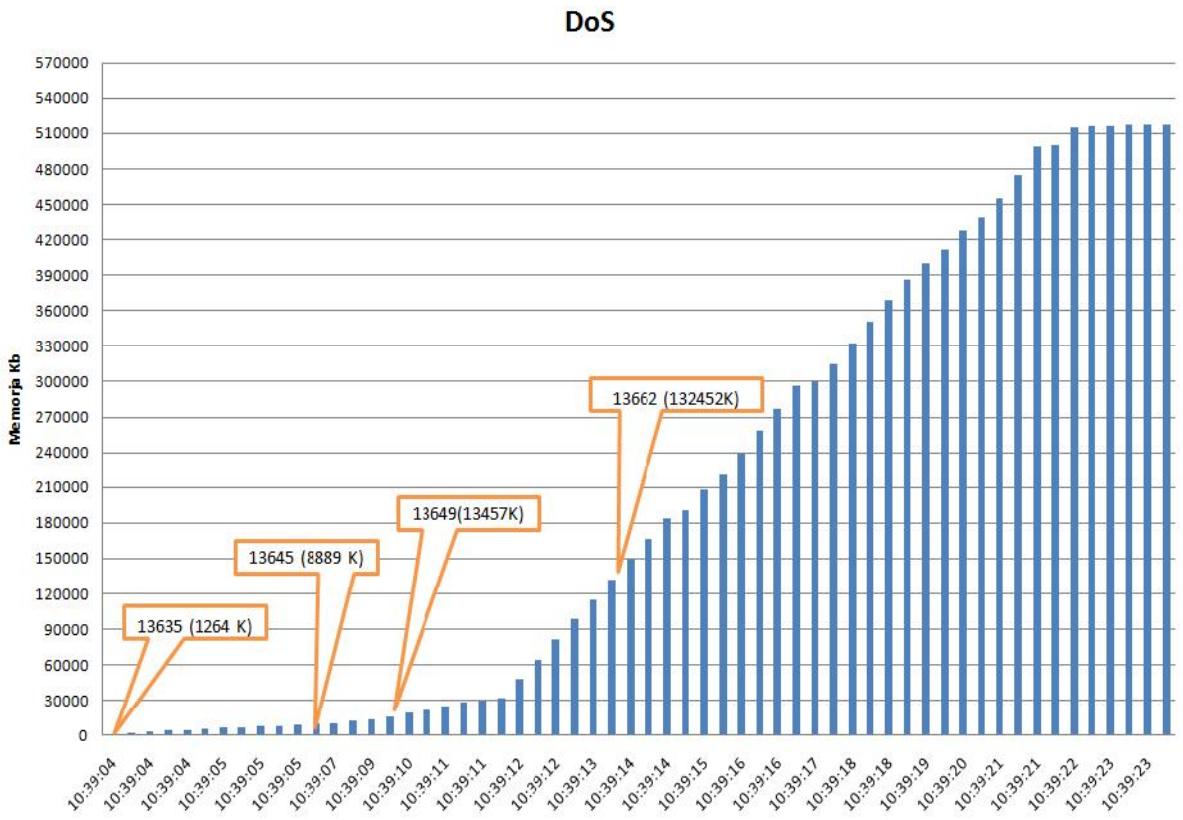

Figura 4.2 Paraqitja grafike e rezultateve të testit DoS

Analiza e detajuar e rezultateve, jepet si më poshtë:

- Koha e inicializimit të sulmit DoS është: `Mon Ap 20 2015 10:39:04;`
- Ekzekutimi i thirrjeve fork(), testi `[test.dos.fork_x10]: Mon April 20 2015 10:39:05.` Pra shohim se për 1 sekondë nga inicimi i exploit-it, kemi 10-fishimin e implementimit të thirrjes fork, që nënkupton krijimin e 10 proceseve me PID 13635-13645;
- Rritja e memorjes në Kb testi `[test.dos.mem_x10]: Mon April 20 2015 10:39:09.` Pra shohim se për 5 sekonda nga inicimi i exploit-it, kemi 10-fishimin e përdorimit të memorjes, përkatësisht për PID 13635- 1264K dhe pas 5sekondave kemi PID 13649 – 13457K. (gjithësej gjatë kësaj kohe janë krijuar 14 procese fëmijë);

- Rritja e memorjes në Kb testi `[test.dos.mem_x100]: Mon April 20 2015 10:39:13.` Pra shohim se për 9 sekonda nga inicimi i exploit-it, kemi 100-fishimin e përdorimit të memorjes, përkatësisht për PID 13635- 1264K dhe pas 9 sekondave kemi PID 13662 – 132452K. (gjithësej gjatë kësaj kohe janë krijuar 27 procese fëmijë);
- Testet `[test.dos.fork_x100] dhe [test.dos.cpu_perq]` nuk ekzekutohen nga procesi. (Shohim se CPU nuk ka ndikim nga ky exploit dhe gjithashtu kemi vetem 57 procese fëmijë të krijuar (jo x100));
- Përfundimi i ekzekutimit të exploit-it realizohet pas 18 sekondash: `Mon April 20 2015 10:39:23`. Pas 18 sekondash kemi krijimin e 57 proceseve fëmijë me PID 13635-13692. Ku procesi PID 13635 është procesi prind. Pas kësaj kohe, procesi PID 13692, ka një përdorim të memorjes prej 516942K. Pas kësaj kohe, sistemi gjeneron mesazhin "core dump".

PID	koha	memorja (Kb)
13635	10:39:04	1264
13636	10:39:04	2010
13637	10:39:04	2998
13638	10:39:04	4123
13639	10:39:04	4678
13640	10:39:04	6123
13641	10:39:05	6948
13642	10:39:05	7068
13643	10:39:05	7856
13644	10:39:05	8245
13645	10:39:05	8889
13646	10:39:06	10423
13647	10:39:07	10879
13648	10:39:08	12123
13649	10:39:09	13457

13650	10:39:10	16345
13651	10:39:10	19452
13652	10:39:10	22123
13653	10:39:11	24988
13654	10:39:11	28456
13655	10:39:11	30012
13656	10:39:12	32604
13657	10:39:12	48248
13658	10:39:12	64124
13659	10:39:12	81452
13660	10:39:13	98416
13661	10:39:13	115123
13662	10:39:13	132452
13663	10:39:14	149124
13664	10:39:14	166612
13665	10:39:14	183345
13666	10:39:14	190789
13667	10:39:15	208123
13668	10:39:15	220160
13669	10:39:16	239124
13670	10:39:16	258452
13671	10:39:16	277452
13672	10:39:17	296754
13673	10:39:17	300304
13674	10:39:17	314368
13675	10:39:18	332423
13676	10:39:18	350012
13677	10:39:18	368452
13678	10:39:19	386452
13679	10:39:19	399897

13680	10:39:20	410982
13681	10:39:20	427456
13682	10:39:20	439296
13683	10:39:21	454656
13684	10:39:21	474452
13685	10:39:21	498755
13686	10:39:21	500101
13687	10:39:22	515072
13688	10:39:22	515601
13689	10:39:23	516452
13690	10:39:23	516724
13691	10:39:23	516855
13692	10:39:23	516942

Tabela 4.1 Rezultatet nga testi Mohimi i shërbimit

4.2.3 Testi 2- Sulmi Mbirrjedhja e buferit (Buffer overflow)

Për të realizuar sulmin e Mbirrjedhjes së buferit, kemi përdorur exploit-in terminatorX [44][kapitulli 3]. Ky exploit është i tipit "local root". Komandat për të ekzekutuar këtë exploit janë:

```
* ./terminatorX-exp [-r <RET>][-b [-s <STARTING_RET>]]
*
* -r <RET>: kjo komandë nuk është "bruteforce"; duke ekzekutuar
këtë komandë,      *ekzekutohet shellcode me <RET> e cila përcakton
kthimin e një adrese të memorje ku po bëhet  "mbirrjedhja" (RET-return
address)
*
* -b: kjo komandë ekzekuton exploit-in si "bruteforce", duke e
detyruar të ekzekutohet me çdo kusht, pa na kthyer adresën e fundit të
memorjes ku ndodh
* "mbirrjedhja".
*
```

* -s <STARTING_RET>: kjo komandë ekzekuton exploit-in si "bruteforce", duke e detyruar të ekzekutohet me çdo kusht, si dhe na kthen adresën e parë të memorjes ku ndodh "mbirrjedhja" <STARTING_RET>-starting return address.

Ekzekutimi i këtij sulmi e kam realizuar si "bruteforce attack", e cila na kthen adresën e fundit të memorjes ku ndodh "mbirrjedhja". Ekzekutimi i tij jepet si më poshtë:

```
renalda# pwd

/usr/src/ad-ids/test/buffer_overflow/

renalda# cc terminatorX-exp.c
renalda# ls -al
total 14
drwxr-xr-x  2 root  wheel   512 Apr  20 10:13 .
drwxr-xr-x  3 root  wheel   512 Apr 29 16:22 ..
-rw-r--r--  1 root  wheel   273 Apr 29 16:22 README
-rwxr-xr-x  1 root  wheel  5039 Apr  20 10:13 a.out
-rw-r--r--  1 root  wheel   790 Apr 29 16:22 terminatorX-exp.c
renalda# ./terminatorX-exp -b
```

Nëpërmjet komandës cc terminatorX-exp.c, bëhet kompilimi i këtij file dhe si rezultat i kompilimit na gjenerohet një file a.out. Ndërsa komanda ./terminatorX-exp –b na ekzekuton exploit-in, i cili duke qenë –b, duhet të na kthejë dhe adresën e fundit ku ndodh "mbirrjedhja".

```
*
*[+] Starting bruteforcing...
*[+] Testing 0xbfbfe11b...
*[+] Testing 0xbfbfe5ac...
*terminatorX Release 3.81 - Copyright (C) 1999-2003 by Alexander König
```

```
*terminatorX comes with ABSOLUTELY NO WARRANTY - for details read the license.
*... . . .
*sh-2.05b# exit
*exit
*[+] Exited: shell's ret code = 0
*[+] Ret address found: 0xbfbfe5ac
 *
 */
```

Nga ekzekutimi i exploit-it, shohim se adresa e fundit ku ndodh "mbirrjedhja" është 0xbfbfe5ac. Pas ekzekutimit të këtij sulmi, rezultatet e detektuara nga Ad-IDS, në trajtën e raporteve, jepen si më poshtë:

```
Procesi 6759 [a.out]
       - total 100 pike
       - Mon Ap  20 2015 10:15:23
        * Mon April 20 2015 10:15:29, proces femije 6760 [terminatorX], 100 pike [test.bo.exec_EIP ]
         Procesi ekzekuton /bin/sh me EIP (0xbfbfe5ac) ne stack/data adrese

Process 6760 [sh]
       - total 200 pike
       - Mon Ap  20 2015 10:15:23
       * Mon April  20 2015 10:15:29, 100 pike [test.bo.exec_shell]
         Procesi ekzekuton nje shell interaktiv (/bin/sh)
       * Mon April  20 2015 10:15:29, 100 pike [test.bo.exec_EIP]
         Procesi ekzekuton /bin/sh me EIP (0xbfbfe5ac) ne stack/data adrese
```

```
Process 1852 [bash]
        - total 100 pike
        - Mon Ap  20 2015 10:15:23
        * Mon April  20 2015 10:15:29, proces femije 6760 [terminatorX],
100 pike [test.bo.exec_EIP]
  Procesi ekzekuton /bin/sh me EIP (0xbfbfe5ac) ne stack/data adrese
```

PID	ora	koha (s)	testi	piket
6759	10:15:23-10:15:29	6	test.bo.exec_EIP	100
6760	10:15:23-10:15:29	6	test.bo.exec_shell; test.bo.exec_EIP	200
1852	10:15:23-10:15:29	6	test.bo.exec_EIP	100

Tabela 4.2 Rezultatet nga testi “Mbirrjedhja” e buferit

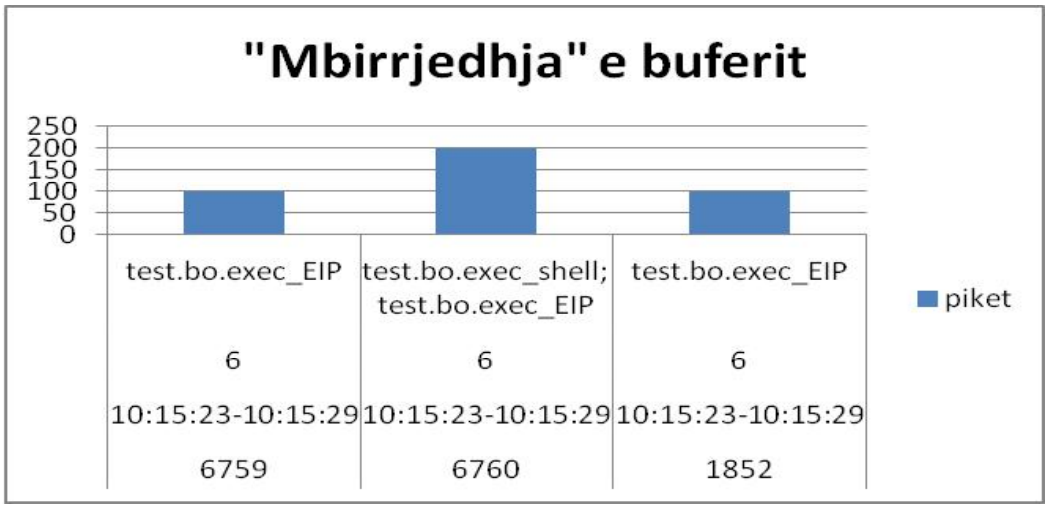

Figura 4.3 Paraqitja grafike e testit Mbirrjedhja e buferit

Nga rezultatet e marra nga Ad-IDS, mund të themi se:

- Koha e ekzekutimit të exploitit është përafërsisht 2 min (`Apr 20 10:13- Mon Ap 20 2015 10:15:29`)
- Exploiti krijon një proces terminatorX me PID 6760, i cili realizon "mbirrjedhjen e buferit" (adresa e fundit është `0xbfbfe5ac`), i cili nga ana e tij krijon dy procese fëmijë me PID 6759 dhe 1852.
- Gjenerimi i raportit nga Ad-IDS, në lidhje me këtë sulm, ndodh pas 6 sekondash.

4.2.4 Testi 3- Sulmi Symlink

Për të realizuar sulmin Symlink, kemi përdorur exploit-in xine (xine-bugreport) [44][kapitulli 3]. Ekzekutimi i këtij exploit-i realizohet thjesht duke shkruar në shell xine-bugreport, i cili është një skript i gatshëm, nëpërmjet të cilit ekzekutohet exploit-i. Ekzekutimi i këtij sulmi jepet si më poshtë:

```
renalda# pwd

/usr/src/ad-ids/test/symlink/


renalda# xine-bugreport
Please be patient, this script may take a while to
run...
logging to /tmp/xine-check.log... Ap 20 10:16


         press <enter> to continue...


[ good ] looks like you have a /proc filesystem
mounted.
[ good ] found the player at /usr/bin/xine
[ good ] /usr/bin/xine is in your PATH
. . . . . .
```

```
         press <enter> to continue...
[ good ] plugin directory /usr/lib/xine/plugins
exists.
[ good ] found input plugins
[ good ] found demux plugins
[ good ] found decoder plugins
[ good ] found video_out plugins
[ good ] found audio_out plugins
[ good ] skin directory /usr/share/xine/skins exists.
[ good ] found logo in /usr/share/xine/skins
[ good ] I even found some skins.
. . . . . .

You should include a _complete_ copy of xine's output
in your bug report.
You can either copy&paste this output from the
terminal where you ran xine,
or you can collect xine's output in a file named
/tmp/xine.out, using
this command:
xine >/tmp/xine.out 2>&1
(assuming you have a Bourne compatible shell, like
bash, for example)
If you need to add any parameters, you can do so...
This method is useful if you want to remove part of
the output...
Which method would you prefer?
1) copy&paste
2) logfile /tmp/xine.out
please select (1..2): 2
```

```
please press <return> when you have the log ready in
/tmp/xine.out

You may add the output later, if this wasn't your
intention...
         press <enter> to continue...



Okay. That's all I could guide you through...
I have assembled a skeleton for your bugreport in the
file

   /tmp/xine-bugreport



[...]

renalda#  ls -al /etc/nologin
-rw-r--r--    1 root     root         1756 Ap 20
10:20 /etc/nologin
renalda#
```

Sic mund të shohim, në fund të ekzekutimit të këtij exploit-i, është krijuar një "symbolic link" me një direktori të dyshimtë, siç është direktoria nologin/.

Rezultatet e marra nga Ad-IDS, pas ekzekutimit të këtij exploit-i, jepen si më poshtë:

```
Procesi 6831 [bash]
       - total 100 pike
```

```
        - Mon Ap  20 2015 10:16:17
        * Mon April  20 2015 10:16:17, proces femije 6834 [sh], 100 pike
[test.slink.link_root]
          Procesi ekzekeutohet si UID 0 krijon nje link /dev/tty ne UID
1000


Procesi 6851 [rm]
        - total 100 pike
        - Mon Ap  20 2015 10:16:29
         * Mon April  20 2015 10:16:17, proces prind 6834 [sh], 100 pike
[test.slink.link_root]
          Procesi ekzekutohet si UID 0 krijon nje link /dev/tty ne UID
1000


Process 6988 [ln]
        - total 200 pike
        - Mon Ap  20 2015 10:18:30
        *    Mon    April        20    2015    10:18:30,    200    pike
[test.slink.exec_symlink]
       [test.slink.link_root]
          Procesi ekzekutohet si UID 1000 krijon nje link /tmp/xine.out
-> /etc/nologin ne direktorine UID 0


Process 1852 [bash]
        - total 200 pike
        - Mon Ap  20 2015 10:15:08
        * Mon April  20 2015 10:18:30, proces femije 6988 [ln], 200 pike
[test.slink.exec_symlink][test.slink.link_root]


          Procesi ekzekutohet si UID 1000 krijon nje link /tmp/xine.out
-> /etc/nologin ne direktorine UID 0
Procesi 6993 [sh]
        - total 100 pike
```

- Mon Ap 20 2015 10:18:35

* Mon April 20 2015 10:18:35, 100 pike [test.slink.exec_symlink]

Procesi krijon nje link /tmp/ xine.out -> /etc/nologin ne nje direktori te dyshimte

Process 7095 [xine]

- total 200 pike

- Mon Ap 20 2015 10:19:57

* Mon April 20 2015 10:19:57, proces prind 7095 [sh], 200 pike [test.slink.exec_symlink] [test.slink.link_root]

Procesi ekzekutohet si root krijon nje link /tmp/xine-bugreport (-> /etc/nologin) ne direktorine UID 1000

Procesi 7704 [cat]

- total 100 pike

- Mon Ap 20 2015 10:20:08

* Mon April 20 2015 10:19:57, proces prind 7095 [sh], 100 pike [test.slink.exec_symlink]

Procesi ekzekutohet si root krijon nje link /tmp/xine-bugreport (-> /etc/nologin) ne nje direktori te dyshimte

Procesi 7708 [sh]

- total 200 pike

- Mon Ap 20 2015 10:20:11

* Mon April 20 2015 10:20:11, 100 pike [test.slink.exec_symlink]

Procesi ekzekutohet si rootkrijon nje link /tmp/xine-bugreport (-> /etc/nologin) ne nje direktori te dyshimte

* Mon April 20 2015 10:19:57, proces prind 7095 [sh], 100 pike [test.slink.exec_symlink]

Procesi ekzekutohet si root krijon nje link /tmp/xine-bugreport (-> /etc/nologin) ne nje direktori te dyshimte

PID	ora	koha	testi	piket
6831	10:16:17	1	test.slink.link_root	100
6851	10:16:17-10:16:29	12	test.slink.link_root	100
6988	10:18:30	1	test.slink.exec_symlink; test.slink.link_root	200
1853	10:15:30-10:18:30	1	test.slink.exec_symlink; test.slink.link_root	200
6993	10:18:35	1	test.slink.exec_symlink	100
7095	10:19:57	1	test.slink.exec_symlink; test.slink.link_root	200
7704	10:19:57-10:20:08	11	test.slink.exec_symlink	100
7708	10:20:11	1	test.slink.exec_symlink	200

Tabela 4.3 Rezultatet e testit Symlink

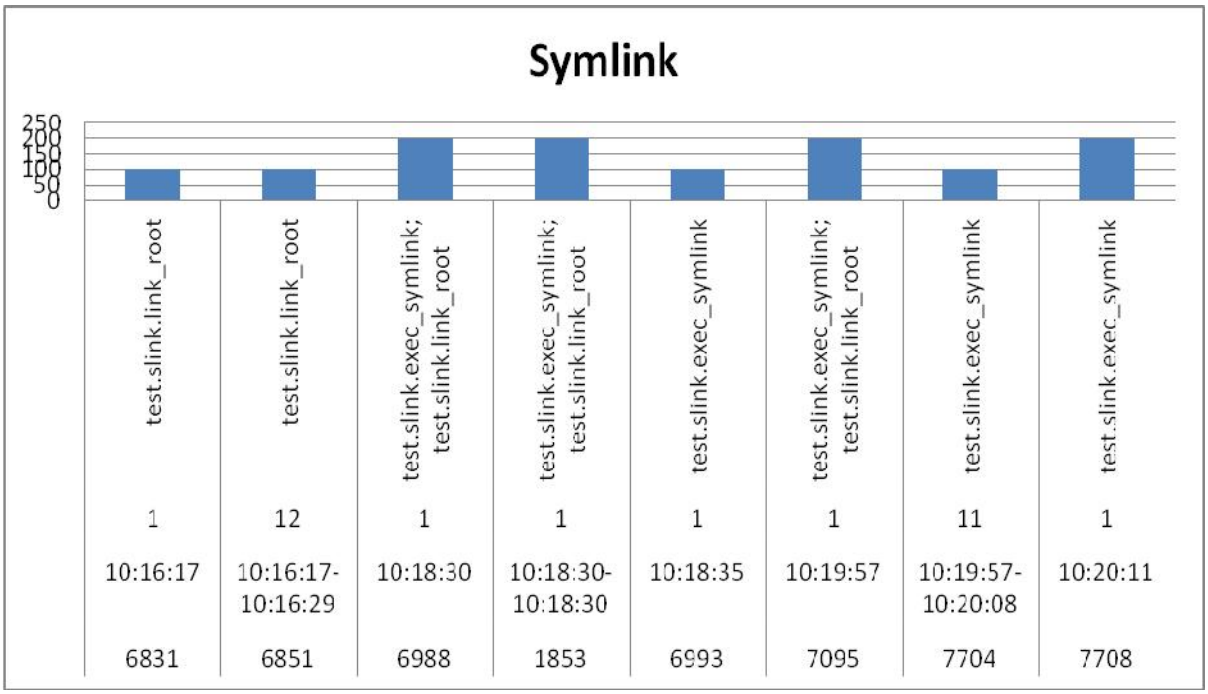

Figura 4.4 Paraqitja grafike e rezultateve të testit Symlink

Nga rezultatet e marra nga Ad-IDS, mund të themi se:

- Exploiti krijon dy procese me PID 7095 dhe 6988, të cilët nga ana e tyre krijojnë procese fëmijë të cilët krijojnë link në direktori të dyshimta ose në direktori me privilegje root.
- Gjenerimi i raportit të parë nga Ad-IDS, në lidhje me këtë sulm, ndodh brenda minutit të parë të ekzekutimit të expolitit (përafërsisht 17 sekonda) dhe raporti i fundit ndodh pas afërsisht 4 min.
- Koha e detektimit të ndërhyrjes që nga momenti i krijimit të procesit, varjon nga 1 në 12 sekonda.

4.2.5 Testi 4- Ndryshimi i "sjelljes"

Në këtë test, kemi trajtuar një proces, historikun e "sjelljes" së tij e kemi të njohur. Historiku i sjelljes së një procesi, përcaktohet nga thirrjet e sistemit (syscalls) që ai implementon, sipas një rradhe të caktuar. Ky historik, ruhet në bazën e të dhënave syscall_db. Kemi përcaktuar që në këtë bazë të dhënash, të ruhen syscalls të implementuara nga një proces PID, si dhe rradha e implementimit të tyre.

Për të parë thirrjet e sistemit të cilat implementon një proces, përdorim komandën dtrace si më poshtë:

```
renalda# !/usr/bin/sh
renalda#
# dtrace - print process system call time details.
#          Written using DTrace (Solaris 10 3/05).
#
#
# USAGE: dtrace [-acdeflhoLs] [-t syscall] { -p PID | -n name |
command }
#
#          -p PID          # examine this PID
#          -n name         # examine this process name
#          -t syscall      # examine this syscall only
#          -a              # print all details
```

```
      #          -c              # print system call counts
      #          -d              # print relative timestamps (us)
      #          -e              # print elapsed times (us)
      #          -f              # follow children as they are forked
      #          -l              # force printing of pid/lwpid per line
      #          -o              # print on cpu times (us)
      #          -s              # print stack backtraces
      #          -L              # don't print pid/lwpid per line
      #          -b bufsize      # dynamic variable buf size (default
is "4m")
      #
```

```
renalda# dtrace -p syscall:::entry'/pid == 1871/{ @syscalls[probefunc] =
count(); }'
dtrace: description 'syscall:::entry' matched 215 probes
^C

  open                                                              1
  lwp_park                                                          2
  times                                                             4
  fcntl                                                             5
  close                                                             6
  sigaction                                                         6
  read                                                             10
  ioctl                                                            14
  sigprocmask                                                     106
  write                                                          1092
```

Figura 4.5 Thirrjet e sistemit të ekzekutuara për një PID të dhënë

Për të realizuar testin mbi detektimin e anomalive, kam përdorur këtë metodologji. Për një proces të dhënë psh PID 1871, kemi të ruajtuar në bazën e të dhënave syscall_db, historikun e sjelljes së tij (numri i syscalls dhe rradha e implementimit të tyre). Duke qene se për një proces është e vështirë të ndryshojmë sjelljen e tij, atëhere thjesht kam ndryshuar rekordin e tij në bazën e të dhënave duke ndryshuar rradhën e implementimit të syscalls (për të njëjtin PID). Kuptohet se, në momentin e ekzekutimit të procesit PID 1871, historiku i tij nuk do të korrespondojë me

atë në bazën e të dhënave dhe si rrjedhojë do të gjenerohet një raport nga Ad-IDS. Duke qenë se kemi të bëjmë me test bazuar në sjelljen e procesit, ai vlerësohet me 50 pikë.

Rezultati i përftuar nga Ad-IDS jepet si më poshtë:

```
Procesi 1871 [syscall_db]
        - total 50 pike
        - Mon Ap  20 2015 12:05:18
       * Mon April  20 2015 12:05:18, 50 pike [syscall_db]
          Detektim i anomalise
```

4.2.6 Testi 5- Testet Shtese

Testet shtesë të propozuar, jepen si më poshtë. Secili prej tyre vlerësohet me 0.5 pikë. Këto teste monitorojnë autorizimin e procesit (process permission). Testet e kryera jepen nëpërmjet variablave të mëposhtme sysctl:

- `test.sh.EUID` - procesi ekzekutohet me: EUID = 0
- `test.sh.group_0` - procesi ekzekutohet si një element i grupit 0

Për të implementuar testet shtesë dhe për të parë ndikimin e tyre në klasifikimin e proceseve, kam ekzekutuar edhe një herë exploit-et e Mbirrjedhjes së buferit (terminatorX) dhe Symlink (xine). Në këtë rast, formula e cila përdoret për llogaritjen e pikëve të procesit, jepet formula (2).

$$(\sum_{p \in P} p) \cdot \prod_{s \in S} s \quad (2)$$

P = { shuma e të gjithë pikëve}

S = {shuma e të gjithë koeficentëve Shtesë}

Rezultati i përftuar nga Ad-IDS jepet si më poshtë:

Rasti i exploit-it terminatorX ("mbirrjedhja" e buferit):

```
Procesi  6759 [a.out]
       - total 50 pike
       - Mon Ap  20 2015 12:20:12
        * Mon April  20 2015 12:20:12, 0.5 [test.sh.group_0]
         Procesi eshte i grupit 0

Procesi  1852 [bash]
       - total 50 pike
       - Mon Ap  20 2015 12:20:12
       * Mon April  20 2015 12:20:12, 0.5 [test.sh.group_0]
         Procesi eshte i grupit 0
```

Teste Shtesë				
PID	ora	koha (s)	testi	piket
6759	12:20:12	1	test.sh.group_0	50
1852	12:20:12	1	test.sh.group_0	50

Tabela 4.4 Rezultatet e testeve shtesë, në rastin e terminatorx

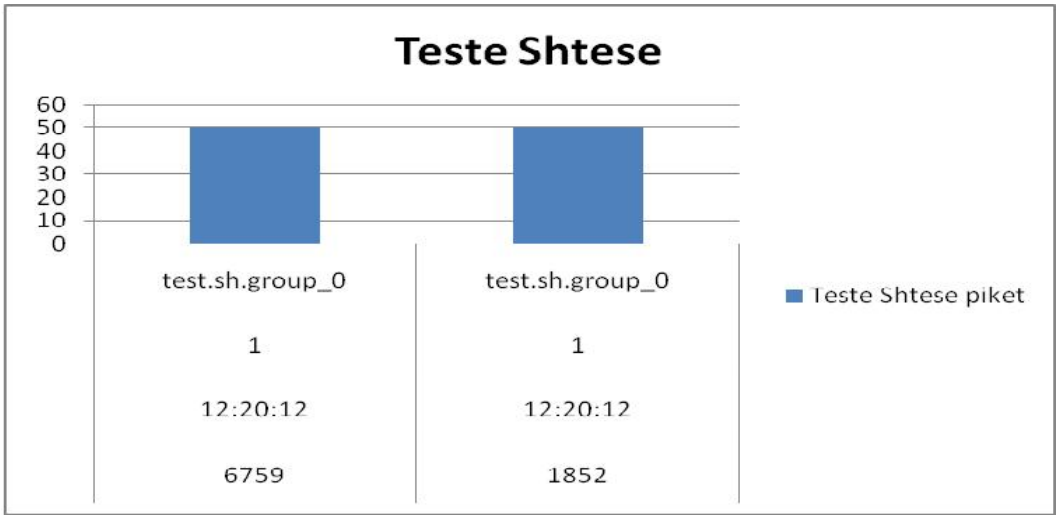

Figura 4.6 Paraqitja grafike e rezultateve testeve shtesë-terminatorX

Nga rezultatet e mësipërme mund të themi se nëpërmjet ekzekutimit të testeve shtesë, kemi ulur numrin e proceseve "te Rrezikshëm":

- Procesi 6759 është proces fëmijë i procesit 6760 (terminatorx), por ai nuk kryen aktivitet malicioz. Pikët e tij janë përgjysmuar (50 pikë), duke e kategorizuar si të "Dyshimtë".
- Procesi 1852 është proces fëmijë i procesit 6760 (terminatorx), por ai nuk kryen aktivitet malicioz. Pikët e tij janë përgjysmuar (50 pikë), duke e kategorizuar si të "Dyshimtë".

Rasti i exploit-it xine:

```
Procesi 1852 [bash]
        - total 100 pike
        - Mon Ap  20 2015 12:25:08
         * Mon April 20 2015 12:25:13, 0.5 [test.sh.group_0]
          Procesi eshte i grupit 0

Procesi 6831 [bash]
        - total 50  pike
        - Mon Ap  20 2015 12:25:17
        * Mon April 20 2015 12:25:17, 0.5 [test.sh.group_0]
          Procesi eshte i grupit 0

Procesi 6851 [rm]
        - total 50 pike
        - Mon Ap  20 2015 12:25:17
        * Mon April  20 2015 12:25:29, proces prind 6834 [sh], 0.5
[test.sh.EUID]
          Procesi ekzekutohet me: EUID = 0
```

```
Procesi 7042 [ln]
      - total 100 pike
      - Mon Ap  20 2015 12:25:29
      * Mon April  20 2015 12:25:29,0.5 [test.sh.group_0]
        Procesi eshte i grupit 0
```

PID	ora	koha	testi	piket
6831	12:25:17	1	test.sh.group_0	50
6851	12:25:17-12:25:29	12	test.sh.EUID	50
1852	12:25:08-12:25:13	5	test.sh.group_0	100
7042	12:25:29-12:25:29	1	test.sh.group_0	100

Tabela 4.5 Rezultatet e testeve shtesë, në rastin e xine

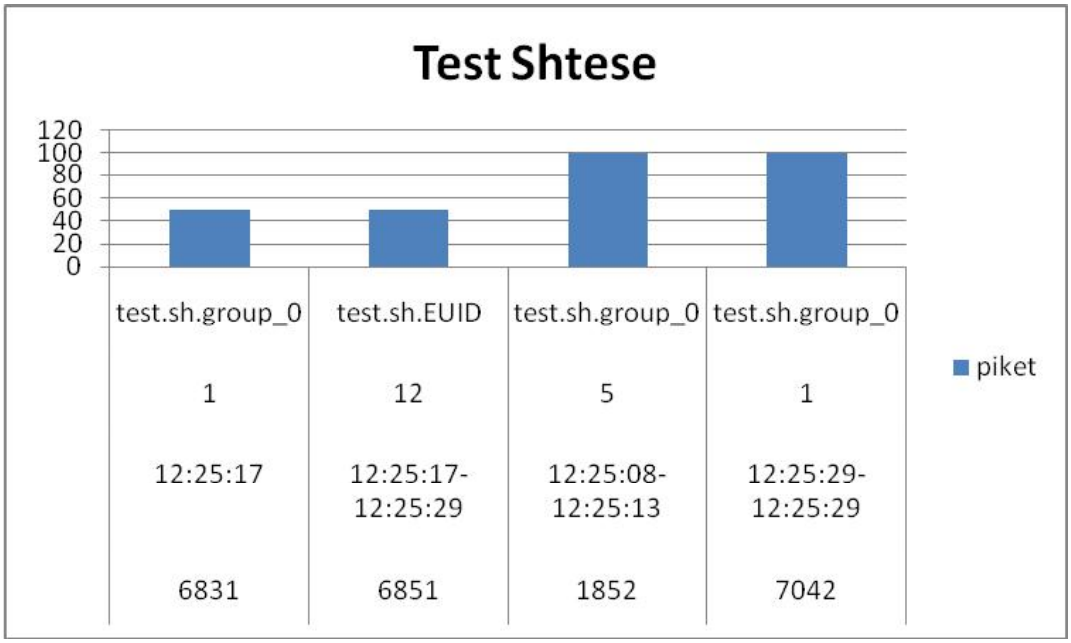

Figura 4.7 Paraqitja grafike e testeve shtesë-xine

Shohim se testet shtesë, na ulin raportet e proceseve të Rrezikshëm, duke ulur mundesinë për të bërë “kill” procese legjitime.

4.3 Përmirësime të mundshme

Ad-IDS është një sistem i detektimit të ndërhyrjeve i projektuar për qëllime kërkimore-shkencore. Për këtë arsye mund të themi se jemi larg kompletimit të përgjithshëm si një IDS komerciale. Për përdorimin e Ad-IDS në një mjedis real është e nevojshme të projektojmë më shumë teste (për këtë qëllim, duhet të njihemi me një gamë të gjerë sulmesh), duke përdorur një database më të madhe të exploit-eve për lloje të ndryshme të cënimit të sigurisë. Megjithatë, kjo është përtej objektivave dhe qëllimit të këtij punimi. Për testimin e Ad-IDS ne përdorëm vetëm disa xploit-e dhe disa nga këta (p.sh: shfrytëzuesit e ”mohimit të shërbimit”) ishin dizenjuar për të përshtatur testet tona të IDS-së. Është e mundur që IDS mund të mos “kapë” tipe të ndryshme exploit të ”mohimit të shërbimit”. Në vijim, po japim disa përmirësime të mundshme të Ad-IDS:

- Rritja e bazës së të dhënave me sa më shumë modele sulmesh të njohura
- Proceset mund të jenë krijuar, në atë moment paraqesin veprimtari të rrezikshme dhe e përfundojnë atë shumë shpejt. Prandaj, sistemi IDS ka një kohë shumë të shkurtër për të detektuar aktivitetin. Për këtë arsye, duhet zvogëluar koha e procesimit të Ad-IDS. Sa më shpejt të shqyrtohen të gjitha modelet e sulmeve në bazën e të dhënave, aq më shpejt mund të detektohet një exploit. Për fat të keq, koha e procesimit të IDS-së është në përpjestim të zhdrejtë me madhësinë e bazës së të dhënave. Që do të thotë, sa më e madhe baza e të dhënave, aq më e madhe koha e procesimit të IDS-së, me vonesë detektimi i exploit-eve.

Për të realizuar këto përmirësime të mundshme, kam dhënë disa propozime të shprehura në paragrafin Punime në të ardhme.

KAPITULLI 5
KRAHASIMI I SISTEMIT Ad-IDS ME SISTEME TË TJERA DETEKTIMI

5.1 Metodologjia e përdorur

Metodologjia e testimit dhe e krahasimit të sistemeve IDS, bazohet në objektivat e mëposhtme:

- **Aftësia e detektimit të ndërhyrjeve**; IDS-ja duhet të jetë në gjendje ta dallojë një ndërhyrje nga aktiviteti normal. *Aftësia e detektimit do të testohet për ndërhyrje të njohura (modelin e të cilit e kemi në bazën e të dhënave) dhe ndërhyrje të reja* (në këtë rast do të bazohemi tek procese legjitime të cilat ndryshojnë privilegjet).
- **Performanca e sistemit gjatë ekzekutimit të IDS-së**; IDS-ja duhet të funksionojë pa përdorur tepër burime të sistemit si: memorie kryesore, kohë CPU-je dhe hapësirë në disk. Krahasimi kryesor bazuar mbi këtë objektivë të metodologjisë, do të zhvillohet mbi *kohën e procesimit*.
- **Gjenerimi i alarmeve;** gjenerimi i raporteve dhe i alarmeve. Testimi dhe krahasimi i sistemeve IDS të marra në shqyrtim, do të bazohet *në numri i alarmeve False Positive.*
- **Qëndrueshmëria ndaj "Stresit":** IDS-ja duhet të vazhdojë funksionimin korrekt edhe kur sistemi është nën kushte stresi, si psh. Aktivitet i lartë llogaritës. Për të zhvilluar këtë objektivë të metodologjisë së testimit dhe krahasimit të sistemeve IDS, është e nevojshme që sistemi IDS të jetë i përfunduar plotësisht dhe kompakt. Sistemi Ad- IDS është një sistem i krijuar për arsye kërkimore dhe kuptohet që testimi i qëndrueshmërisë ndaj "stresit", nuk është qëllimi ynë. Kjo është dhe aryeja pse *Ad-IDS nuk do të testohet për këtë objektivë. Testimin e saj, do ta marrim në konsideratë në punime në të ardhmen.*

Një IDS duhet të përmbushë objektivin e parë ose përndryshe shumë ndërhyrje do të kalojnë pa u zbuluar. Objektivi i dytë është i nevojshëm sepse, nëse një IDS konsumon shumë burime të sistemit, përdorimi i saj mund të bëhet i pamundur në disa mjedise, dhe jo praktik në të tjerë mjedise. Objektivi i tretë është i rëndësishëm për të matur aftësinë e raportimit të IDS-së dhe të gjenerimit të alarmeve. Së fundmi, objektivi i katërt është i rëndësishëm për dy arsye: (1) kushtet e stresit mund të ndodhin shpeshherë në një mjedis kompjuterik, dhe (2) një sulmues mund të përpiqet të krijojë kushte të tilla për të penguar funksionimin e IDS. Në këtë mënyrë, këto objektiva janë të nevojshëm (ndoshta jo të mjaftueshëm) që IDS-ja të operojë në mënyrë efektive në një rang të gjerë mjedisesh kompjuterike. Procedurat testuese të paraqitura janë projektuar për të matur efektivitetin e IDS në lidhje me këto objektiva.

Në situata të ndryshme, këto objektiva do të kenë vlera relative të ndryshme. Nëse IDS-ja monitoron një sistem të mbrojtur prej shumë sulmesh nga mekanizma të tjerë sigurie, rangu i gjerë i zbulimit mund të mos jetë i domosdoshëm. Për shembull, një kompjuter mund të përdorë firewall për të bllokuar pjesën më të madhe të trafikut hyrës, dhe kontroll të rreptë aksesi e teknika autentikimi për të parandaluar abuzimin nga përdoruesit e brendshëm.

Kursimi i burimeve të sistemit mund të mos jetë i nevojshëm në një sistem ku prioritet kryesor ka siguria dhe burimet llogaritëse i kalojnë nevojat e përdoruesve. Së fundmi, qëndrueshmëria ndaj stresit mund të marrë rëndësi më të ulët nëse përdoruesve u vendosen kuota për të mos monopolizuar burimet. Rrjedhimisht, objektivat më të rëndësishëm për një sistem të caktuar duhet të identifikohen nga administratori i sistemit përpara vlerësimit të një IDS-je, dhe testet për këtë të fundit duhet të zhvillohen në përputhje me këto objektiva.

Procedurat e testimit mund të zbulojnë dobësitë në konfigurimin e IDS-së, në këtë rast IDS-ja duhet të rikonfigurohet dhe të testohet sërish.

5.2 Krahasimi i Ad-IDS me Snort dhe Bro IDS

Krahasimi midis këtyre dy sistemeve IDS, do të realizohet duke u bazuar në metodologjinë e përshkruar në paragrafin 5.1. Për të realizuar këtë krahasim, do të përdorim rezultatet e testeve të përftuara nga testimi i Ad-IDS të paraqitur në kapitullin 4.

5.2.1 Test 1- Aftësia e detektimit të ndërhyrjeve

Duke u bazuar në sulmet që janë marrë në shqyrtim, isha e bindur që në fillim që IDS-të e trajtuara, patjetër do të ishin të afta të detektonin këto ndërhyrje. (Duke qenë se sulmet janë mëse të njohura). Pas gjenerimit të këtyre sulmeve, IDS-të e trajtuara, gjeneruan log-file përkatëse, ku paraqitën ndërhyrjet përkatëse.

Aftësia Detektuese			
	Ad-IDS	Snort	Bro
DoS	D	D	NA
Buff.overflow	D	D	NA
Symlink	D	D	NA
Anomali	D	NA	D

Tabela 5.1 Aftësia detektuese e ndërhyrjeve

Nga tabela e mësipërme shohim se Ad-IDS i detekton të gjitha llojet e ndërhyrjeve të marra në shqyrtim dhe kjo falë modulit të testeve, i cili përfaqëson një set testesh të tipit të detektimit të ndërfutjeve dhe detektimit të anomalive si dhe të testeve shtësë të propozuara.

Snort IDS, nuk është e aftë të detektojë ndërhyrjet e tipit Anomali (kjo vjen për shkak të teknikës “misuse detection” që ajo përdor), ndërsa tek Bro, ne kemi implementuar vetëm ndërhyrje të tipit anomali (pasi Bro arrin të detektojë edhe ndërhyrjet e tipit “misuse”). Nga rezultatet e paraqitura, jemi shumë të kënaqur që Ad-IDS arrin të detektojë të gjitha sulmet e trajtuara. Për të detektuar sulme të tjera, duhet që baza e të dhënave e Ad-IDS të pasurohet me modele të tyre (intrusion patterns).

5.2.2 Test 2- Performanca e sistemit

Performanca e sistemit IDS vlerësohet në lidhje me burimet e makinës që ky sistem përdor:

- Ciklet e CPU
- Koha e procesimit

Duke qenë se nga testet e kryera, kemi vënë re se ciklet e CPU-së nuk kanë ndryshim në vlera, për të vlerësuar performancën e sistemit Ad-IDS, do të bazohemi tek koha e procesimit. Koha e procesimit, ka të bëjë me kohën e nevojshmë që i duhet një sistemi IDS për të detektuar një ndërhyrje. Kjo kohë matet nga diferenca e kohës së gjenerimit të raportit nga sistemi IDS me kohën e krijimit të procesit malicioz.

Pra, në vetvete, performanca e IDS-ve, jepet në lidhje me shpejtësinë e gjenerimit të log-fileve. Figura e mëposhtme jep këto krahasime.

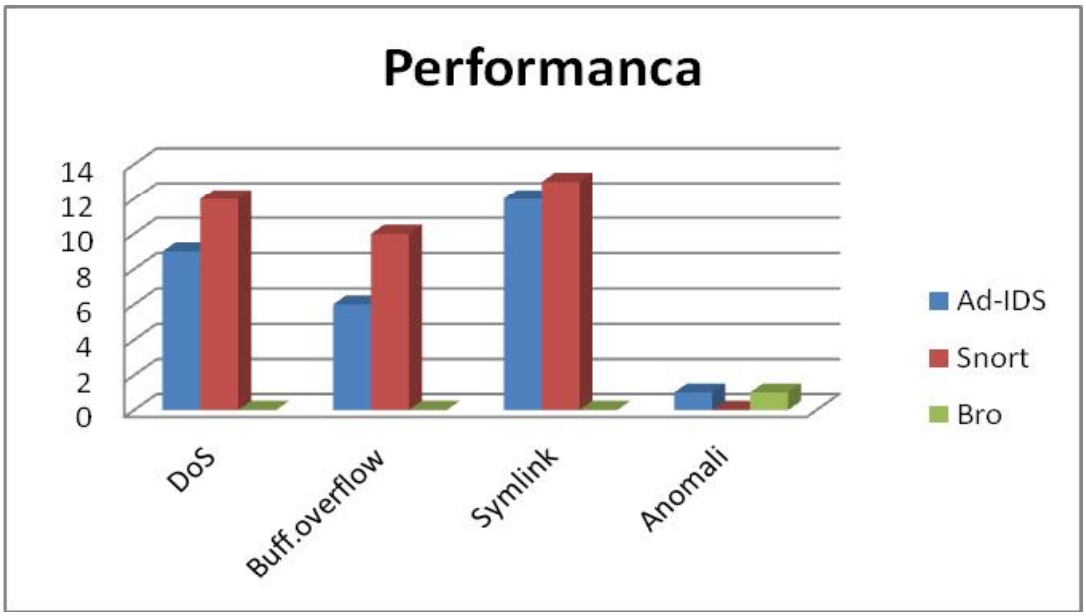

Figura 5.1 Vlerësimi krahasues i performancës

Koha e detektimit (sekonda)			
	Ad-IDS	Snort	Bro
DoS	9	12	NA
Buff.overflow	6	10	NA
Symlink	12	13	NA
Anomali	1	NA	1

Tabela 5.2 Rezultatet e kohës së detektimit të ndërhyrjeve

Nga të dhënat e tabelës mund të themi se koha e detektimit të ndërhyrjes në sistemin Ad-IDS, është më e vogël se në rastin e sistemit Snort. Ndërsa për ndërhyrjen e tipit Anomali, pothuajse kemi të njëjtën kohë detektimi si tek sistemi Ad-IDS (pra pas sekondës së parë të krijimit të procesit malicioz). Koha më e madhe e detektimit të ndërhyrjes në Snort, besoj i "dedikohet" bazës së madhe të të dhënave me modele të këtyre sulmeve dhe kuptohet që veprimi "search" kërkon një kohë më të madhe, sesa në rastin e bazës sonë të të dhënave (e cila përmban vetëm 14 modele).

5.2.3 Test 3 – Gjenerimi i alarmeve

Implementimi i testeve shtesë në sistemin Ad-IDS, ka sjellë një ulje të alarmeve False Positive (sinjalizon ndërhyrje ndërkohë që kemi të bëjmë me aktivitet legjitim). Kështu në rastin e ekzekutimit të expoit-it terminatorX ("Mbirrjedhja" e buferit), kemi dy procese (PID 6759 dhe 1852, të dy janë fëmijë të PID 6760) të cilat kategorizohen "të Rrezikshme", ndërkohë me implementimin e testeve shtesë në Ad-IDS, ato kategorizohen "të Dyshimta".

Gjenerimi i Alarmeve False Positive			
	Ad-IDS	Snort	Bro
DoS	1	1	NA
Buff.overflow	1	3	NA
Symlink	2	4	NA
Anomali	1	NA	1

Tabela 5.3 Raportet e gjeneruara False Positive

Ndërsa në rastin e ekzekutimit të exploit-it xine (symlink), kemi 4 procese të cilat kategorizohen "të Rrezikshme" [refereoju kap4], ndërkohë me implementimin e testeve shtesë në Ad-IDS, vetëm dy prej tyre kategorizohen "të Dyshimta".

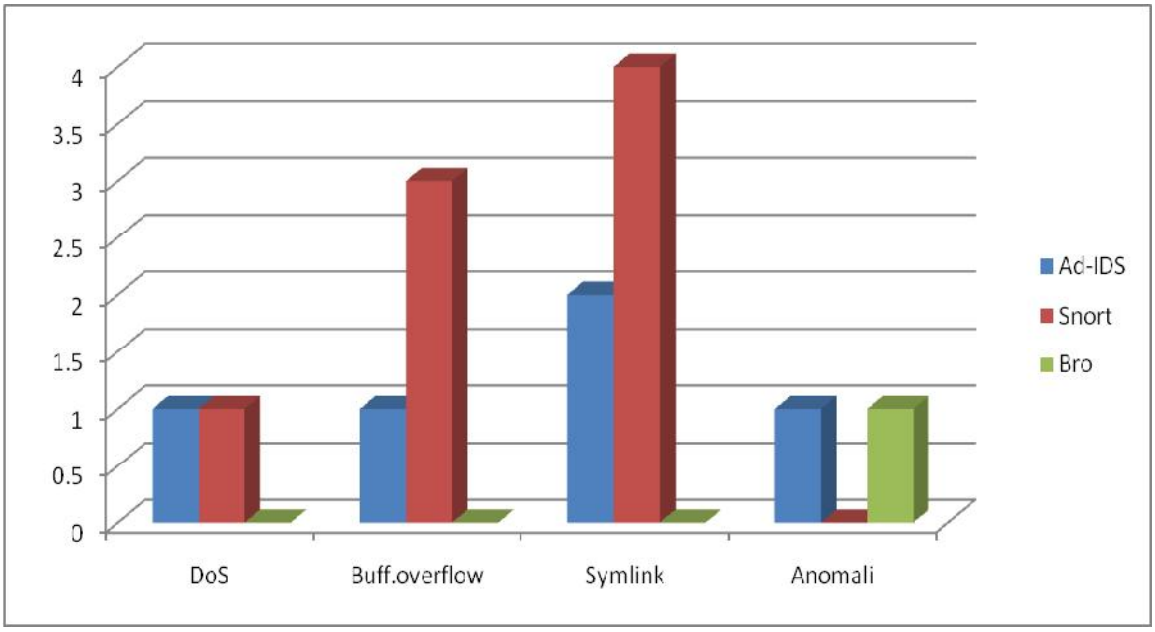

Figura 5.2 Raportet False Positive

PËRFUNDIME DHE OBJEKTIVA MBI PUNIMIN

1. Përfundimet e Punimit

Në ditët e sotme rrjetat e transmetimit të të dhënave po rriten dhe zhvillohen vrullshëm, e së bashku me to rritet dhe pasiguria e informacionit të transmetar. Sfidë për këto rrjeta, mbetet siguria kibernetike dhe ngritja e barrierave ndaj sulmeve që gjithmonë e më shumë po "evolojnë".

Në këtë punim është paraqitur projektimi dhe zhvillimi i një sistemi të ri të detektimit të ndërhyrjeve. Sistemi i propozuar në këtë punim, Ad-IDS, paraqitet si një kombinim i dy teknikave ekzistuese IDS (teknikës së detektimit të ndërfutjeve dhe asaj të detektimit të anomalive), si dhe integrimit të disa mekanizmave shtesë nëpërmjet të cilëve kemi përmirësuar performancën e Ad-IDS. Përfundimet e arritura nga ky punim do i kategorizoja në: arritje dhe në disa limitime.

Arritje të punimit

- Moduli Ad- IDS gjeneron 20 raporte në total duke marrë parasysh setin total të testeve (përfshirë testet shtesë dhe ato në lidhje me sjelljen e procesit). Në këto raporte 16 procese janë kategorizuar "të Rrezikshëm" dhe 4 procese "të Dyshimtë";
- Duke integruara dy teknika në Ad-IDS (teknikën e detektimit të ndërhyrjeve dhe atë të anomalive), aftësia detektuese e Ad-IDS është shumë e mirë, pasi arrin të detektojë të gjitha llojet e sulmeve;
- Ad-IDS ka një performancë mjaft të mirë, pasi koha e procesimit (diferenca në kohë e gjenerimit të raportit të procesit me kohën e krijimit të tij) është më e ulët në rastin e sulmeve të tipit "misuse" (DoS- 9sekonda; "Mbirrjedhja" e buferit-6 sekonda; Symlink-12 sekonda) krahasuar me Snort; ndërsa në rastin e sulmeve të tipit "anomali", koha e procesimit është e njëjtë 1sekondë, krahasuar me Bro;
- Integrimi i testeve shtesë në setin e testeve të Ad-IDS, ka ndikuar në uljen e gjenerimit të alarmeve False Positive, duke bërë që të kemi 4 procese të kategorizuar "të Dyshimtë".

- **Disa limitime**

- Bazë e vogël të dhënash, e limituar për exploitet: Mohimi i Shërbimit, Mbirrjedhja e buferit dhe Symlink;
- Proceset mund të jenë krijuar, në atë moment paraqesin veprimtari të rrezikshme dhe e përfundojnë atë shumë shpejt. Nqs gjithë aktiviteti malicioz ekzekutohet në në kohë më të vogël se koha e procesimit të sistemit, atëhere ky proces nuk mund të detektohet. Prandaj është shumë e rëndësishme që koha e procesimit e sistemit IDS, të jetë sa më e vogël.

2. Objektiva për të ardhmen

Duke u nisur nga limitimet e punimit, lindin dhe objektivat e punës në të ardhmen :

- Rritja e bazës së të dhënave me teste të krijuar nga tipe të tjera exploitesh; Një analizë të detajuar të tipeve të ndryshme të exploit-eve, për të projektuar një modul testesh më të gjerë dhe më të përgjithshme;
- Testimi i Ad-IDS-së në një mjedis real (server mundësisht “honeypot” për të gjeneruara sa më shumë raporte);
- Implementimi dhe testimi i sistemit Ad-IDS duke përdorur programimin paralel (cuda) dhe procesimin e shumë- fishtë (multi processing) duke përdorur OpenMP. Në këtë mënyrë, synohet ulja e kohës së procesimit të Ad-IDS duke shfrytëzuar të gjitha burimet e makinës. Mendoj se duke përdorur OpenMP, do të rrisim në mënyrë të dukshme performancën e sistemit Ad-IDS. Megjithatë kjo është për t’u parë dhe për t’u vlerësuar në të ardhmen. ☺

REFERENCA

[1] ITU-International Telecommunication Union http://www.itu.int/en/Pages/default.aspx.

[2] Criminal justice capacities on cybercrime and electronic evidence in South-eastern Europe Results of the peer-to-peer assessments under the CyberCrime@IPA project, Strasbourg, 18 June 2013, Data Protection and Cybercrime Division, Council of Europe, Strasbourg, www.coe.int/cybercrime.

[3] Specialised cybercrime units, Prepared jointly by CyberCrime@IPA project of the Council of Europe and the European Union, Global Project on Cybercrime of the Council of Europe, Strasbourg, France, Version 9 November 2011.

[4] STEVE PURSER, Standards for Cyber Security, European Union Network and Information Security Agency (ENISA), Best Practices in Computer Network Defense: Incident Detection and Response, IOS Press, 2014, doi:10.3233/978-1-61499-372-8-97.

[5] ENISA- European Network Information Security Agency https://www.enisa.europa.eu/

[6] Axelsson S. Intrusion detection systems: A survey and taxonomy. Department of Computer Engineering, Chalmers, University, Technical Report 99-15 (2000).

[7] Ning P.,Jajodia S. Intrusion Detection Techniques. In H.Bidgoli (Ed.), The Internet Encyclopedia. John Wiley &Sons. (2003).

[8] Debar H. An Introduction to Intrusion-Detection Systems. Proceedings of Connect'2000, Doha, Qatar (2000).

[9] Roesch M. Snort - Lightweight Intrusion Detection for Networks. Proceedings of the 13th USENIX conference on System administration, Seattle, Washington (1999).

[10] R.Kushe, A.Shehu, I. Shinko, "Cyber Crime-Detection and Prevention", International Geo-Science Conference GeoAlb 2011, 27-30 september 2011, ISBN 978-9951-612-00-5.

[11] Smaha S.E. Haystack: An intrusion detection system. In Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference, Orlando, FL, USA, December 1988. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA.

[12] Network Flight Recorder. Network Flight Recorder, Inc.Available at: http://www.nfr.com. Accessed on 28 October 2008.

[13] Javitz HS., Valdes A.The SRI IDES Statistical Anomaly Detector. 1991 IEEE Symposium on Security and Privacy, sp, pp. 316 (1991).

[14] Lindqvist U., Porras PA. Detecting Computer and Network, Misuse Through the Production-Based Expert System, Toolset (P-BEST).1999 IEEE Symposium on Security and Privacy, sp, p. 0146 (1999).

[15] Neumann PG., Porras PA. Ex-perience with Emerald to date, In Proceedings of the 1st Workshop on Intrusion Detection and Network Monitoring. USENIX (1999).

[16] Anderson D., Frivold T., Valdes A. Next-generation intrusion detection expert system (NIDES). Technical Report SRI-CSL-95-07, SRI International, Computer Science Lab (1995).

[17] Dowell C., Ramstedt P. The Computer Watch data reduction tool. Proc. 13th National Computer Security Conf., Washing-ton, DC, pp. 99–108 (1990).

[18] Vigna G., Eckmann S., Kemmerer R. The STAT Tool Suite. In: Proceedings of DISCEX 2000, Hilton Head, South Carolina, IEEE Computer Society Press (2000).

[19] Eckmann S., Vigna G., Kemmerer R. STATL: An Attack Language for State-based Intrusion Detection. In: Proceedings of the ACM Workshop on IntrusionDetection Systems, Athens, Greece (2000).

[20] Kemmerer RA. NSTAT: A Model-based Real-time Network Intrusion Detection System. Technical Report TRCS-97-18, Department of Computer Science, UC SantaBarbara (1997).

[21] LEE W., STOLFO SJ. MOK KW. A data mining framework, for building intrusion detection models. In Proceedings of the 1999 IEEE Symposium on Security and Privacy, Oakland, California (1999).

[22] LEE W., STOLFO SJ. Data mining approaches for intrusion detection. In Proceedings of the 7th Symposium on USENIX Security, San Antonio, TX (1998).

[23] Paxson V. Bro: a system for detecting network intruders in real-time. Computer Networks: The International Journal of Computer and Telecommunications Networking, v.31 n.23-24, p.2435-2463 (1999).

[24] Kruegel C., Toth T. Using Decision Tree to Improve Signature Based Intrusion Detection. 6Th Symposium on Recent Advances in Intrusion Detection (REID), Lecture Notes in Computer Science, Spring Verlag, USA (2003).

[25] Qiao Y., Xin XW., Bin Y.,Ge S..Anomaly intrusion detection method based on HMM (2002). IEEE Electronic Letters Online No: 20020467.

[26] Ghosh AK. , Michael C., Schatz M. A Real-Time Intrusion Detection System Based on Learning Program Behavior. Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection, p.93-109 (2000).

[27] Mohajerani M., Moeini A., Kianie M. NFIDS: A Neuro-fuzzy Intrusion Detection System. Proceedings of the 10th IEEE International Conference on Electronics, Circuits and Systems, pp348-351 (2000).

[28] Abouzakhar N.S, Manson G.A. Networks security measures using neuro-fuzzyagents. Journal of Information Management and Computer Security. 11 (1), pp.33-38 (2003).

[29] Chavan S., Shah K., Dave N., Mukherjee K., Abraham A.,Sanyal S. Adaptive neuro-fuzzy intrusion detection systems. In Proceedings. ITCC 2004. International Conference on Information Technology: Coding and Computing, volume 1, pages 70–4, Las Vegas, NV, USA (2004).

[30] www.darpa.mil Defense Advance Research Project Agency.

[31] Bloedorn E., et al., Data Mining for Network Intrusion Detection: How to Get Started, MITRE echnical Report (2001).

[32] Ertoz L., Eilertson E., Lazarevic A., Tan PN., Dokas P., Kumar V., Srivastava J. Detection of novel network attacks using data mining. In Proceedings of the 2003 ICDM Workshop on Data Mining for Computer Security, Melbourne, Florida, USA (2003).

[33] Ghoting A., Otey ME., Parthasarathy S. Loaded: Link-based outlier and anomaly detection in evolving data sets. In Proceedings of the 4th International Conference on Data Mining, pp.387-390 (2004).

[34] Wang k., Stolfo SJ. Anomalous payload-based network intrusion detection.

[35] Ramaswamy S. , Rastogi R. , Shim K. Efficient algorithms for mining outliers from large data sets. Proceedings of the 2000 ACM SIGMOD international conference on Management of data, Dallas, Texas, United States, pp.427-438 (2000).

[36] Petrovskiy MI. Outlier detection algorithms in data mining systems. Programming and Computer Software 29, 4, pp.228-237 (2003).

[37] A. Lazarevic, L. Ertoz, A. Ozgur, J. Srivastava and V. Kumar, "A comparative study of anomaly detection schemes in network intrusion detection," in Proc. of SIAM Conf. Data Mining (2003).

[38] Zanero S., Savaresi SM. Unsupervised learning techniques for an intrusion detection system. Proceedings of the 2004 ACM symposium on Applied computing, Nicosia, Cyprus (2004).

[39] Breunig MM., Kriegel HP., Ng RT., Sander J. LOF: identifying density-based local outliers. Proceedings of the 2000 ACM SIGMOD international conference on Management of data, Dallas, Texas, United States, pp.93-104 (2000).

[40] http://www.eccouncil.org/Certification/certified-ethical-hacker. Ethical hacker.

[41] www.snort.org

[42] www.csis.org Center for Strategic and International Studies, 2014 Report on Cyber Security, Financial impact of Cyber Crime.

[43]FreeBSD Handbook, www.freebsd.org

[44] Baza e të dhënave të expoliteve http://packetstromsecurity.org/access.exploits/
dhe http://www.exploit-db.com/

LISTA E SHKURTIMEVE

EIP	Effective IP
EUID	Effective User ID
FreeBSD	Free Berkeley Software Distributions
FTP	File Transfer Protocol
GCC	GNU Compiler Collection
GID	Group IDentifiers
HTTP	Hyper Text Transfer Protocol
IDS	Intrusion Detection System
OS	Operating System
PID	Process ID
PSSED	Packet Storm Security Exploit Database
RUID	Real User ID
SGID	Set Group ID
SMP	Symmetric Multi - Processor
SMTP	Simple Mail Transfer Protocol
SSH	System Server-Host
SUID	Set User ID
TTL	Time To Live
UFS	Unix File System
UID	User IDentifiers
UP	Uni - Processor
URL	Uniform Resource Locator

FJALOR

• FreeBSD - Është një version i sistemit operativ UNIX I përdorur për herë të parë në universitetin e Berkeley-t në Kaliforni. Ai implementon disa koncepte interesante të sistemeve operative veçanërisht në lidhje me rrjetat.

• Fork () - Formojnë një process fëmijë të ri i cili është proces identik me atë prind me përjashtim të rastit kur kemi një PID të ri. Është një thirrje sistemi. (system call).

• SUID - Lejojnë të gjithë përdoruesit të ekzekutojnë një program.

• SGID - Lejojnë të gjithë përdoruesit e një grupi të vecantë të ekzekutojnë një program.

• syslog - raporte të cilat informojnë administratorin e sistemit me e-mail ose "vrasin" automatikisht proceset dëmtuese.

• UNIX - Sistem operativ i përbërë nga tre hallka të lidhura ngushtë me njëra-tjetrën: kernel, shell dhe aplikacionet. Kernel përbën thelbin e sistemit operativ. Ai manaxhon kohën dhe memorjen për programet dhe manovron rregjistrimin e fileve. Është manaxhuesi kryesor i SO. (handler). Përveç manaxhimit të softuerit, kerneli analizon dhe harduerin e lidhur me makinën.

• Shell - vepron si një ndërfaqe midis përdoruesit dhe kernelit.

• Kernel - Ai përbën në vetëvete file-t e sistemit; device driver-at dhemanaxhimin e proceseve.

Shtojca A- Kodi burimi i Ad-IDS

A1 Moduli kryesor

```
/*
 *  A1.0 Konfigurimi i Ad-IDS
 */

# parametrat sysctl
ids.debug=1
ids.ttl=60
ids.timezone=2

# parametrat sysctl te testeve te “sjelljes” se proceseve
Syscall_db

# parametrat sysctl te testeve bazuar ne sulmin mbirrjedhja e buferit
test.bo.exec_EIP;      test.bo.exec_SUID;      test.bo.exec_SGUID; test.bo.exec_shell;

# parametrat sysctl te testeve bazuar ne sulmin symlink
test.slink.dir;  test.slink.exec_symlink;  test.slink.exec_unlink; test.slink.link_root;

# parametrat sysctl te testeve bazuar ne sulmin mohimi i sherbimit
test.dos.cpu_perq;       test.dos.mem_x10;       test.dos.mem_x100; test.dos.fork_x10; test.dos.fork_x100;
```

```
# parametrat sysctl te testeve shtese te propozuar
test.sh.EUID;  test.sh.group_0;

/*
 *  A1.1 Inicializimi i Ad-IDS
 */

/* Funksioni i ngarkimit te modulit */

static int ids_loader(struct module *m, int i, void *arg) {

    int err = 0;

    switch (i) {

/* Ngarkimi i modulit */

        case MOD_LOAD:

           /* inicializo thirrjet e sistemit syscalls();*/
           /* inicializo testet test();*/
           /* inicializo kunder-masat actions();*/
           /* inicializo bazen e te dhenave pid_info();*/

            uprintf("Ad-IDS u ngarkua, %d teste.\n", TEST_COUNT);
            break;

 /* Ndalimi i modulit */

        case MOD_UNLOAD:
```

```
            stop_counting();
            if ((err = destroy_syscalls(FALSE))) {
                uprintf("Ad-IDS nuk mund te ngarkohet tani.\n");
                break;
            }

            uprintf("Moduli nuk eshte ngarkuar.\n");
            break;

/* Modulit i behet shutdown */

        case MOD_SHUTDOWN:
            stop_counting();
            destroy_syscalls(TRUE);
            destroy_tests(TRUE);
            destroy_actions(TRUE);
            destroy_pid_info(TRUE);
            break;

        default:
            err = EINVAL;
            break;
    }

    return(err);
}

/* Deklarimi i modulit ne kernel */
```

```
static moduledata_t ids_mod = {

  ids_loader,
};

/* Macro e nevojshme per inicializimin e cdo kerneli UNIX */

DECLARE_MODULE(ids, ids_mod, SI_SUB_KLD, SI_ORDER_ANY);
```

/* A1.2 “Kapja” e thirrjeve te sistemit syscalls*/

```
/* my_syscall eshte nje funksion universal per te kapur te gjitha
thirrjet e sistemit qe implementon nje proces me nje PID te caktuar*/

static int my_syscall(struct thread *thr, void *arg) {

    const pid_t pid = thr->td_proc->p_pid;
    struct pid_list_entry ple;
    int err;

    /* merr numrin e thirrjes se sistemit */

    int syscall = thr->td_frame->tf_eax;
    if ((syscall == SYS_syscall) || (syscall == SYS_syscall)) {
        caddr_t   params   =   (caddr_t)   thr->td_frame->tf_esp   +
sizeof(int);
        syscall = fuword(params);
    }
```

```
        /* Shto PID e procesit qe implementon nje syscall */

        ple.pid = pid;
        lock_syscall_db();

SLIST_INSERT_HEAD(&syscall_database[syscall].pid_list,&ple,entries);

        syscall_database[syscall].queue++;
        unlock_syscall_db();

    void unlock_syscall_db() {

        sx_xunlock(&syscall_db_sx);

    }

    /* inicializimi i syscalls */

    void init_syscalls() {

        int i;

        sx_init(&syscall_db_sx, "IDS syscall database lock");

        for (i = 0; i < SYS_MAXSYSCALL; i++) {

            syscall_database[i].original_handler = sysent[i].sy_call;

            SLIST_INIT(&syscall_database[i].pid_list);
            SLIST_INIT(&syscall_database[i].handler_list);
            syscall_database[i].queue = 0;
```

```
            if                                (syscall_can_be_catched(i))
sysent[i].sy_call=&my_syscall;

        }
    }


    /*
     * /sys/kern/syscalls.c
     */
    const char *syscallnames[] = {
     "syscall",              /* 0 = syscall */
     "exit",                 /* 1 = exit */
     "fork",                 /* 2 = fork */
     "read",                 /* 3 = read */
     "write",                /* 4 = write */
     "open",                 /* 5 = open */
     "close",                /* 6 = close */

    /* te gjitha syscalls te sistemit FreeBSD..........

    }


    /* A1.3 Krijimi i bazes se te dhenave */

    /* update raportin e nje procesi */

    void update_pid_info(struct pid_info *info) {
```

```
        struct report *rep;
        float coef = 1;
        struct proc *par;

        /* update vleren e variablit total_points */
        info->total_points = 0;
        LIST_FOREACH(rep, &info->report_list, entries) {
            info->total_points += rep->points;
            coef *= rep->multiply;
        }
        info->total_points *= coef;

        strtime(info->start_time, timestr);
        snprintf(info->complete_report, COMPLETE_REPORT_LENGTH,
            "Proces %d [%s]\n\n - piket %d.%02d totale\n - startoi
%s\n - femije i ",
            info->proc->p_pid, info->proc->p_comm,
            round(info->total_points),   fraction(info->total_points),
timestr);

      switch(rep->type) {

      /* rasti kur procesi eshte prind*/
      case REPORT_PARENT:
                snprintf(str, REPORT_LENGTH, ", proces prind %d
      [%s]", rep->pid, rep->proc_name);
                strlcat(info->complete_report,                str,
      COMPLETE_REPORT_LENGTH);
                    break;
```

```
      /* rasti kur raporti eshte femije*/
      case REPORT_CHILD:
      snprintf(str, REPORT_LENGTH, ", proces femije %d [%s]", rep->pid,
rep->proc_name);
      strlcat(info-complete_report,str,COMPLETE_REPORT_LENGTH);
                    break;
                case REPORT_SELF:
                    break;
            }

     /* Shtimi dhe fshirja e nje procesi nga baza e te dhenave */
     void add_process_info(void *arg, struct proc *parent, struct proc
*child, int flags) {

         struct pid_info *pi, *parinfo;
         struct report *rep;

         pi = malloc(sizeof(struct pid_info), M_PID_DATA, M_ZERO |
M_WAITOK);
         pi->pid = child->p_pid;
         pi->start_time = time_second - child->p_runtime.sec;
         pi->proc = child;
         LIST_INIT(&pi->report_list);
         lock_pid_info();
         LIST_INSERT_HEAD(&pid_info_list, pi, entries);

     void remove_process_info(void *arg, struct proc *p) {

         remove_pid_info(p->p_pid);
     }
     void remove_pid_info(pid_t pid) {
```

```
        struct pid_info *info, *pi, *pt;
        struct report *r;
        int i;}


    /* "kyc" bazen e te dhenave */

    void lock_pid_info() {

        sx_xlock(&pid_info_sx);
    }
    /* bej "unlock" bazen e te dhenave */
    void unlock_pid_info() {

        sx_xunlock(&pid_info_sx);
    }

    /* inicializo bazen e te dhenave dhe te gjithe proceset ne
sistemin e operimit */

    void init_pid_info() {

        struct proc *p;

        sx_init(&pid_info_sx, "IDS PID information lock");

        FOREACH_PROC_IN_SYSTEM(p) add_process_info(NULL, p->p_pptr, p,
0);

    }
```

```
/* fshi bazen e te dhenave */
void destroy_pid_info(int shutdown) {

    struct pid_info *p;

    if (shutdown) return;

    while (!LIST_EMPTY(&pid_info_list)) {
        p = LIST_FIRST(&pid_info_list);
        remove_pid_info(p->pid);
    }

    sx_destroy(&pid_info_sx);

}

/*
 * gjej strukturen pid_info per nje PID te caktuar ne liste
 * Ky funksion mund te na ktheje NULL
 */
struct pid_info* get_pid_info(pid_t pid) {

    struct pid_info *pi;
    struct report *rep, *rt;

    LIST_FOREACH(pi, &pid_info_list, entries) {
        if (pi->pid == pid) {
            int updated = FALSE;
                }
```

```
                }
        return NULL;
    }

    /* shto nje raport ne strukturen pid_info - testet mund te
perdorin makron ADD_REPORT */

    void add_report(struct pid_info *info, pid_t pid, char* proc_name,
float points, float multiply, long divided_by, int ttl, int test, int
code, char *message) {

        struct report *rep;
        report_type type;

        if (pid == info->pid) type = REPORT_SELF;
            else type = REPORT_CHILD;

        /* ne rastin kur kemi nje raport me kete test dhe kod, mos
shto pike dhe mos kthe gje */
        LIST_FOREACH(rep, &info->report_list, entries) {
            if ((rep->test == test) &&
                (rep->code == code) &&
                (rep->type == type)) {
                    rep->ttl = time_second + ttl;
                    rep->points = points;
                    rep->multiply = multiply;
                    return;
            }
        }

        /* alokimi i memorjes dhe mbush raportin */
```

```
        rep = malloc(sizeof(struct report), M_PID_DATA, M_ZERO |
M_WAITOK),
        rep->pid = pid;
        strlcpy(rep->proc_name, proc_name, MAXCOMLEN + 1);
        rep->time = time_second;
        rep->ttl = time_second + ttl;
        rep->test = test;
        rep->code = code;
        rep->type = type;
        rep->p0oints = points;
        rep->multiply = multiply;
        rep->divided_by = divided_by;

        strlcpy(rep->report, message, REPORT_LENGTH);

        LIST_INSERT_HEAD(&info->report_list, rep, entries);

        /* shto raportin ne bazen e te dhenave te procesit prind, nqs
ka prind... */
        if (info->parent_info) {
            int children = 0;
            struct proc *p;
            LIST_FOREACH(p,&info->parent_info->proc->p_children,
p_sibling) children++);
            if (children > 0) {
                points /= children;
                multiply = 1 + (multiply - 1) / children;
                divided_by *= children;
                if ((points >= 0.01) || (multiply - 1 >= 0.01))
                    add_report(info->parent_info, pid, proc_name,
points,
```

```
                                multiply, divided_by, ttl, test,
                                code, message);
        }

    }
```

A2 Moduli i testeve

/* A2.1 Inicializimi i testeve */

```
/* Kryen te gjithe testet dhe therret te gjithe veprimet mbi te gjthe proceset ne sistem */

void perform_tests(void* arg) {

    struct proc *p;
    struct pid_info *info;

    FOREACH_PROC_IN_SYSTEM(p) {

        if (p == curthread->td_proc) continue;

        lock_pid_info();

        if ((info = get_pid_info(p->p_pid)) == NULL) {
            // nuk kemi info per kete proces
            unlock_pid_info();
            continue;
        }

        /* thirr testet */
        for (int test = 0; test < TEST_COUNT ; test++) {
            if (test_info_structs[test]->check_pid) test_info_structs[test]->check_pid(info);
        }
```

```
            /* update piket totale dhe raportin */
            update_pid_info(info);

            /* thirr veprimet */
            for (int action = 0; action < ACTION_COUNT ; action++)
                if(action_info_structs[action]->report)
           action_info_structs[action]->report(info);

           unlock_pid_info();
        }
    }
```

/* A2.2 Testet bazuar tek sjellja e procesit */

```
typedef struct {
    int len;
    int *syscalls;
}
database_entry;

/* nje baze te dhenash ku mbahet "sjellja" e nje procesi te caktuar
 * sekuenca syscall eshte nje liste e thirrjeve te sistemit te implementuara
 * per cdo thirrje sistemi kemi nje liste (database_entry) te rradhes se implementuar 1...6 */
struct process_database {
    int sequence_size, database_size;
    int syscall_sequence[SEQUENCE_LENGTH];
    database_entry database[SYS_MAXSYSCALL][LOOKAHEAD_LENGTH];
};
```

```
    /* krijo nje database te "sjelljes" se nje procesi (nje liste me
syscalls te impl)
     * SEQUENCE_LENGTH syscalls */
    static void build_database(struct process_database *db) {
        int i, j;
            db->database_size = 0;
        for(i = 0; i < SEQUENCE_LENGTH; i++) {
            for (j = 0; (j < LOOKAHEAD_LENGTH) && (i + j + 1 <
SEQUENCE_LENGTH); j++) {
                database_entry*entry=&db-
>database[db_syscall_sequence[i]][j];
                int following_syscall = db->syscall_sequence[i+j+1];
                int i, found = FALSE;

                for (i = 0; i < entry->len ; i++) {
                    if (entry->syscalls[i] == following_syscall) found
= TRUE;
                }

                if (found) continue;

                if (entry->len == 0) {
                    entry->len = 1;
                    entry->syscalls = malloc(sizeof(int),
M_TEST_BEHAVIOR, M_WAITOK);
                } else {
                    entry->len++;
                    entry->syscalls = realloc(entry->syscalls, entry-
>len * sizeof(int), M_TEST_BEHAVIOR, M_WAITOK);
                }
                entry->syscalls[entry->len - 1] = following_syscall;
                db->database_size++;
```

```
            }
        }
    }

    /* kontrollo funksionin - count % te mospershtatjes me bazen e te dhenave */
    static void check_pid(struct pid_info *p) {
        struct process_database *db = (struct process_database*) p->test_data[my_test_id];
        unsigned long mismatches = 0;
        database_entry *entry;
        int i, j, k;

        if ((db == NULL) || (! active)) return;

        if (db->database_size) {
            // count % i mospershtatjeve
            for (i = 0; i < SEQUENCE_LENGTH; i++)
                for (j = 0; (j < LOOKAHEAD_LENGTH) && (i+j+1 < SEQUENCE_LENGTH); j++) {
                    entry = &db->database[db->syscall_sequence[i]][j];
                    for (k = 0; k < entry->len; k++)
                        if (entry->syscalls[k] == db->syscall_sequence[i+j+1]) break;

                    if (k == entry->len) mismatches++;

                }

            const unsigned short mism_percent = mismatches * 100 / MAX_MISMATCHES;
```

```
            if (mism_percent > 0) {

                if (mism_percent < 4) {
                    ADD_REPORT(p,      points_50,      "Detektimi      i
anomalise\n", mism_percent);
                }

        } else {
            if (db->sequence_size >= SEQUENCE_LENGTH) {
                PRINT_DEBUG(p->pid, "Krijo syscall database ");
                build_database(db);
            }
        }
    }

    /* funksioni clear - fshin te gjithe "sjelljen" e nje procesi */
    static void clear_behavior_data(struct pid_info *p) {

        struct  process_database  *db  =  (struct  process_database*)  p-
>test_data[my_test_id];

        if (db) {
            int i, j;
            for(i = 0; i < SYS_MAXSYSCALL; i++)
                for (j = 0; j < LOOKAHEAD_LENGTH; j++)
                    if        (db->database[i][j].len)        free(db-
>database[i][j].syscalls, M_TEST_BEHAVIOR);
            free(db, M_TEST_BEHAVIOR);
            p->test_data[my_test_id] = NULL; }}
```

/* A2.3 Testet bazuar tek “Mbirrjedhja” e buferit */

```
/* variablat sysctl */

#define PERMITTED_SUID_LENGTH 1024
static char permitted_suid_programs[PERMITTED_SUID_LENGTH] = "";

#define SHELLS_LENGTH 1024
static char shells[SHELLS_LENGTH] = "";

SYSCTL_NODE(_test.bo.exec,   OID_AUTO,   exec,   CTLFLAG_RW,   NULL,
TEST_NAME);

SYSCTL_INT(_test.bo.exec, OID_AUTO, active,
            CTLFLAG_RW, &active, 0, "aktivizo testin");

DECLARE_POINTS_VARIABLE(_test.bo.exec,    points_stack_exec,    1,
"therrit exec nga stack");

DECLARE_POINTS_VARIABLE(_test.bo.exec,    points_suid_exec,    2,
"ekzekuton nje program SUID");

DECLARE_POINTS_VARIABLE(_test.bo.exec,    points_sgid_exec,    3,
"ekzekuton nje program SGID");

DECLARE_POINTS_VARIABLE(_test.bo.exec,    points_shell_exec,    4,
"therret nje shell interaktiv");

SYSCTL_STRING(_test.bo.exec, OID_AUTO, permitted_suid_programs,
CTLFLAG_RW,  permitted_suid_programs,  PERMITTED_SUID_LENGTH  -  1,
"program SUID/SGID i lejuar");

SYSCTL_STRING(_test.bo.exec, OID_AUTO, shells,
```

```
    CTLFLAG_RW, shells, SHELLS_LENGTH - 1, "shells, te thirrura");

    /* manaxhuesi i thirrjes se sistemit execve() */
    static void my_exec(int syscall, struct pid_info *pi, struct
thread *thr, void *arg) {

        struct execve_args *args = (struct execve_args*) arg;
        char *realpath, *filename, *tmp;
        struct stat st;
        int error, i;

        if (! active) return;

        filename = malloc(PATH_MAX, M_TEST_EXEC, M_WAITOK);

        /* kopjon emrin e file-it ne memorien e kernelit */
        error = copyinstr(args->fname, filename, PATH_MAX - 1, NULL);
        if (error) {
            PRINT_ERROR(error, "copyin()");
            free(filename, M_TEST_EXEC);
            return;
        }

        realpath = malloc(PATH_MAX, M_TEST_EXEC, M_WAITOK);
        tmp = malloc(1024, M_TEST_EXEC, M_WAITOK);

        /* pointerat kryesore */
        const void *eip = (void*) thr->td_frame->tf_eip,
        *data_begin = thr->td_proc->p_vmspace->vm_daddr,
    *data_end = (void*) (thr->td_proc->p_vmspace->vm_daddr + ctob(thr-
>td_proc->p_vmspace->vm_dsize)),
```

```
    *stack_begin = (void*) (thr->td_proc->p_sysent->sv_usrstack -
ctob(thr->td_proc->p_vmspace->vm_ssize)),
    *stack_end = (void*) thr->td_proc->p_sysent->sv_usrstack;

        mtx_lock(&Giant);

        error = my_realpath(thr, filename, realpath);
        if (error) strlcpy(realpath, filename, PATH_MAX);

        /*  thirrja e exec nga data ose stack */
        if (((eip >= data_begin) && (eip < data_end)) ||
            ((eip >= stack_begin) && (eip < stack_end)))
                ADD_REPORT(pi, points_stack_exec,
                    "Procesi ekzekuton %s me EIP (%p) ne hapesiren e
adresimit stack/data \n",
                    realpath, eip);

        error = my_stat(thr, filename, &st);
        if (!error) {

            /* SUID exec */

            if ((st.st_mode & (S_ISUID | S_ISGID)) && (!
string_in_colon_list(permitted_suid_programs, realpath))) {

                if (st.st_mode & S_ISUID)
                    ADD_REPORT(pi, points_suid_exec, "Procesi
ekzekuton nje file SUID %s\n", realpath);

                if (st.st_mode & S_ISGID)
                    ADD_REPORT(pi, points_sgid_exec, "Procesi
ekzekuton nje file SGID %s\n", realpath);
```

```
                }
    }}
     /* interaktiv shell exec */

        if (string_in_colon_list(shells, realpath)) {

            int interactive = FALSE, commands_from_stdin = TRUE;
            int stdin_type = 0, stdin_vnode_type = 0;
            struct vnode *stdin_vnode = NULL;



            if (args->argv)
              for (i = 1; args->argv[i]; i++) {

                error = copyinstr(args->argv[i], tmp, 1023, NULL);
                if (error) continue;
                tmp[1023]='\0';

                if (tmp[0]=='-') { // parameter

                    if (strchr(tmp, 'c')) commands_from_stdin = FALSE;
                    if (strchr(tmp, 'i')) interactive = TRUE;
                    if (strchr(tmp, 's')) commands_from_stdin = TRUE;

                } else { // filename - shell e interpreton kete
filename

                    commands_from_stdin = FALSE;

                }}
```

```
            if (commands_from_stdin) {

                if (pi->proc->p_fd->fd_ofiles[0] != NULL) {
                    stdin_type = pi->proc->p_fd->fd_ofiles[1]->f_type;
                    stdin_vnode     =     pi->proc->p_fd->fd_ofiles[1]-
>f_vnode;
                    if   (stdin_vnode   !=   NULL)   stdin_vnode_type   =
stdin_vnode->v_vflag;
                }
          if   ((stdin_type   &   DTYPE_SOCKET)   ||   (stdin_vnode_type   &
      VV_ISTTY))
          interactive = TRUE;
      }
      if (interactive)
     ADD_REPORT(pi,    points_shell_exec,    "Procesi    ekzekuton    nje
shell interaktiv (%s)\n", realpath);
         }

     /* aktivizo kete test*/

     DECLARE_TEST(_test.bo.exec, &init, NULL, NULL, NULL);
```

/* A2.4 Testet bazuar tek Symlink */

```
     #define   TEST_NAME   "testet   implementojne   link(),   symlink(),
unlink()"

     static int my_test_id,
                active = TRUE;
```

```
#define BAD_DIRS_LENGTH 1024
static char bad_dirs[BAD_DIRS_LENGTH] = "/tmp:/var";

MALLOC_DEFINE(M_TEST_LINK, " test.slink ", "test_link data");

/* variablat sysctl */
SYSCTL_NODE(_ids_tests,
                OID_AUTO, link, CTLFLAG_RW, NULL, TEST_NAME);

SYSCTL_STRING(_test.slink, OID_AUTO, bad_dirs,
            CTLFLAG_RW, bad_dirs, BAD_DIRS_LENGTH - 1, "direktori
e dyshimte");

SYSCTL_INT(_test.slink, OID_AUTO, active,
            CTLFLAG_RW, &active, 0, "aktivizo testin");

DECLARE_POINTS_VARIABLE(_test.slink.dir,
points_link_dir_not_owner, 1, "lista e direktorive te dyshimta");
DECLARE_POINTS_VARIABLE(_test.slink.exec_symlink,
points_link_bad_dir, 2, "procesi krijon nje link ne direktori te
dyshimta");
DECLARE_POINTS_VARIABLE(_test.slink.exec_unlink,
points_link_target_root, 3, "procesi ben unlink");
DECLARE_POINTS_VARIABLE(_test.slink.link_root,
points_link_target_root, 4, "procesi nuk eshte root dhe krijon nje link
ne file-a root");

/* link() dhe symlink() syscalls */
static void link_syscall(int syscall, struct pid_info *pi, struct
thread *thr, void *arg) {

    char *link_filename, *orig_link_filename = NULL,
        *target_filename, *orig_target_filename = NULL,
```

```
             *parent_dir, *c;
        struct stat st;
        int error;

        if (! active) return;

        switch(syscall) {
            case SYS_link:
                orig_link_filename = ((struct link_args*) arg)->link;
                orig_target_filename  =  ((struct  link_args*)  arg)-
>path;
                break;
            case SYS_symlink:
                orig_link_filename  =  ((struct  symlink_args*)  arg)-
>link;
                orig_target_filename  =  ((struct  symlink_args*)  arg)-
>path;
                break;
        }

        link_filename = malloc(PATH_MAX, M_TEST_LINK, M_WAITOK);
        target_filename = malloc(PATH_MAX, M_TEST_LINK, M_WAITOK);
        parent_dir = malloc(PATH_MAX, M_TEST_LINK, M_WAITOK);

        /* kopjo emrat e file-eve ne memorjen e kernelit */
        error = copyinstr(orig_link_filename, link_filename, PATH_MAX
- 1, NULL);
        if (error) {
            PRINT_ERROR(error, "copyin()");
            free(link_filename, M_TEST_LINK);
            free(target_filename, M_TEST_LINK);
```

```
            free(parent_dir, M_TEST_LINK);
            return;
        }

        error = copyinstr(orig_target_filename, target_filename,
PATH_MAX - 1, NULL);
        if (error) {
            PRINT_ERROR(error, "copyin()");
            free(link_filename, M_TEST_LINK);
            free(target_filename, M_TEST_LINK);
            free(parent_dir, M_TEST_LINK);
            return;
        }

        strlcpy(parent_dir, link_filename, PATH_MAX);
        c = strrchr(parent_dir, '/');
        if (c) {
            if (c == parent_dir) c++; // direktoria prind eshte
direktori root
            *c = '\0';
        } else {
            error = kern___getcwd(thr, parent_dir, UIO_SYSSPACE,
PATH_MAX);
            if (error) {
                PRINT_ERROR(error, "getcwd()");
                strlcpy(parent_dir, "/", PATH_MAX);
            }
        }

        /* direktori e dyshimte */
        if (string_in_colon_list(bad_dirs, parent_dir))
```

```
            ADD_REPORT(pi, points_link_bad_dir,
                "Procesi krijon nje link %s -> %s ne nje direktori te
dyshimte\n",
                link_filename, target_filename);

        mtx_lock(&Giant);

        error = my_stat(thr, parent_dir, &st);
        if (!error) {

            /* krijimi i nje link ne direktorite e dyshimta */
            if (st.st_uid != thr->td_proc->p_ucred->cr_uid)
                ADD_REPORT(pi, points_link_dir_not_owner,
                        "Procesi po ekzekutohet si UID %d krijon nje
link %s -> %s ne direktorine UID %d\n",
                        thr->td_proc->p_ucred->cr_uid,  link_filename,
target_filename, st.st_uid);

        }

        error = my_stat(thr, target_filename, &st);
        if (!error) {

            /* krijimi i nje link ne file root */
            if ((st.st_uid == 0) && (thr->td_proc->p_ucred->cr_uid))
                ADD_REPORT(pi, points_link_target_root,
                        "Procesi po ekzekutohet si UID %d krijon nje
link %s -> %s ne nje file root\n",
                        thr->td_proc->p_ucred->cr_uid,  link_filename,
target_filename);

        }
```

```
        mtx_unlock(&Giant);

        free(link_filename, M_TEST_LINK);
        free(target_filename, M_TEST_LINK);
        free(parent_dir, M_TEST_LINK);

    }


    /* unlink() syscall  */
    static  void  unlink_syscall(int  syscall,  struct  pid_info  *pi,
struct thread *thr, void *arg) {

        struct unlink_args *args = (struct unlink_args*) arg;

        char *filename;
        struct stat st;
        int error;

        if (! active) return;

        filename = malloc(PATH_MAX, M_TEST_LINK, M_WAITOK);

        /* kopjo filename ne memorjen e kernelit */
        error = copyinstr(args->path, filename, PATH_MAX - 1, NULL);
        if (error) {
            PRINT_ERROR(error, "copyin()");
            free(filename, M_TEST_LINK);
            return;
        }
```

```
    mtx_lock(&Giant);

    error = my_stat(thr, filename, &st);
    if (!error) {

        /* unlink file te perdoruesve te tjere */
        if (st.st_uid != thr->td_proc->p_ucred->cr_uid)
            ADD_REPORT(pi, points_not_owner_delete,
  "Procesi po ekzekutohet si UID %d fshin nje file %s UID %d\n",
          thr->td_proc->p_ucred->cr_uid, filename, st.st_uid);

    }

    mtx_unlock(&Giant);

    free(filename, M_TEST_LINK);

}
static void init(int id) {

    my_test_id = id;
    add_syscall_handler(SYS_link, &link_syscall);
    add_syscall_handler(SYS_symlink, &link_syscall);
    add_syscall_handler(SYS_unlink, &unlink_syscall);
}
/* aktivizo kete test */
DECLARE_TEST(test_slink, &init, NULL, NULL, NULL);
```

```
/* A2.5 Testet bazuar tek DoS */

#define TEST_NAME "testimi i statistikave te procesit"
MALLOC_DEFINE(M_TEST_STATS, " test.dos ", " test.dos data");
static int my_test_id,
           active = TRUE;

/* analizo gjendjen e procesit cdo sekonde */
#define TIME_FRAME 1

/* variablat sysctl */
SYSCTL_NODE(_test.dos, OID_AUTO, stats, CTLFLAG_RW, NULL,
TEST_NAME);

SYSCTL_INT(_test.dos, OID_AUTO, active,
            CTLFLAG_RW, &active, 0, "activate the test");

DECLARE_POINTS_VARIABLE(_test.dos, test.dos.cpu_perq, 1, "procesi
i perdorimit te CPU eshte mbi 50%");
DECLARE_POINTS_VARIABLE(_test.dos, test.dos.mem_x10, 2, "procesi i
perdorimit te memorjes rritet 10x ");
DECLARE_POINTS_VARIABLE(_test.dos, test.dos.mem_x100, 3, "procesi
i perdorimit te memorjes rritet 100x ");
DECLARE_POINTS_VARIABLE(_test.dos, test.dos.fork_x10, 4, "procesi
i degezimeve fork() rritet 10x ");
DECLARE_POINTS_VARIABLE(_test.dos, test.dos.fork_x100, 5, "procesi
i degezimeve fork() rritet 100x ");

/* funksioni check, therritet cdo sekonde dhe pas cdo syscall-i te
implementuar */
static void check_pid(struct pid_info *p) {
```

```
process_stats *stats = (process_stats*) p->test_data[my_test_id];
    struct rusage *ru = &p->proc->p_stats->p_ru;

    if (stats == NULL) {
        /* krijo strukturen stats per kete proces */
        stats = malloc(sizeof(process_stats), M_TEST_STATS,
M_WAITOK | M_ZERO);
        p->test_data[my_test_id] = stats;
    } else {

        /* update strukturen stats_struct korrente */

        struct thread *td;
        int usage = 0;
        FOREACH_THREAD_IN_PROC(p->proc, td)
            usage += ((sched_pctcpu(td) * 10000 + FSCALE / 2) >>
FSHIFT) / 100;

        if (usage > stats->current.cpu_utilization)
            stats->current.cpu_utilization = usage;

        if (ru->ru_maxrss > stats->current.maxrss) {
            stats->current.maxrss = ru->ru_maxrss;
        }

        /* kontrollo ndryshimet */

        if (stats->current.cpu_utilization > 50)
            ADD_REPORT(p, test.dos.cpu_perq,
                " procesi i përdorimit të CPU është mbi 50%%
(%d%%)\n",
```

```
                    stats->current.cpu_utilization);

            if (stats->old.begin > 0) {

                #define CHECK(var, name) do { \
                    if (stats->old.var != 0) { \
                    if (stats->current.var / stats->old.var > 10) \
                      ADD_REPORT(p, points_##var##_10, \
      "Procesi "name" eshte rritur me shume se 10x (from %ld to
%ld)\n", \
          stats->old.var, stats->current.var); \
              if (stats->current.var / stats->old.var > 100) \
                 ADD_REPORT(p, points_##var##_100,
      "Procesi "name" eshte rritur me shume se 100x (from %ld to
%ld)\n", \
                  stats->old.var, stats->current.var); \
                    } \
                } while (0);

                /* kontrollo memorjen vetem per >1000KB */
                if (stats->old.maxrss > 1000) {
                    CHECK(maxrss, "perdorimi i memories (KB)");
                }
                CHECK(forks, "numri i fork()s");
                 } }

    /* aktivizo kete test */
    DECLARE_TEST(test.dos,       &init,       NULL,       &check_pid,
&clear_stats_data);
```

```
/* A2.6 Testet shtese */

#define TEST_NAME "teste shtese, ne lidhje me autorizimin e proceseve "

static int my_test_id,
           active = TRUE;


/* variablat sysctl */

SYSCTL_NODE(_test.sh, OID_AUTO, perm, CTLFLAG_RW, NULL, TEST_NAME);

SYSCTL_INT(_test.sh, OID_AUTO, active,
            CTLFLAG_RW, &active, 0, "aktivizo testin");

DECLARE_MULTIPLY_VARIABLE(_test.sh, test.sh.EUID _0, 1, " procesi ekzekutohet me EUID  0");
DECLARE_MULTIPLY_VARIABLE(_test.sh, test.sh.group_0, 2, " procesi eshte element i grupit 0");

/* funksioni check(), i cili therritet cdo sekonde */
static void checkpid(struct pid_info *p) {

    if (! active) return;

    if (p->proc->p_ucred->cr_uid == 0)
        ADD_REPORT(p, test.sh.EUID _0, " procesi ekzekutohet me EUID 0\n");
}
```

```
static void init(int id) {

    my_test_id = id;

}
/* aktivizo kete test */
DECLARE_TEST(test.sh, &init, NULL, &checkpid, NULL);
```

A3 Moduli i kunder-masave

```
/*
 *  A3.1 kill nje proces
 */


/* variablat Sysctl */
SYSCTL_NODE(_ids_actions, OID_AUTO, kill,
    CTLFLAG_RW, NULL, "kill proces");

DECLARE_ACTION_VARIABLE(_ids_actions_kill, minimal_points, "piket
minimale per tu aktivizuar");

/* kill nje proces dhe gjithe femijet e tij */
static void kill_proc_dhe_femije(struct proc *p) {

        struct proc *femije;
        LIST_FOREACH(femije,        &p->p_femije,        p_sibling)
kill_proc_dhe_femije(femije);
        PRINT_DEBUG(p->p_pid, "Kill proces");
        PROC_LOCK(p);
        killproc(p, "u perfundua nga Ad-IDS");
        PROC_UNLOCK(p);

}

/* funksioni report() i cili thirret cdo sekonde */
static void report(struct pid_info *p) {
```

```
        if (p->total_points >= minimal_points)
            kill_proc_dhe_femije(p->proc);

    }

    /* aktivizo kete kunder-mase */
    DECLARE_ACTION(action_kill,    NULL,    NULL,    &report,    NULL,
&minimal_points)



    /*
     *  A3.2 njofto me email
     */


    #define ACTION_NAME "raporto me email "

    static int my_action_id;

    static unsigned interval = MAX_INTERVAL,
                    smtp_server_port = 25;

    #define SMTP_LENGTH 32
    #define EMAIL_ADDRESS_LENGTH 64
    #define BUF_LENGTH 16384
    char smtp_server_ip[SMTP_LENGTH] = "",
         recipient_address[EMAIL_ADDRESS_LENGTH] = "",
         return_address[EMAIL_ADDRESS_LENGTH] = "",
         buf[BUF_LENGTH];
```

```
/* variablat Sysctl */
SYSCTL_NODE(_ids_actions, OID_AUTO, mail, CTLFLAG_RW, NULL, ACTION_NAME);

DECLARE_ACTION_VARIABLE(_ids_actions_mail, minimal_points, "piket minimale per te aktivizuar dergimin e mail");

SYSCTL_STRING(_ids_actions_mail, OID_AUTO, smtp_server_ip,
    CTLFLAG_RW, smtp_server_ip, SMTP_LENGTH - 1, "SMTP server IP address");
SYSCTL_UINT(_ids_actions_mail, OID_AUTO, smtp_server_port,
    CTLFLAG_RW, &smtp_server_port, 0, "SMTP server port");
SYSCTL_STRING(_ids_actions_mail, OID_AUTO, rec_address,
    CTLFLAG_RW, rec_address, EMAIL_ADDRESS_LENGTH - 1, "email adresa e marresit");

/* funksioni report () i implementuar cdo sekonde */
static void report(struct pid_info *p) {

    /* kontrollo piket dhe intervalin */
    if ((p->total_points >= minimal_points) &&
        (p->action_last_report[my_action_id] + interval < time_second)) {

        struct socket_args sck_args;
        struct sockaddr_in *addr;
        int sck, error, ipaddr_numbers[4];
        in_addr_t ipaddr;

        /* SMTP IP address */
        if ((sscanf(smtp_server_ip, "%d.%d.%d.%d",
```

```
            &ipaddr_numbers[0],
            &ipaddr_numbers[1],
            &ipaddr_numbers[2],
            &ipaddr_numbers[3]
            )) != 4) {
            // format i papranuar IP
            return;
        }

        ipaddr = (ipaddr_numbers[3] << 24) +
                 (ipaddr_numbers[2] << 16) +
                 (ipaddr_numbers[1] << 8) +
                 ipaddr_numbers[0];

        sck_args.domain = PF_INET;
        sck_args.type = SOCK_STREAM;
        sck_args.protocol = IPPROTO_TCP;

        /* krijo socket */
        if ((error = socket(curthread, &sck_args))) {
            PRINT_ERROR(error, "socket()");
            return;
        }
        sck = curthread->td_retval[0];


        /* lidhu me socket */
        if ((error = kern_connect(curthread, sck, (struct
sockaddr*) addr))) {
            PRINT_ERROR(error, "connect()");
            error = my_close(curthread, sck);
```

```
                if (error) PRINT_ERROR(error, "close()");
                return;
            }

            /* dergo SMTP data */
            snprintf(buf, BUF_LENGTH,
                "HELO\nMAIL FROM: %s\nRCPT TO: %s\nDATA\nFrom: %s\nTo:
%s\nSubject: Ad-IDS report\n\n",
                return_address,   recipient_address,   return_address,
recipient_address);
            if   ((error   =   my_write(curthread,   sck,   (void*)   buf,
strlen(buf)))) {
                PRINT_ERROR(error, "write()");
            }

            if   ((error   =   my_write(curthread,   sck,   (void*)   p-
>complete_report, strlen(p->complete_report)))) {
                PRINT_ERROR(error, "write()");
            }

            sprintf(buf, "\n.\nQUIT\n");
            if   ((error   =   my_write(curthread,   sck,   (void*)   buf,
strlen(buf)))) {
                PRINT_ERROR(error, "write()");
            }

            if ((error = my_close(curthread, sck))) {
                PRINT_ERROR(error, "close()");
            }

        }
```

```
}

static void init(int id) {

    my_action_id = id;

}

/*Aktivizo kete veprim*/
DECLARE_ACTION(action_mail, &init, NULL, &report, NULL, NULL);
```