RUNTIME INTRUSION DETECTION SYSTEM

Daniel Ortega Berja

Resum– Motivat per la contínua evolució en tècniques d'atac a dispositius electrònics i per la sofisticació que arriben a aconseguir, aquesta investigació es basa en aconseguir informar de les modificacions indesitjades que un atacant pugui fer a la part més important d'un sistema operatiu, el nucli. Entre els objectius del treball es buscarà que aquest codi de detecció no tingui un impacte gran en el rendiment de l'equip, adaptant-se a les càrregues existents i que sigui aplicable al mercat de les impressores d'HP. Es tindran en compte les últimes mesures de protecció com pot ser l'aïllament de taules de pàgina del nucli i aprofitarà l'arquitectura de les taules de pàgina per detectar qualsevol intent de modificació en el seu espai de memòria, situant aquest projecte entre els pocs que intenten mitigar les vulnerabilitats del nucli en dispositius que integrin una versió de Linux actualitzada.

Paraules clau– Amenaces, arquitectura de computadors, ciberseguretat, Linux .

Abstract– Motivated by the continuous evolution in electronic device attack techniques and the sophistication they achieve, this research is based on being able to report unwanted modifications that an attacker can make to the most important part of an operating system, the kernel. Among the objectives of the project will aim that this detection code does not have a large impact on the performance of the equipment, adapting to existing workloads and that it is applicable to the market of HP printers. The latest protection measures such as the kernel page table isolation will be taken into account and the architecture of the page tables will be used to detect any attempt to modify their memory space, placing this project among the few which attempt to mitigate kernel vulnerabilities on devices that integrate an updated version of Linux.

Keywords– Threats, computer architecture, cybersecurity, Linux .

1 INTRODUCCIÓ

A Mesura que els anys passen, la tecnologia és més avançada i, per tant, complexa, els sistemes de seguretat també han anat millorant posant més difícil als ciberdelinqüents atacar un sistema. És per això que les tècniques d'atac a sistemes es fan més sofisticades i es busquen noves maneres de defensa enfocades tant en la prevenció com en la detecció dels possibles atacs. Per una banda els casos més habituals són quan es fa ús de cadenes d'eines ja conegudes per atacar empreses que no han actualitzat o protegit els seus equips, però també es poden trobar casos com són les vulnerabilitats de dia zero que són noves tècniques de les quals no existeix cap manera de protegir-se, com al seu moment foren Meltdown i Spectre [1] que van ser descobertes després de molts anys sent vulnerables a elles, aquestes fan que els experts en seguretat hagin d'adaptar-se, detectar i prevenir a temps per estalviar danys inesperats. En tots aquests casos, l'objectiu de l'atacant pot ser o fer malbé els sistemes o bé obtenir permisos especials dins d'ell per fer operacions no permeses.

Un projecte relacionat amb la detecció d'intrusions fa endinsar-te de ple en el món de la ciberseguretat, creant la necessitat d'ampliar els coneixements que es poden tenir sobre l'arquitectura d'un computador, ja que per aplicar mesures de protecció a un sistema primer s'ha de conèixer a la perfecció les parts implicades del Sistema Operatiu i entendre les tecnologies que s'han d'aplicar per aconseguir un sistema de detecció de modificacions robust.

1.1 Què és el nucli d'un Sistema Operatiu

Qualsevol dels dispositius utilitzats avui en dia porta un sistema operatiu (SO), des de telèfons intel·ligents passant pels dispositius IoT fins a ordinadors basen les seves aplicacions i el seu funcionament segons el SO que utilitzin. Aquest sistema operatiu està compost per diferents compo-

• E-mail de contacte: danielortegaberja@gmail.com
• Menció realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Angel Elbaz (DEIC)
• Curs 2020/2021

nents que el formen, però serà el nucli (en anglès Kernel) la part principal d'ell i on es recolzen moltes de les funcionalitats bàsiques d'un ordinador. Per l'usuari això serà una capa totalment transparent, on només podrà percebre la part gràfica de tot el conjunt de components que conformen la lògica d'un ordinador, en canvi, per un atacant poder modificar el contingut d'aquest nucli és un punt clau on poder obtenir el control sobre tot el que es fa a l'ordinador, recordem que en ser una part essencial i a priori immutable del Sistema Operatiu es confia en el seu correcte funcionament, té la total capacitat en permisos de modificació d'altres mòduls i cap altra part del sistema té accés per modificar-lo a ell.

Els anells de protecció serveixen perquè el nucli pugui distingir els privilegis d'on provenen les crides al sistema, d'aquesta manera si s'intenta accedir des d'una posició no privilegiada a un recurs privilegiat, es nega l'accés o bé es produeix un error. A la gran majoria de nuclis i sobretot a Linux se sol utilitzar ring 0 i ring 3 on es diferencia de l'espai del nucli i l'espai d'usuari, permetent des de la capa interior accedir a les exteriors però no a l'inrevés. S'estableix així una capa de seguretat que evita l'escalament de privilegis dels atacants on els accessos amb privilegis ring 1 i 2 poden interaccionar directament amb el hardware, però només l'anell interior pot modificar les funcions del software més crítiques [2].

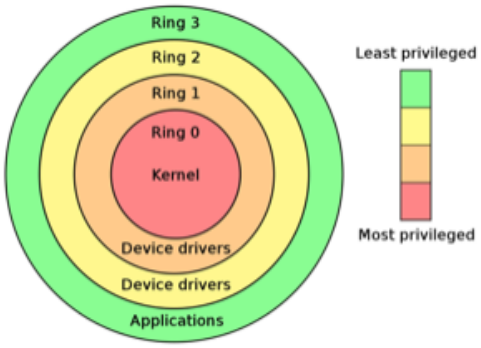

Fig. 1: Nivells d'anells de protecció a Linux

1.2 Què són els hipervisors i la virtualització

Aquest projecte basarà el seu funcionament en l'aprofitament dels beneficis de la virtualització, amb ell s'aconsegueix desacoblar les mesures de protecció del sistema que es vol defensar, d'aquesta manera en cas que l'equip contingui una vulnerabilitat explotable es redueixen notablement les maneres de saltar-se les mesures de detecció que s'apliquin.

El concepte de virtualització es basa a crear una capa de software que permeti a programes o fins i tot a sistemes operatius executar-se concurrentment, de manera que queden aïllats entre ells però dins d'una mateixa màquina. Aquesta nova capa actua d'interfície entre les altres, emulant o adaptant el comportament que tindrien les crides de capes superiors a inferiors (instruccions màquina, crides al sistema, llibreries...), és a dir, tenim diferents conceptes de virtualització segons la capa on s'apliquen. L'hipervisor ens farà d'orquestrador de totes les instruccions que s'han de dur a terme per la virtualització i ens ajudarà a visualitzar i com aplicar-lo, gràcies a ell podrem situar-nos en diferents escenaris com per exemple, com a capa intermèdia entre el Sistema Operatiu i el hardware o bé per tenir múltiples instàncies d'aplicacions o adaptacions de llibreries. Aprofitarem la capacitat dels hipervisors de poder desacoblar el hardware i el sistema operatiu, ja que permeten tenir diferents instàncies del sistema o també anomenades màquines virtuals, aquest concepte apareix de tenir diferents màquines virtualitzades aïllades entre elles però orquestrades pel nucli de l'hipervisor. Al nucli de l'hipervisor l'anomenarem monitor de màquines virtuals o per les sigles angleses VMM (Virtual Machine Monitor).

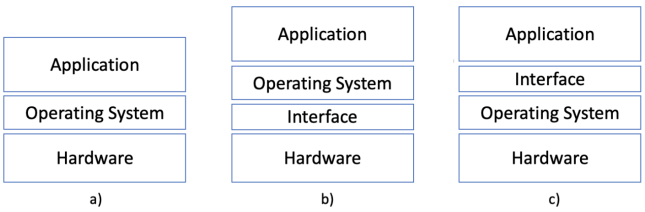

Fig. 2: Diferents aplicacions de la virtualització en l'arquitectura d'un computador

A la figura 2 podem veure com es poden trobar possibles composicions en l'ús dels hipervisor per l'arquitectura del sistema que vulguem utilitzar, en l'escenari a) trobaríem totes les capes estàndard com poden ser la de maquinari (hardware en anglès), sistema operatiu (Operating System) i capa d'aplicació (Application). Els hipervisors més coneguts són les versions d'escritori de VMWare, VBox o Parallels i és situarien a l'escenari c), s'anomenen hipervisors de tipus dos i es situen per sobre del Sistema Operatiu hoste. En concret, aquest projecte es basarà en el concepte b) de la figura 2 on gràcies a la capa que fa d'interfície hardware, podrem tenir múltiples sistemes operatius i aplicacions aïllades entre ells i sobre el mateix maquinari, a aquest tipus d'interfície se'ls coneix com hipervisors de tipus 1 (o Bare-Metal Hypervisors) alguns exemples poden ser Bareflank, VMWare vSphere i Xen.

A la imatge s'ha nombrat a l'hipervisor com a Interface, ja que una de les tasques es facilitar la comunicació entre capes gairebé com si no existís intermediari. Quan es fa ús dels hipervisors bare-metal el que busquem és crear una nova capa dins de l'anell de permissos esmentat anteriorment, elevant totes les altres capes, d'aquesta manera l'hipervisor es situaria a la capa de l'anell ring -1 i tindria accés a totes les capes superiors. A més, el SO natiu pensarà que té tot el control i es comunica directament amb el hardware.

1.3 Què és un sistema d'intrusions

Els sistemes de detecció d'intrusions es focalitzen en l'anàlisi de comportaments anòmals del sistema i en el posterior avís a un administrador. D'aquesta manera, mitjançant la monitorització de diferents variables es poden descobrir possibles accessos no controlats a un sistema, detectant així quan un intrús faci modificacions del sistema. Segons les mètriques triades un sistema de detecció serà:

- Network-Based: Monitoritza tot el tràfic anòmal de xarxa d'entrada i sortida. Snort és un sistema de prevenció d'intrusions ofert per Cisco i analitza el tràfic

en temps real i els missatges dels paquets de la axarxa IP [3].

- Host-Based: Analitza diferents esdeveniments dins d'un mateix hoste. Ossec és open-source i fa un anàlisi dels missatges (logs en anglès) dels fitxers, a més, d'altres sistemes de deteccions propis [4].

En aquest projecte es farà un servir un sistema de detecció de tipus Host-Based, per tant, s'haurà d'elaborar un programa que monitoritzi el sistema hoste. Els tipus de detecció d'intrusions generals solen ser basats en signatures, en estadístiques de comportament (ús de Machine Learning), o estats preguardats del sistema per comparar-los amb els estats actuals.

És molt important l'entorn on s'executarà aquest programa, i una de les finalitats és aconseguir que les mesures de protecció aplicades estiguin allotjades separades del sistema a protegir, gràcies a l'ús dels bare-metal hipervisors podrem muntar un escenari com el següent:

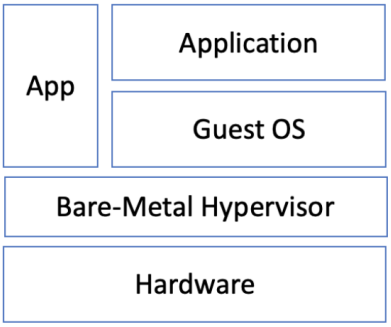

Fig. 3: Ús dels hipervisors de tipus 1 (o Bare-Metal) per allotjar el nostre sistema de detecció

Així doncs, aconseguirem executar codi i suportar sistemes operatius en espais de memòria diferents, aconseguint una protecció fiable i inalterable separada dels mateixos tipus d'atacs dels quals es vol protegir. A la figura 3 podem veure com a l'espai App anirà el codi de detecció i a GuestOS i Application el sistema original que es vol protegir.

2 Objectius

L'objectiu del projecte serà monitoritzar el nucli de Linux en temps d'execució per detectar modificacions no desitjades, per això s'utilitzarà una tecnologia de virtualització per poder crear una capa per sota del SO de manera que el codi que s'hi trobi estigui aïllat i protegit. S'espera que en un futur pròxim aquest disseny sigui aplicable en una impressora i s'han de tenir en compte les restriccions que tindrem de memòria i processament, per tant el codi ha de tenir un bon rendiment i haurà d'adaptar-se a la càrrega del sistema si s'escau i fer menys comprovacions en moments que l'ús dels recursos del sistema és elevat. Separant per punts els objectius del projecte:

- Entendre l'arquitectura de tecnologies de virtualització i hipervisors.
- Entendre el funcionament de cada mòdul del nucli de Linux.
- Modificar i estendre un monitor de màquines virtuals de codi obert.
- Implementar un codi de detecció i monitorització.
- Aconseguir que el codi detecti en temps d'execució.
- Observar l'impacte en rendiment que tingui el nostre codi a l'equip instal·lat.
- Adaptar el codi a uns nivells de rendiments acceptables segons el sistema.

També es contempla opcionalment altres objectius que es poden anar integrant més endavant durant el projecte o bé en un futur, aquests poden ser:

- Implementar un segon mètode de detecció de modificacions.
- Adaptar el codi als requisits hardware específics d'un model d'impressora.
- Observar nous impactes de rendiment després de noves implementacions.
- Adaptar el nou codi a nivells de rendiment acceptables segons la càrrega del sistema, creant noves mètriques específiques per les impressores.

3 Metodologia

Durant l'execució d'aquest projecte s'han adaptat diferents fases segons les necessitats tècniques de cada part de la implementació, en moments del projecte on la dificultat era major, el balanç entre fases ha pogut variar, però en general sempre han seguit la mateixa estructura.

En primer lloc la cerca d'informació ha estat organitzada de dues maneres, cada cop que una tasca era planificada es reservava un interval de temps suficientment ample perquè aquesta tasca deixés madurar els conceptes que es buscaven, a més, un cop trobats s'havien de redactar en esborrany abans de la següent etapa. En segon lloc es posava en pràctica els conceptes apressos per avançar en l'objectiu principal del projecte, permetent modificacions i nous plantejaments, però sempre deixant versions estables que vagin avançant per no perdre el fil del projecte. Finalment, podem dir que a escala de projecte i per fer reunions es feia servir una metodologia basa en SCRUM personalitzada, fent l'estructura esmentada anteriorment en una setmana de duració, així les funcionalitats anaven integrant-se en el projecte i era sempre funcional.

Altres metodologies emprades per poder seguir un bon ritme de treball, han estat fer servir tècniques de gestió del temps com temporitzadors Pomodoro on els intervals de treballs eren de 50 minuts i els descans de 10. I finalment esmentar l'ús d'eines de control de versions per desenvolupament software com GitHub, això ha servit per poder tenir la seguretat de no patir imprevistos i perdre parts del projecte, també el fet de tenir una branca principal i una altra per fer proves ha agilitzat molt per veure la diferència entre versions, l'últim avantatge que ens ha atorgat l'entorn Git és poder donar ullades al codi sense necessitat de muntar tot l'entorn de desenvolupament.

4 ESTAT DE L'ART

En iniciar aquest projecte, es van buscar d'altres que oferissin una solució a la detecció de modificacions del nucli del SO, alguns daten sobre el 2007 i adrecen el problema utilitzant deteccions específiques de certes parts del kernel mitjançant resums MD5 o SHA1, altres implementacions del sector es focalitzen més en el tipus de detecció Network-Based on controlen esdeveniments de la xarxa. Aquestes solucions no solen ser de codi obert ni estan ben documentades, i totes plegades, són molt antigues i no s'han actualitzat als canvis que el nucli de Linux i les arquitectures dels processadors han patit, per tant, de totes les trobades no se n'ha vist cap que solucioni el problema amb els mètodes que trobarem en aquest informe.

L'únic projecte que pot servir de referència és la solució proposada pel projecte Numchecker de BlackHat [4] que adreça el mateix problema de la detecció d'amenaces al nucli de Linux i proposa una solució molt interessant, fa ús d'un hipervisor de tipus 1 per monitoritzar les crides al sistema i els esdeveniments hardware que es fan entre el SO i el hardware, aprofita l'ús dels HPCs (High Performance Counters) per poder fer una anàlisi de patrons anòmals a unes crides determinades, en resum, si es detecta un major ús d'instruccions per fer una crida al sistema coneguda, pot ser un indici que aquesta ha estat modificada.

En definitiva, trobem que l'estat de l'art d'aquest projecte és escàs a causa de la dificultat tècnica d'aconseguir un bon sistema de detecció de modificacions basat en el comportament de l'hoste i que integri les últimes mesures de protecció del nucli de Linux.

5 ELECCIÓ D'UN HIPERVISOR

Tot i saber que uns dels requisits per poder dur a terme el projecte amb l'empresa es que el sistema sigui adaptable a una impressora, l'únic aspecte tècnic indispensable perquè aquest projecte sigui interoperable entre distribucions Linux és que la versió del nucli de Linux sigui igual o superior a la 4.15 aspecte del qual es parlarà més tard. Com la distribució de Linux instal·lada a les impressores és privada de l'empresa fins no tenir una primera versió del projecte no es tindria accés a poder fer proves amb ella, conseqüentment, la tria del SO ha estat Ubuntu en versió Long Term Support (LTS) 20.04.1 que incorpora les mateixes característiques i és ideal per fer proves.

En la tria d'un hipervisor de tipus 1 els projectes amb més comunitat i projectes darrere són Xen, vSphere, però l'interès de l'empresa per implicar-se en un nou projecte d'hipervisor Open-Source anomenat Bareflank [6] va fer que aquest fos l'agent on es codificaria el nostre projecte. Aquest ofereix un software development kit (de l'anglès, SDK) que proveeix una interacció més fàcil i un llançament de màquines virtuals més àgil que la competència. Bareflank, a partir d'ara anomenat amb el terme general hipervisor, permet editar totes les parts d'aquest així com estendre'l amb mòduls funcionals extres.

Altres requisits que s'han de tenir en compte és el hardware sobre el qual estiguem fent les proves, de fet segons el model de CPU en el que ens trobem estarà habilitat o no la possibilitat de virtualitzar sistemes operatius. L'arquitectura objectiu del projecte és Intel x86 de la que s'aprofundirà més endavant i s'ha de mirar que dins de les opcions de configuració del processador que estigui habilitat Intel VT-X aquesta tecnologia és la que permet que l'abstracció del hardware que permet a múltiples fluxos de treball compartir recursos sigui compatible amb la virtualització, segons els processadors poden o no tenir aquesta funcionalitat activada per defecte, això es pot comprovar amb el model de CPU al lloc oficial a [7], l'utilitzat al projecte serà Intel Core i5 6267U.

6 ESTRUCTURA DEL PROJECTE

Recollint tots els objectius del projecte, i seguint les recomanacions en el disseny del software, s'ha definit una arquitectura del programari que es desenvoluparà en aquest projecte, s'ha dividit en mòduls funcionals i persegueixen els 4 grans blocs per aconseguir fer un sistema de deteccions d'intrusions en temps real. Per poder millorar la llegibilitat del codi, seguir bones pràctiques de codificació i poder reutilitzar parts del codi i no repetir-lo, es va decidir separar bé les funcions principals del projecte i alhora aprofitar per veure que es volia fer a cada part, d'aquesta manera poden dividir-se també com a Milestones. El projecte esquematitzat queda definit a l'apèndix 1.

En ell es poden trobar totes les parts que componen aquest projecte, el primer mòdul correspon a l'accés de les pàgines del nucli mitjançant l'estructura jeràrquica de paginació i del registre CR3; el segon defineix quina part s'emmagatzemarà dels resultats de l'anàlisi i on es guardaran; en tercer lloc es comparen resultats de diferents anàlisis i com es detectaran els possibles canvis per un atacant i finalment com funciona el mòdul d'execució en temps real, especificant la mètrica utilitzada i el control del codi a la càrrega.

7 ACCÈS A LES PÀGINES DEL KERNEL

Després de la seqüència d'inici del sistema, el nucli Linux és el primer a ser iniciat i per atendre a totes les tasques del sistema es llencen processos concurrents d'inici del SO, aquests demanen recursos ja sigui de connectivitat, de còmput, memòria, etc. Especifiquem doncs, que el nucli és el codi encarregat d'orquestrar el consum de recursos del hardware que demanen els processos de les aplicacions. Ja sigui d'un sistema operatiu Windows, Mac o ara bé de qualsevol distribució basada en Linux el nucli funciona com a interfície i base entre els esmentats processos i el hardware del sistema encarregant-se de 5 grans tasques:

- Gestionar la memòria: La mida de memòria de l'ordinador és immensa i s'estableixen polítiques per poder accedir a diferents rangs de memòria de la manera que més rendiment s'obtingui. Per evitar haver de fer cerques innecessàries i disposar de més espai de memòria en els processos es fa ús de la memòria virtual i és gestionada pel gestor de memòria del nucli, més endavant s'explicarà amb més detall l'espai virtual de memòria.

- Gestionar els processos: S'estableixen mitjançant planificador de tasques, quins processos poden fer ús dels recursos de la CPU i per quant temps, també estableix com es comuniquen els processos entre ells (senyals, pipes ...).

- Controladors de dispositius: Actua com a mediador amb cada perifèric connectat al sistema, estableix totes les operacions de control sobre els dispositius que s'està utilitzant.
- Xarxa: Entrada i sortida de paquets, esdeveniments asíncrons, i en general totes les operacions de la xarxa han de ser classificats i identificats abans que un procés els utilitzi. El sistema és l'encarregat del control de la resolució d'adreces i de distribuir la informació per les interfícies de xarxa.
- Crides al sistema: Eines utilitzades per les aplicacions per comunicar-se amb el sistema operatiu.
- També estableix el sistema de fitxers on defineix l'estàndard necessari segons les necessitats del sistema, comprèn una gran varietat de formats, per tant, portar un sistema de fitxer a Linux és molt més fàcil que a altres nuclis.

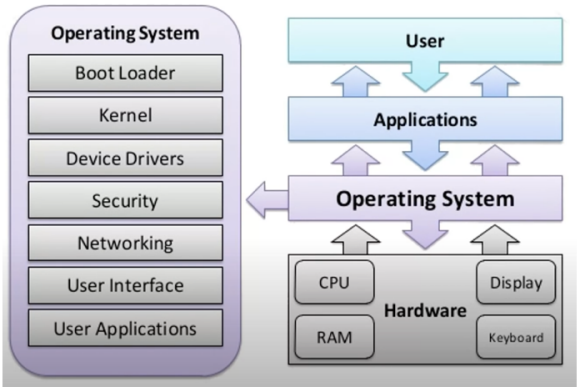

Fig. 4: Arquitectura d'un computador i desglossament del sistema operatiu

Com s'ha esmentat amb anterioritat, la intenció és fer un sistema d'intrusions de tipus Host-Based, és a dir, que s'analitzaran diferents esdeveniments dins del mateix hoste, per tant, no mirarem la part del nucli corresponent a la xarxa i es pot plantejar quins dels altres aspectes del nucli es poden monitoritzar: memòria, processos, controladors o crides al sistema. La importància de protegir el codi essencial i inalterable del sistema és màxima, cada aplicació farà ús de qualsevol de les 5 grans tasques del nucli i, per tant, qualsevol alteració en aquell codi pot tenir un impacte crític en el funcionament del nostre sistema. S'enfoca, doncs, el sistema de detecció de modificació en la comprovació de l'existència de només codi legítim al nucli, per això en concret s'investigarà en la gestió de memòria que fa el nucli i dels permisos que ha de tenir.

7.1 Sistema de paginació

Quan s'executa un programa, no es fa ús de direccions físiques de memòria, sinó que per poder adreçar tota la memòria d'una manera més òptima es fa ús direccions lògiques establertes en els sistemes per especificar adreces d'operacions o instruccions, aquestes tenen un format segmentat i segons la seva longitud i tipus poden interpretarse de maneres diferents. En comú tenen que cada conjunt de l'adreça identifica un nivell de segment de dades i un desplaçament que es fa a partir del segment esmentat. La seva notació pot trobar-se en hexadecimal de 32 bits o 64 bits.

- Adreça Lineal de 32 bits: 0x00002000
- Adreça Lineal de 64 bits: 0xFFFF800000000000

L'arquitectura del nostre sistema és Intel x86 [8], es duu a terme la traducció d'adreces lineals en estructures jeràrquiques de paginació a través del registre CR3. Tot i disposar de 64 bits a les adreces, aquesta arquitectura només en fa ús de 48 a 52 bits de cada direcció, amb 48 bits adrecem 256 TBytes d'espai d'adreces lineals i amb 52 es podria arribar a 4 PBytes, per tant es veu com no és necessari fer ús de tot l'espai comprès pels 64 bits.

L'estructura jeràrquica esmentada anteriorment i observada a l'apèndix 2, es separa en 4 nivells: Page Map Level 4 (PML4), Page Directory Pointer Table (PDPT), Page Directory Table (PDT) i Page Table (PT) on es troben totes les entrades que apunten a les següents taules. Segons la mida de la pàgina de destinació la distribució de bits per cada nivell de paginació és diferent. Per trobar la pàgina d'una direcció lògica en concret se segueix el següent procés:

- Obtenció PML4: Per poder accedir a la direcció lineal de la taula de nivell 4 s'agafen els bits 51:12 del registre CR3, seguidament es treu l'entrada concreta de la taula amb els bits 47:39 de l'adreça lineal a analitzar, finalment se sumen els dos últims bits amb valor 0.
- Obtenció PDPT: Seguidament agafem els bits 51:12 de la direcció que es troba a l'entrada de la PML4 (PML4e), concatenem els bits 38:30 de la direcció lineal i dos zeros.
- Obtenció PT: Per obtenir l'entrada del Page Directory agafem també els bits 51:12 de l'adreça física que es troba a l'entrada, els traduïm a lògica per poder sumar l'índex d'entrada corresponent als bits 29:21 de la direcció lineal i afegim dos bits a 0 al final.

A l'apèndix 3 es pot observar la distribució de bits per cada adreça segons el nivell, a més de l'espai corresponent als bits 51:12 on es troba l'adreça del següent nivell, també trobem entre el bit 1 i 8 alguns dels flags que analitzarem per trobar possibles modificacions en el codi. Tot i que parlarem més endavant de quins d'aquests analitzem els bits més comuns són present, lectura/escriptura, pwt, pcd, accessed, dirty, pat i global.

És important esmentar que també s'ha descartat l'opció d'utilitzar la funcionalitat de taules de pàgina esteses (EPT), aquesta pràctica utilitzada en la tecnologia de virtualització és útil per poder acomodar múltiples màquines virtuals i que el monitor d'aquestes pugui gestionar la paginació de l'arquitectura x86 correctament, com en el nostre cas només tenim un sistema hoste, que ens interessa tenir un espai d'adreces d'1:1 (1 adreça lògica de l'hoste per 1 de la màquina virtualitzada) i que aquesta mesura introdueix força sobrecàrrega al sistema, s'ha decidit no fer-la servir.

7.2 Mapes de memòria i registres

Un dels requisits del projecte es trobava al voltant de la necessitat de complir amb la versió del nucli de Linux superior a 4.15, aquestes versions apareixen després del descobriment de les vulnerabilitats Meltdown i Spectre del 2018 que afectaven a totes les CPUs que integraven arquitectura x86. Aquest atac permetia llegir i executar codi del nucli arribant al màxim de permisos que es pot obtenir, ja que tant codi del nucli com el codi de l'usuari es trobava al mateix espai d'adreces. Per poder mitigar aquest problema a la versió 4.15 es va integrar l'aïllament de pàgines de taules del nucli (en anglès kernel page table isolation) creant dos escenaris amb una taula amb fortes restriccions d'accés a codi del nucli i una altra que té accés a tot el conjunt d'espai d'adreces, nucli i usuari, així quan es necessita executar o llegir codi del nucli es fa un canvi de taula de manera puntual.

Aquesta mesura afecta el plantejament inicial del projecte, es volia seguir l'exemple d'anteriors projectes on es podia accedir a determinades regions del mapa de memòria i trobar allà el codi del nucli, aquestes regions predefinides dins d'un espai de memòria formen un mapa de memòria. Ara es presenta l'escenari on, tot i patir les noves mesures de seguretat emprades, s'intentarà accedir d'igual manera a les taules de pàgina del nucli. Val a dir que tot i separar mode privilegiat i mode usuari a les taules, dins del mode usuari és conserven encara certs apuntadors al codi del nucli que encara poden posar en risc l'aleatorietat del sistema.

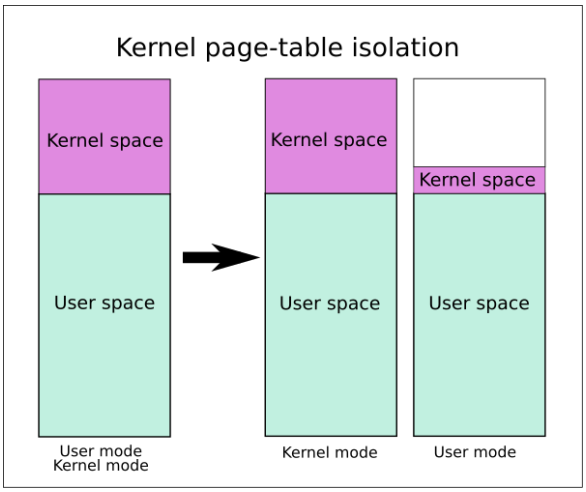

Fig. 5: Introducció de l'aïllament de taules de pàgina

Per plantejar la nova formar de poder accedir a aquestes pàgines s'ha investigat en una part concreta de l'hipervisor, mirant la part de codi que es comunica amb el gestor de memòria del nucli. En ella es troba documentat que per aquest hipervisor existeix un mapa de memòria (que guarda com està distribuïda la memòria del sistema) i que conté totes les direccions de taules de pàgina del nucli per la màquina virtual. De fet, segons indiquen els mètodes de la classe, pot retornar la referència del registre CR3 en aquest mapa de memòria per poder accedir a totes les entrades.

Aquest nou plantejament deixa fins a 3 taules de pàgina que contenen espais d'adreces per accedir a un codi determinat. Per poder accedir a cadascuna es fa a través del registre CR3, encarregat de guardar la direcció física de la primera entrada de cada taula, per tant, per poder accedir al codi que desitgem s'ha d'aconseguir que la direcció de la taula desitjada estigui al registre CR3. Pot veure's la distribució dels mapes de memòria a l'apèndix 4.

Tenint en compte la documentació de l'hipervisor i com queda la nova distribució, seria possible accedir als espais de memòria del nucli a través de l'hipervisor, complint també els permisos necessaris per poder llegir les pàgines protegides del nucli segons els anells de protecció establerts a Linux. Tot i discutir els resultats de l'anàlisi més tard, per ara cal saber que després d'implementar aquesta solució, a l'apèndix 6 es veu que els resultats de caminar per aquestes taules de pàgines no són els esperats. Per donar resposta a aquest imprevist i també per la manca de documentació, es parla amb el creador de l'hipervisor Rian Quinn mitjançant Slack [9], un xat estil IRC que permet parlar amb els desenvolupadors dels projectes. En ell ens esmenta que aquest mapa de memòria corresponent al VMM no necessàriament ha de tenir totes les regions presents, de fet només mapeja la porció del sistema mínima perquè el VMM funcioni. Així doncs, tot i que aquest mapa pot servir per tenir identificades fàcilment les porcions de mapa de memòria, la part corresponent al nucli de Linux no està encara referenciada, ja que abans de la vulnerabilitat Meltdown sí que es podia tenir tot el mapa de memòria mapejat però ara no és possible si no es fan unes operacions prèvies.

Es parla amb el creador de l'hipervisor per investigar com es poden fer aquestes operacions prèvies i s'arriba a una opinió comuna que no hi ha cap solució establerta i generalment acceptada ja que Meltdown fou introduït al 2018 amb la finalitat de protegir adequadament l'accés a aquestes pàgines, deixant molt difícil accedir senzillament. Finalment, recordant que existeixen dues taules de pàgina i que si es vol accedir a la taula del nucli s'ha de referenciar primer al CR3, l'última solució que es proposa és esperar que el mateix sistema operatiu faci aquest canvi de registre en el mateix CR3, l'hipervisor té la capacitat d'intervenir en les sortides de funcions que referenciïn un canvi de CR3 i, per tant, es tindria accés en aquell moment a la direcció on comença la taula de pàgines del nucli.

Aquesta solució, tot i no haver-se pogut aplicar pel curt temps de desenvolupament que es disposa, és el camí a seguir per aconseguir l'objectiu del projecte. S'ha de tenir en compte, que dependre explícitament dels canvis del registre CR3 fa que el codi sigui molt més lent a l'hora de poder fer l'anàlisi de totes les direccions ja que quan el sistema operatiu necessiti de nou el registre CR3 per l'espai d'usuari, perdríem la referència per on podem analitzar noves direccions.

És molt important veure que aquesta solució no és òptima, ja que per poder fer una anàlisi de totes les pàgines del nucli es necessiten més canvis a les taules de pàgina i, conseqüentment, es necessita un major nombre de canvis en el registre CR3. Aquest funcionament alterarà molt el rendiment final del codi, ja que cada cop que es fa aquest canvi s'ha de netejar el TLB (de l'anglès Translation Lookaside Buffer), aquest emmagatzema a caché les assignacions entre adreces virtuals i físiques, així es redueix el temps en les traduccions i les vegades que s'ha de recórrer les taules de pàgina, en cas de canviar el registre CR3 aquesta assignació ja no és correcta i ha d'esborrar la TLB per generar-se les assignacions de nou. En resum, les operacions de netejat de la TLB són costoses i s'han de mirar de reduir el màxim.

8 EMMAGATZEMATGE DELS RESULTATS

Un cop s'ha pogut accedir als recursos desitjats, s'ha de decidir quins seran els següents procediments del programa, en primer lloc els recursos s'han d'analitzar per veure si a la integritat d'aquests s'ha vist compromesa i després s'ha de trobar una manera de poder-los desar per poder tenir constància de l'estat i resultat de cada anàlisi.

8.1 Anàlisi

Doncs, es farà una comprovació dels atributs que van associats a cada pàgina, aquests atributs es troben a les posicions posteriors de cada direcció lògica com s'ha esmentat a l'apartat 7.1. Es pretén tenir un recull d'aquests bits i dels permisos que alguns d'ells indiquen a cada direcció, es pot veure un resum a la taula 1.

Inicialment ens interessa donar una ullada als atributs pertinents a permisos d'execució, aquests són l'executable (X) o el no executable (NX) i el de lectura-escriptura (R/W) o bé només lectura (RO). Per demostrar que qualsevol codi executable allotjat en aquestes direccions no ha estat modificat i es farà una comprovació que miri que totes les pàgines estiguin marcades com a executable i siguin només de lectura (RO), addicionalment, per veure que el codi és del nucli caldrà mirar el bit de supervisor. En aquest cas ens fixem a la taula 1 per veure que el segon bit de l'adreça si està a 1 vol dir que es de lectura/escriptura i en cas que estigui a 0 l'adreça correspondria a només lectura. També en aquesta taula trobarem els bits Accessed que caldrà retornar al valor 0 després de cada visita a una nova pàgina i també el bit Dirty podria ser analitzat per futures versions ja indica quan ha estat modificada una pàgina, però el fet de no poder diferenciar si ha sigut el nostre propi sistema o un atacant ha fet que no sigui utilitzat per l'anàlisi de modificacions. D'altra banda, per veure els permisos d'executable caldrà mirar el 64é bit (present a la figura 9 de l'apèndix B) on en cas que es trobi a 1 serà una pàgina executable i, per tant, que podria executar codi maliciós.

A més de comprovar que la correspondència de pàgines i permisos és la mateixa, és a dir, que una direcció que abans tenia uns permisos ara té uns altres, també es mirarà que cap pàgina amb permisos NX ara en té de X. En cas de no revisar que el contingut de les direccions ja conegudes no hagi canviat, cal fer un ull que una pàgina que anteriorment era no executable ara no sigui executable, ja que pot ser un indici que s'ha introduït codi maliciós. És important que una nova direcció abans no existent amb els permisos supervisor, executable i només lectura serà marcada també com a possible modificació.

TAULA 1: FORMAT D'UNA ENTRADA DE 64 BITS

Distribució bits a direccions lògiques								
Número bit	1	2	3	4	5	6	7	8
Contingut	Present	R/W	PWT	PCD	Acessed	Dirty	PAT	Global

Per saber que hem de guardar, buscarem sempre emmagatzemar el mínim d'informació que ens permeti identificar una modificació en una entrada de taula de pàgina del nucli, d'aquesta manera podrem reduir l'espai necessari per emmagatzemar els resultats i els temps d'execució i sobrecàrrega del codi seran menors al fer menys comprovacions posteriors, per tant, s'han recollit dues opcions on s'intenta guardar el mínim d'informació.

En primer lloc a partir de l'apuntador de l'entrada corresponent de la taula de pàgines es pot obtenir la direcció física on trobar el codi desitjat, d'aquesta manera es podria fer un resum (SHA-512) total o parcial del contingut de la direcció amb conjunt dels permisos que pot tenir la direcció lògica. Amb aquesta solució es poden recollir els permisos inicials, la direcció associada a ells i, a més, la integritat del codi que trobaríem en aquesta direcció. Per poder minimitzar l'espai d'aquests resums, es pot elaborar per regions de memòria una estructura de dades en arbre Merkle Tree, així es podria comprovar fàcilment un major nombre de pàgines amb una operació.

Així i tot, cal recordar que un dels requeriments és que aquest codi pugui córrer a una impressora, l'opció de fer operacions criptogràfiques queda fora de l'abast pel cost computacional d'aquestes operacions i les restriccions en les especificacions tècniques en impressores de més baix cost. S'ha mirat i recomanat l'opció d'integrar un xip que actualment estan incorporant altres solucions IoT per poder donar més facilitat a la unitat de processament a les operacions criptogràfiques, aquest és el xip ATECC608B [10] aquest coprocessador podria permetre implementar en aquest projecte les operacions criptogràfiques necessàries per fer signatures i resums sense afegir una càrrega extra.

En segon lloc, i l'opció que també és molt vàlida és poder tenir un recull de les direccions lògiques de les pàgines i la relació dels seus permisos, i és la senzillesa d'aquesta opció la que ha fet que la triem, guardar la direcció lògica i els seus permisos associats compleix els requisits que s'han de complir per comprovar modificacions o addicions de codi maliciós.

8.2 Desament

S'ha de tenir present que qualsevol emmagatzematge de resultats conté dades sensibles o privilegiades i no han de ser accessibles per l'usuari o des de l'espai de memòria de l'usuari. Un cop definit l'anàlisi de què guardarem ara caldrà dir on es pot emmagatzemar, els tres plantejaments que s'han fet per definir on es guardaran les dades són:

1. Guardar-ho a l'espai de memòria de l'usuari.
2. Guardar-ho en un espai de memòria del monitor de la màquina virtual (VMM).
3. Guardar-ho al mateix espai de memòria del codi de detecció.

La primera idea plantejada seria la més simple de poder implementar, constaria en guardar tots els missatges de resultat del programa de detecció en un arxiu emmagatzemat al sistema. Això podia fer-se mitjançant la comanda '$\gg$' a Linux que permet fer un dump a un fitxer a partir de l'output de l'execució d'una comanda. Aquest escenari pretenia ser temporal, ja que comporta grans problemes de seguretat extreure informació del nucli a un fitxer d'accés per l'usuari, en cas de restringir-li l'accés a només permisos de superusuari tampoc serviria, ja que situant dades sensibles del nucli a l'espai d'usuari trencaria amb l'objectiu d'aquest projecte d'aïllar les mesures de protecció del que es vol protegir.

Amb aquest plantejament es pretenia anar omplint un fitxer mentre s'executava el codi de detecció i escriure-hi el que anés trobant, però no es va tenir en compte que des d'on està situat el nostre codi a escala d'arquitectura del sistema i, bàsicament, no es disposa d'un sistema de fitxers, ja que aquest és una part del sistema operatiu i els directoris es creen dins d'ell. La segona opció tot i no ser aplicada, presenta una molt bona alternativa a l'aplicada actualment, el fet de poder reservar espai de memòria del VMM pot permetre tenir dades guardades a memòria i ser accessibles inclús després d'apagar el sistema. Així s'obririen noves possibilitats d'anàlisi on es pogués aprofitar l'estat d'execucions passades o també es podria permetre parar el procés en qualsevol moment. Finalment, es va triar la tercera opció agafant com a resultats de l'anàlisi les direccions físiques i permisos, es poden guardar totes les dades dels resultats a l'espai de memòria del programa, per tant, crear una variable general al nostre programa que sigui accessible per cada una de les iteracions perquè es puguin fer les comprovacions necessàries. Aquesta variable o variables emmagatzemen les entrades de totes les taules de pàgina del kernel i els seus respectius permisos, per fer-ho possible el projecte inclou una variable de tipus map on les claus serien les direccions físiques i el valor el recull de permisos obtinguts en la detecció. Aquesta solució ens podrà permetre comprovar modificacions respecte anteriors deteccions, ja que si el nostre codi no finalitza, seguirem tenint els resultats en l'espai de memòria del programa.

9 COMPARACIÓ DE RESULTATS

Un cop conservat l'estat de les anàlisis anteriors, podem utilitzar-los per detectar les modificacions inesperades al nostre sistema, per això caldrà accedir a les dades guardades i distingir per cada direcció els seus permisos associats.

9.1 Accés a les dades existents

Per poder accedir a les dades existents, el nostre codi disposa d'unes variables generals on es guarden les direccions físiques i els seus permisos, de manera que a cada iteració en la cerca de noves direccions a analitzar, s'anirà guardant el resultat en elles. Quan comença l'execució del codi de detecció, aquests té dues tasques inicials, per un lloc ha de llegir les direccions lògiques següents o noves que corresponguin al nucli del SO i per l'altre ha de ser capaç de detectar les modificacions que puguin anar apareixent.

El codi és capaç de contemplar interrupcions enmig d'una anàlisi, ja que quan rebi aquests avisos de posposament de cerca, guarda l'última direcció on s'ha quedat i, en reprendre la cerca, ho fa per l'últim registre de direccions que es tingui, en cas de no trobar permisos associats es tornarà a analitzar aquella direcció i si ja ha estat analitzada passarà a la següent.

També és molt important tenir en compte, que quan es camina per les taules de pàgina hi ha una sèrie de bits que poden ser modificats, aquests és el cas del accesed bit (o bit d'accés en català) que determina si aquesta pàgina ha estat consultada anteriorment, un cop s'hagi accedit a les dades necessàries de la pàgina es tornarà a posar al seu estat original.

9.2 Deteccions de modificacions

El codi dins de les seves tasques, és encarregat de llegir noves direccions lògiques i anirà combinant aquesta funció amb la comparació de dades d'anàlisis anterior. Per tant, el primer cop que el codi sigui executat llegirà una certa quantitat de direccions lògiques i les guardarà al registre general, després a la segona iteració podrà comprovar que les direccions observades de l'anterior iteració conserven els mateixos permisos, per cada nova iteració també es revisarà si existeixen noves direccions lògiques per analitzar. Un cop el codi ha fet una primera passada en l'anàlisi de totes les direccions lògiques disponibles ja podrà detectar i classificar noves direccions sospitoses amb permisos executables, de lectura i supervisor, fins aquell moment només podrà detectar diferències en permisos en direccions ja analitzades.

Finalment, quan una modificació és detectada, el nostre codi el guarda en un registre per una posterior anàlisi d'un administrador, per futures implementacions es pot plantejar en crear una alerta o report que vagi directament a l'administrador sense que el subjecte hagi de consultar-ho expressament.

10 EXECUCIÓ EN TEMPS REAL

Com s'ha esmentat amb anterioritat, el codi corre mentre el sistema està en marxa, l'anàlisi que es fa i la detecció de qualsevol modificació es duu a terme mentre el sistema fa les seves tasques o l'usuari l'utilitza, per això podem dir que és un sistema de deteccions en temps real. Tot i això, executar un sistema de detecció de modificacions en temps real comporta un augment del consum de recursos de l'equip, ja que es comparteixen recursos entre el codi de detecció i el sistema. Per això és molt important posar mesures de comprovació del rendiment del sistema per poder anar vigilant que el nostre codi no sigui un impediment i tingui un impacte gran en el rendiment de l'equip, gràcies a la comprovació de rendiment el codi de detecció podrà regular el seu grau de consum de recursos en moments que el sistema ho necessito o bé aprofitar tots els recursos possibles quan aquests estiguin lliures. Quan els recursos són limitats, com pot ser en el cas de les impressores domèstiques o bé quan les tasques de processament necessiten molts recursos com pot ser el cas d'impressores 3D o impressores industrials, aquest apartat agafa un pes més important.

10.1 Comprovació de rendiment

Per poder analitzar el rendiment del codi, s'ha decidit fer una comprovació de la càrrega de la CPU i consum de recursos abans i durant l'execució del nostre codi de detecció. Per poder obtenir aquestes mètriques es pretenia fer ús d'eines d'anàlisi des de l'hipervisor per tal d'obtenir-les directament i poder-les aprofitar. Altres hipervisors del mercat com Xen disposen d'eines com pot ser la utilitat XenTop o també XenMon, ambdues són eines de monitorització del rendiment i permeten obtenir els resultats des del mateix hipervisor. Tot i això, recordem que Bareflank és un projecte nou i, de moment, no té una gran comunitat darrera com per adaptar-li aquestes eines, també després de parlar amb el creador de l'hipervisor Rian Quinn ens comenta que per

ara no existeix cap, però que s'espera que n'hi hagi per un futur.

La utilitat d'anàlisi perf, serveix per poder analitzar el rendiment de diferents mètriques del sistema i permet utilitzar un gran ventall de subcomandes més específiques, en concret, es buscava integrar l'ús de perf stat o perf top pel nostre codi. Tot i això, integrar aquestes eines com s'ha fet amb XenTop o XenMon no és feina fàcil i no es disposava del temps necessari per fer-ho al no entrar als objectius del projecte. El no poder implementar aquestes solucions no ens permet poder fer un codi que s'adapti automàticament però si que es pot fer ús d'aquestes comandes des de l'espai d'usuari per veure l'impacte en rendiment del codi

D'altres idees que poden servir per implementar en un futur és l'ús dels HPCs (de l'anglès High Performance Counters) que són registres propis de l'arquitectura Intel que es poden utilitzar per fer aquestes mesures a un nivell d'abstracció més alt.

Finalment, tot i no poder fer la implementació completa es fa ús de les comandes manualment des del sistema analitzat per poder veure en quant augmenta l'ús de la CPU durant l'execució del codi de l'hipervisor i així es pot observar a la següent figura.

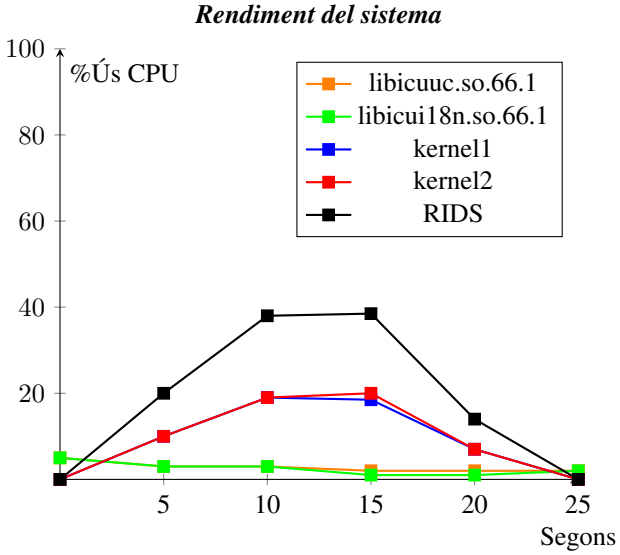

Fig. 6: Llistat de la càrrega dels processos durant l'execució de l'anàlisi

Podem observar com en l'inici de l'execució del nostre programa l'ús de CPU per part d'alguns dels processos generals del SO no supera el 5%, representats pels colors taronja i verd es pot observar com la càrrega que impliquen es independent a l'execució del programa. En canvi, els processos kernel1 i kernel2 que corresponen a dos fils d'execució del codi de detecció arriben a un pic màxim d'un 20% d'ús cada un, en aquest cas podem veure com els processador llença dos processos (colors blau i vermell) pel nostre programa i com augmenten progressivament, després de 15 segons comencen a baixar fins que parem l'execució.

Quan el codi comença a llegir direccions lògiques recordem que ha de baixar 4 nivells de paginació, filtrant només després del tercer nivells els permisos de les pàgines, això provoca que conforme es vagin analitzant noves direccions s'acumulin al final del tercer nivell. Pensem en la metàfora d'unes bales baixant per unes escales i on només al baixar el tercer esglaó podem classificar-les, el fet de parar allà a distingir-les pot fer que se'ns acumulin bales, doncs en aquest cas la classificació de bales serà l'anàlisi dels bits de cada direcció i l'acumulació serà un augment progressiu de la càrrega del sistema, un cop s'han classificat i emmagatzemat les direccions necessàries la càrrega torna a nivells de càrrega més baixos.

Comptant que el codi comença als 5 segons i acaba als 20, durant 15 segons analitza des de la direcció 0xFFFF800000000000 fins 0xFFFFc87FFFFFFFFF, una prova d'anàlisi d'aproximadament 2 MBytes i que mostra com l'impacte màxim de rendiment en aquest cas és del 40% (etiqueta negra RIDS).

10.2 Control del codi de detecció

Gràcies al coneixement de l'estat de càrrega del sistema hoste, podrem controlar la demanda de recursos que tindrà el nostre codi de detecció. Amb aquest objectiu s'aconsegueix que en moments on la saturació del sistema és alta, les comprovacions del nostre codi quedin ajornades per quan el sistema es descongestioni en càrrega de CPU.

Per fer-ho possible, el mòdul de control de codi anirà fent peticions d'estat de càrrega al mòdul de comprovació de rendiment de manera que les respostes inclouran l'informe detallat de l'estat del sistema. S'ha establert un valor llindar al 75% de la càrrega de la CPU i el control de detecció rebi que l'estat de l'equip ha arribat a tal valor, posposarà futurs accessos a pàgines del kernel, d'aquesta manera la tasca que afegeix més càrrega al codi quedarà suspesa fins a nou avís. Tot i això, sí que seguirà operant la part de control de codi i la de comprovació de rendiment, en tornar-se a fer una nova petició després d'un interval de temps determinat, si la càrrega ha disminuït del 30% es reprendrà l'accés a les pàgines del kernel anterior, cal recordar que el mòdul del codi encarregat de l'accés a les pàgines guarda el progrés que s'havia fet i de la direcció de l'última pàgina visitada per si necessita posposar els seus accessos.

Com a conseqüència, aconseguim un codi adaptatiu capaç de saber en quins moments ha de consumir menys recursos i en quins pot consumir-ne els necessaris per fer una anàlisi més exhaustiva, d'aquesta manera l'impacte en el rendiment de l'equip on aquesta solució és implementada és molt menor i queda disminuït només als costos de ser un sistema custodiat per un hipervisor.

11 Conclusions

Durant el projecte s'ha intentat desenvolupar un sistema de detecció d'intrusions en temps real al nucli de Linux, d'inici s'esperava que un projecte amb aquesta complexitat tècnica fos un veritable repte que faci aplicar els coneixements d'un enginyer, i així ha estat, buscant trobar en l'accés a les parts privilegiades d'un sistema una solució innovadora al sector. S'han aplicat solucions de codi obert i aspectes com la virtualització del sistema de protecció, per tal d'aconseguir que el codi no sigui vulnerable per un atacant i s'ha adaptat el consum de recursos perquè aquest projecte sigui fàcilment adaptable a les impressores i al món de l'internet de les coses en general. Tanmateix, s'ha assolit que el codi sigui capaç de detectar les modificacions fruït d'un atac

i que el codi adapti l'impacte generat en el rendiment de l'equip, i convé ressaltar que ha quedat plantejada una investigació totalment apte per fer possible la resolució total del projecte.

AGRAÏMENTS

Per aquest projecte cal agrair a tota la família, amics i companys de vida pels consells i suport psicològic que han fet possible que aquest projecte hagi florit. També al tutor del treball Angel Elbaz per totes les recomanacions, suport i disponibilitat que ha mostrat durant tot el procés i que m'ha animat a investigar tot allò que desconeixia. Finalment a la companyia HP i tutors en l'empresa que em van orientar durant el projecte i van facilitar els recursos en temps i pressupost necessaris per fer viable la investigació.

REFERÈNCIES

[1] J.Horn, "Meltdown: Reading Kernel Memory from User Space" , 27th Security Symposium Security 18, 2018 [Abstract] Available: https://meltdownattack.com/meltdown.pdf .[Accessed 18 Nov 2020]

[2] Andrew S. Tanenbaum, Maarten Van Steen "Distributed Systems: Principles and Paradigms", 2nd Ed. Addision-Wesley, Pearson 2007.

[3] Snort. (2021). Cisco Systems. Accessed: Feb. 4, 2021. [Online]. Available: https://talosintelligence.com/snort

[4] Ossec. (2020). Ossec Project Team. Accessed: Feb. 4, 2021. [Online]. Available: https://www.ossec.net/

[5] Black Hat. Numchecker: A System Approach for Kernel Rootkit Detection. (16 Aug. 2016). Accesed: Nov. 1, 2020. [Online Video]. Available: www.youtube.com/watch?v=TgMsMwsfoQ0.

[6] Bareflank. (2016). Assured Information Security, Inc. Accessed: Sept. 15, 2020. [Online]. Available: http://bareflank.github.io/hypervisor/

[7] Intel. Does My Processor Support Intel® Virtualization Technology?. Nov 10, 2020. [Online] Available: https://www.intel.com/content/www/us/en/support/articles/000005486/processors.html.

[8] Intel, Intel® 64 And IA-32 Architectures Software Developer Manual: Vol 3. Nov. 10, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html

[9] Join Bareflank On Slack!, Bareflank, 2020. [Online] Available: https://app.slack.com/client/TPN7LQKRP/CPHB7UQPL

[10] Microchip.com, ATECC608B - Crypto Authentication, https://www.microchip.com/wwwproducts/en/ATECC608B (accessed Dec. 20, 2020).

APÈNDIX

A.1 Arquitectura del programari del projecte

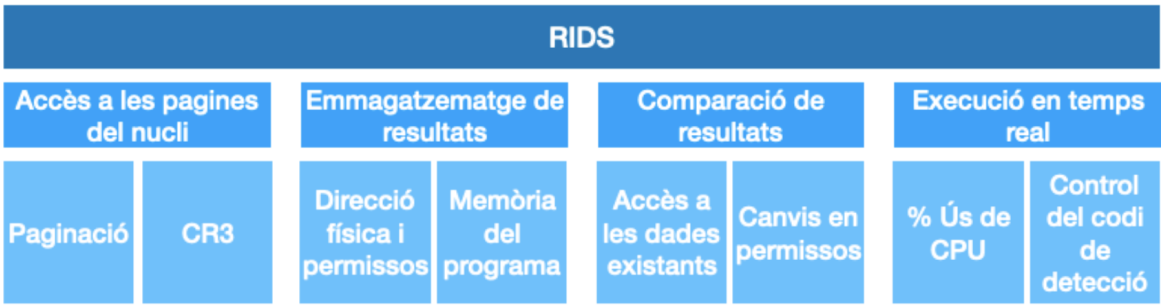

Fig. 7: Desglossament per mòduls del software RIDS

B.2 Sistema de paginació

Demostració de l'estructura jeràrquica d'una direcció lògica per adreçar una direcció física, en ella es poden observar els nivells de paginació de l'arquitectura x86 i com s'obtenen els apuntadors i els índexs per arribar a la posició de cada taula de pàgina: PML4, PDPT, PDT i PT. Imatge procedent la documentació oficial Intel a [8].

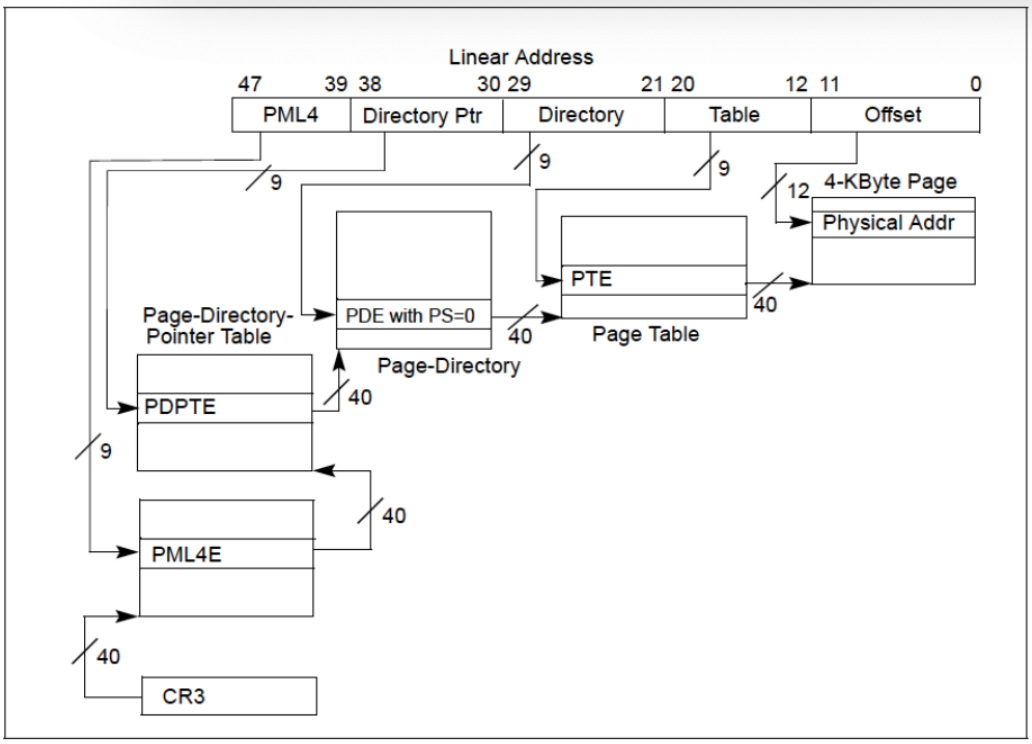

Fig. 8: Estructura jeràrquica de les direccions lògiques de 64 bits

A la figura 9 es pot observar en detall la posició i ús de cada bit de la direcció de 64 bits, on es troben els bits a analitzar U/S, R/W i A entre d'altres. També ajuda a entendre la distribució dels bits per arribar a obtenir des del registre CR3 fins a la pàgina on trobar la direcció física.

63	62–M	M-1	32–12	11–8	7	6	5	4	3	2	1	0	
Reserved[2]			Address of PML4 table		Ignored			PCD	PWT	Ign.			CR3
XD	Ignored		Rsvd.	Address of page-directory-pointer table		Ign.	Rsvd	Ign	A	PCD	PWT	U/S	R/W	1	PML4E: present
Ignored														0	PML4E: not present
XD	Prot. Key[4]	Ignored	Rsvd.	Address of 1GB page frame	Reserved			PAT	Ign.	G	1	D	A	PCD	PWT	U/S	R/W	1	PDPTE: 1GB page
XD	Ignored		Rsvd.	Address of page directory					Ign.	0	Ign	A	PCD	PWT	U/S	R/W	1	PDPTE: page directory
Ignored														0	PDTPE: not present
XD	Prot. Key[4]	Ignored	Rsvd.	Address of 2MB page frame	Reserved			PAT	Ign.	G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 2MB page
XD	Ignored		Rsvd.	Address of page table					Ign.	0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored														0	PDE: not present
XD	Prot. Key[4]	Ignored	Rsvd.	Address of 4KB page frame					Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored														0	PTE: not present

Fig. 9: Format del registre CR3 i entrades en l'estructura de paginació

```
800000015e18c00b PTE  R/W  S  X
800000015e18d00b PTE  R/W  S  X
800000015e18e00b PTE  R/W  S  X
800000015e18f00b PTE  R/W  S  X
800000015e17800b PTE  R/W  S  X
800000015e17900b PTE  R/W  S  X
800000015e17a02b PTE  R/W  S  X
800000015e17b02b PTE  R/W  S  X
800000015e12800b PTE  R/W  S  X
800000015e12902b PTE  R/W  S  X
800000015e12a00b PTE  R/W  S  X
[0x3] DEBUG: --------------------FINISH--------------------
[0x3] DEBUG: host os is now in a vm
```

Fig. 10: Navegació fins a l'últim nivell de paginació i permisos de les pàgines analitzades

C.3 Mapes de memòria

En aquesta figura es veu una representació de quin és l'escenari en el plantejament de les tres taules de pàgina: una pel mode nucli, la segona pel mode usuari i la tercera pel mode hipervisor. A la segona taula es pot observar l'espai que conserva apuntador de l'espai del nucli com es pot veure a la figura 5 i a la tercera les regions del mapa de memòria que ens interessen per obtenir el nucli del sistema operatiu.

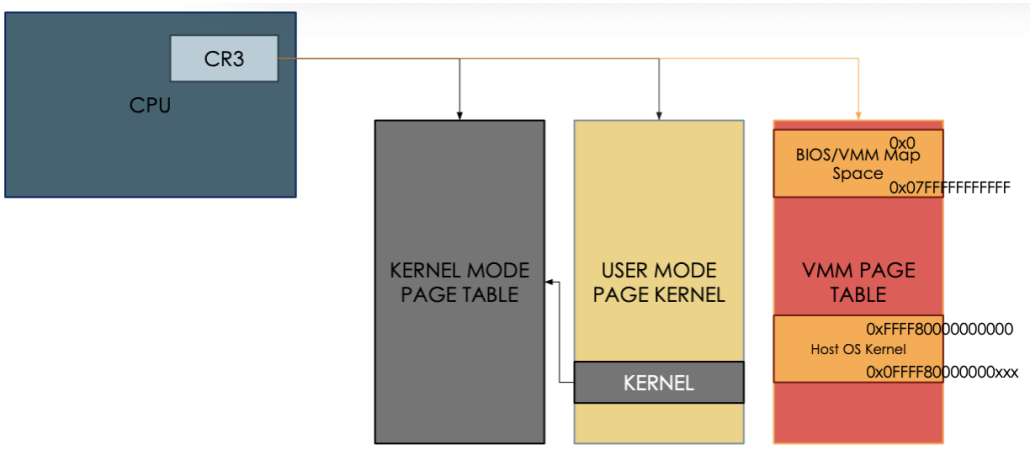

Fig. 11: Mapes de memòria i apuntador en el registre CR3 de l'inici de la taula

D.4 Rendiment del codi

Anàlisi del rendiment del codi durant la seva execució, es pot observar a la figura 12 l'estat abans d'iniciar la detecció i a la figura 13 els nous processos creats amb nom kernel, capturats al punt on causen major sobrecàrrega al sistema.

```
Samples: 2M of event 'cycles', 4000 Hz, Event count (approx.): 100349092463 los
Overhead  Shared Object                          Symbol
   5,07%  libicuuc.so.66.1                       [.] 0x00000000000f1bbb
   5,05%  libicui18n.so.66.1                     [.] icu_66::RuleBasedColla
   5,05%  libsqlite3.so.0.8.6                    [.] 0x000000000005ac57
   4,83%  libsqlite3.so.0.8.6                    [.] 0x000000000005ac23
   2,37%  libicuuc.so.66.1                       [.] 0x00000000000f1cf1
   2,29%  libicuuc.so.66.1                       [.] 0x00000000000f1d24
   2,01%  libicuuc.so.66.1                       [.] 0x00000000000f1be7
   1,98%  perf                                   [.] hpp__sort_overhead
   1,88%  libicuuc.so.66.1                       [.] 0x00000000000f1bc2
   1,74%  libicuuc.so.66.1                       [.] 0x00000000000f1bd4
   1,55%  libicuuc.so.66.1                       [.] 0x00000000000f1bb8
   1,45%  perf                                   [.] rb_next
   1,25%  libicuuc.so.66.1                       [.] 0x00000000000f1b97
   1,09%  libicuuc.so.66.1                       [.] 0x00000000000f1baf
   1,03%  libicuuc.so.66.1                       [.] 0x00000000000f1ba6
```

Fig. 12: Percentatges de càrrega de CPU per processos abans de l'execució del codi de detecció

```
Samples: 1M of event 'cycles', 4000 Hz, Event count (approx.): 127044763115 los
Overhead  Shared Object                          Symbol
  19,22%  [kernel]                               [k] 0x00002b8740051416
  19,06%  [kernel]                               [k] 0x00002b8740051482
   2,36%  libsqlite3.so.0.8.6                    [.] 0x000000000005ac23
   1,90%  [kernel]                               [k] xhci_update_erst_dequeu
   1,39%  [kernel]                               [k] do_syscall_64
   1,35%  libsqlite3.so.0.8.6                    [.] 0x000000000005ac57
   1,21%  perf                                   [.] hpp__sort_overhead
   1,04%  [kernel]                               [k] acpi_os_read_port
   0,86%  perf                                   [.] rb_next
   0,80%  [kernel]                               [k] xhci_irq
   0,71%  libicui18n.so.66.1                     [.] icu_66::RuleBasedCollat
   0,46%  libc-2.31.so                           [.] __memmove_avx_unaligned
   0,45%  [kernel]                               [k] copy_user_enhanced_fast
   0,42%  perf                                   [.] hist_entry__sort
   0,38%  libicuuc.so.66.1                       [.] 0x00000000000f1bbb
```

Fig. 13: Percentatges de càrrega de CPU per processos durant de l'execució del codi de detecció