LinuxFocus article number 294
http://linuxfocus.org

IDS - Intrusion Detection System, Deel II

door Klaus Müller
<Socma(at)gmx.net>

Over de auteur:
Op het moment is "Klaus Müller (ook bekend als "Socma") nog altijd student, hij is bezig met linux programmeren en met beveiliging-gerelateerde onderwerpen.

Vertaald naar het Nederlands door:
Guus Snijders
<ghs(at)linuxfocus.org>

Kort:

In Part I richtten we ons op typische aanvallen op Intrusion Detection Systems (inbraak detectie systemen). Deel II introduceert methoden voor het ontdekken ervan en onze reacties - waaronder de toepassing van signatures (profielen) en filters. Ten slotte introduceren we Snort en LIDS.

Analyse Mogelijkheden

Eerder keken we naar aanvallen om IDSsen en verschillende andere systemen tegen te beschermen. Vervolgens bekeken we methodes voor de analyse en hoe een IDS bepaalt of er een aanval plaats vond, respectievelijk of de aanval succes had, of niet.

In principe maken we onderscheid tussen Misbruik Detectie en Anomaly Detectie. Misbruik Detectie gebruikt specifiek gedefinieerde patronen om een aanval te ontmaskeren. Deze patronen worden "signatures, profielen" genoemd, en zullen besproken worden in een eigen sectie. Voor het moment is het genoeg om te weten dat we signatures kunnen definiëren die het netwerk verkeer doorzoeken naar bepaalde strings (bijvoorbeeld /etc/passwd), toegangsvragen voor bepaalde bestanden afwijzen en een alarm in werking stellen. Het voordeel van Misbruik Detectie is de lage waarschijnlijkheid van false alarms daar de zoek criteria van signatures zeer strak gedefiniëerd kan worden. De nadelen zijn ook erg duidelijk, nieuwe aanvallen worden vaak gemist omdat ze niet gedefinieerd zijn (zie sectie... over signatures).

De andere methode is Anomaly Detectie. Dit betekend gewoon dat er een profiel van de normale activiteiten van de gebruiker is vastgelegd. Als het gedrag van de gebruiker teveel afwijkt van het

profiel, wordt er een alarm afgegeven. De eerste stap van deze analyse is het aanleggen van de profielen (database) van "normale" activiteiten van de gebruiker. Een aantal stappen kan worden vastgelegd: Hoe vaak voert de gebruiker specifieke commando's uit? Wanneer voert hij specifieke commando's uit? Hoe vaak opent hij specifieke bestanden? ... Een kort voorbeeld: - Gebruiker "Voorbeeld" voert *su* drie keer per dag uit. Plotseling - op een dag - voert gebruiker "Voorbeeld" *su* zeven keer op een dag uit, meer dan twee keer zoveel als normaal. Anomaly Detectie zou dit "abnormale" gedrag opmerken en de beheerder waarschuwen over gebruikers "Voorbeeld" zeven keer uitvoeren van *su* tegen drie keer "normaal" gemiddelde. De nadelen van deze procedure werden me duidelijk toen ik begon met de implementatie ervan (zie voorbeeld aan het einde - COLOID). De methode om een database op te zetten voor gebruikersactiviteiten is nogal belastend voor het systeem. We monitoren, bijvoorbeeld, hoe vaak de gebruiker tien specifieke bestanden heeft geopend. Met ieder *open()* commando moet gecontroleerd worden of het een van de tien specifieke bestanden is en als het resultaat positief is, gaat de teller met een omhoog. Desalniettemin, is dit een goede mogelijkheid om nieuwe aanvalstechnieken te achterhalen, daar ze meestal zullen opduiken als "abnormaal". Verder kan de beheerder zelf aangeven welke afwijking wordt beschouwd als "abnormaal", een afwijking van 10% of zelfs 75% ... Door methode te gebruiken moeten we oppassen dat we de gebruikers profielen genereren in een "veilig" netwerk, anders zou het gedrag van een aanvaller als normaal kunnen worden gezien en het gedrag van een legitieme gebruiker als abnormaal.

In het algemeen omvat Anomaly Detectie de volgende procedures:

- Threshold Detection = in dit gebied worden tellers gebruikt, welke bijhouden hoevaak wat wordt uitgevoerd, geopend, gestart... Deze statistische analyse kan worden gebruikt met de zogenoemde Heuristic Treshold Detection.
- Rule-Based Detection = (NL: Detectie gebaseerd op regels), zou het gebruik de regels overtreden wordt er een alarm gestart.
- Static Measure = (Statisch meten) het gedrag van gebruikers/ systemen voldoet aan een profiel welke ofwel was voorgedefinieerd, of op een andere manier gegenereerd. Een programma om normale gebruikers activiteiten te monitoren voor de profielen is vaak inbegrepen.

Heuristic Threshold Detection, in dit geval de teller (hoe vaak wat mag worden uitgevoerd), wordt in het begin niet op een statische, maar op een dynamische waarde gezet. Voert een gebruiker normaal gesproken */bin/login* 4 keer per dag uit, kan de teller op 5 worden gezet...
De Protocol Anomaly Detection representeerd een sub-group van de anomaly Detectie. Dit is een relatief nieuwe techniek, het werkt in principe net als de Anomaly Detectie. Ieder protocol heeft een "voor-gedefinieerd" profiel (zie bijbehorende RFC's). Het doel van de Protocol Anomaly Detectie is om uit te vinden of het gedrag van het protocol overeenkomt met het voor-gedefinieerde, of niet. Er zijn waarschijnlijk meer aanvallen gebaseerd op protocol misbruik dan de lezer zich kan voorstellen, deze subgroep is daarom nogal belangrijk voor IDS'sen. Terugkijkend op de sectie over scannen kunnen we misschien wat indicators vinden voor Protocol Anomaly Detectie.

- Controleer of er parameters voorkomen (NULL scan)
- Controleer of alle parameters voorkomen (XMas scan)
- Controleer op "onzin" combinaties van parameters, zoals SF
- .....

De juiste specifieken zijn te vinden in de gerelateerde RFCs, alsmede welk gedrag niet zou moeten voorkomen, resp. welk soort reactie zou moeten voorkomen op specifieke gebeurtenissen. Verder is er

Application Anomaly Detection (werkt ongeveer gelijk aan Application Based IDSs). In sommige literatuur vond ik een indicatie in die richting, dus keek ik er verder naar. Natuurlijk heeft een programma "normaal" gedrag, bijvoorbeeld hoe het reageert op gebeurtenis X... Y of als de invoer van een gebruiker fout is. Veel voorkomende binaries (bijvoorbeeld *ps, netstat, enz* worden aangepast voor invoer van de gebruiker, in geval van *ps* om bepaalde processen te verbergen. Met behulp van Application Anomaly Detection is het mogelijk om "abnormaal" gedrag van een programma te ontdekken. Sommige applicatie gebaseerde IDS'en werken op deze manier, maar ik heb er weinig ervaring mee.

Tenslotte een andere nieuwe methode van ID-systemen: Intrusion Prevention (inbraak afweren). Het wordt gebruikt in sommige nieuwe ID-systemen, en verschilt van de beschreven methoden van intrusion detection. In plaats van *logbestanden* van het verkeer te analyseren, en dus om aanvallen te ontdekken nadat deze zijn voorgekomen, proberen ze aanvallen tegen te gaan.

In tegenstelling tot de klassieke IDS'sen worden er geen profielen gebruikt om de aanval te ontdekken. Het volgende is een korte uitleg van hoe IPS'en werken, hun functionaliteit wordt duidelijk met de volgende voorbeelden:

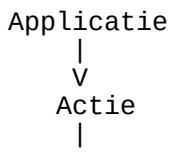
- Monitor applicatie gedrag
- creëer applicatie regels
- Waarschuw bij overtredingen
- Vergelijk met andere gebeurtenissen
- Grijp in op systeemaanroepen (system calls)
- .....

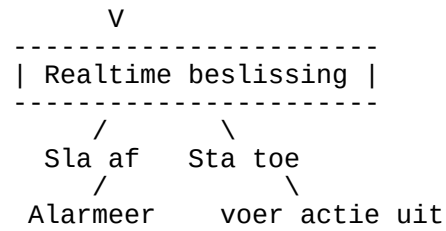
'Monitor applicatie gedrag' komt dicht bij Applicatie-gebaseerde IDS'en, oftewel het gedrag van een applicatie wordt geanalyseerd en gelogged, zoals welke data normaal gesproken wordt opgevraagd, welke bronnen het gebruikt. Net als Anomaly Detectie probeert het uit te vinden hoe een programma normaal opereert, respectievelijk wat het toe gestaan is te doen.

Het derde punt ('Waarschuw bij overtredingen') zou geen uitleg nodig moeten hebben, het is alleen van betekenis in geval van een afwijking. Dit kan resulteren in een simpele log entry of geblokeerde bronnen.

Bij de tweede stap ('creëer applicatie regels') wordt een zogenoemde applicatie regelset samengesteld, gebaseerd op informatie van de analyse in Stap 1 ('Monitor applicatie'). Deze regelset levert informatie over wat een applicatie is toegestaan te doen (welke bronnen gebruikt mogen worden) en wat de applicatie niet is toegestaan.

'Vergelijk met andere gebeurtenissen' betekend het delen van informatie met samenwerkende sensoren, dit levert betere bescherming tegen aanvallen.



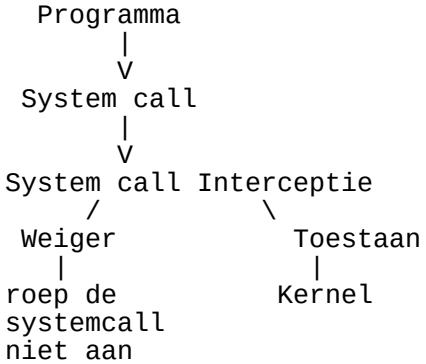


Dit vereenvoudigde diagram verduidelijkt het proces nog een keer. Voordat er een actie wordt uitgevoert, wordt er een ’Realtime beslissing’ uigevoerd (de activiteit wordt vergeleken met de regelset). Als de activiteit illegaal is (het programma vraagt data op of probeert deze te veranderen, zonder dat het is toegestaan om systeemdata te veranderen), wordt er een alarm in werking gesteld. In de meeste gevallen zullen de andere sensoren (of een centrale console) ook geïnformeerd worden. Dit voorkomt andere computers in het netwerk bij het openen van openen/uitvoeren van specifieke bestanden. Als de activiteit overeenkomt met de regelset, zal het worden toegestaan en zal het verwerken eindelijk verder gaan.

Tenslotte het laatste punt op onze lijst: ’Grijp in op systeemaanroepen’. Gemanipuleerde systeemaanroepen (bijvoorbeeld rootkits) worden vaak gedetecteerd. De benadering voor het ingrijpen van system calls is nogal eenvoudige: voordat een system call wordt "geaccepteerd", wordt hij grondig onderzocht. Onderzoeken betekend, bijvoorbeeld het stellen van de volgende vragen (zie ook [5]):

- wie riep de system call aan (welk programma) ?
- met welke gebruikers authorisatie loopt het proces (root...) ?
- wat probeert de system call te benaderen?

Dit maakt het monitoren van pogingen tot wijzigen van belangrijke configuratie/systeem bestanden mogelijk, we hoeven "alleen maar" te controleren of de system call overeenkomt met de gedefinieerde regelset, of niet...



Daar Intrusie Preventie in vergelijking met andere methoden relatief nieuw is, zal er meer informatie

over dit onderwerp verschijnen.

Als conclusie een hint voor OKENA, een erg potente IPS. In de white paper sectie van www.okena.com kun je meer informatie over Storm Watch vinden. Om meer te leren over de tekortkomingen van Storm Watch, zie [6].

Profielen

Nu zullen we het gebruik van profielen op IDS'en bespreken, het tweede deel zal gaan over hun zwakheden.

Concept

Met behulp van profielen kunnen bekende aanvallen worden herkend. Een profiel zoekt naar een bepaald patroon in het data verkeer. Dit patroon kan gebruik maken van een variatie van dingen als strings, verdachte headers (met ongebruikelijke parameter combinaties), poorten die vaak door trojans worden gebruikt. De meeste aanvallen hebben bepaalde karakteristieken, bijvoorbeeld specifieke parameters die zijn gezet, of bepaalde strings in de lading. Met profielen wordt geprobeerd om een aanval via deze karakteristieken te ontdekken.

Ik zou willen beginnen met de zogenoemde Payload Signatures. Hier is een deel van pakket payload (lading van het pakket):

```
00 00 00 00 E9 FE FE FF FF E8 E9 FF FF FF E8 4F    ...............
0 FE FF FF 2F 62 69 6E 2F 73  68 00 2D 63 00 FF FF .../bin/sh.-c...
```

Zoals we later in de bechrijving van de SNORT regels zullen zien, zijn er een paar opties voor wat te doen. Vaak wordt de inhoud van de lading onderzocht op specifieke strings (in SNORT met 'content' of 'content-list'). Stel dat iemand probeert een wachtwoord bestand probeert te openen (bijvoorbeeld */etc/passwd*), de lading kan worden doorzocht (op */etc/passwd*) als het pakket deze string bevat, kunnen tegenmaatregelen worden genomen, zoals:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80        \
(msg:"WEB-MISC/etc/passwd";flags: A+; content:"/etc/passwd"; \
nocase;classtype:attempted-recon; sid:1122; rev:1;)
```

Een andere mogelijkheid is de detectie van pakketten die niet een specifieke string bevatten.

Een andere beveiliging tegen een mogelijk buffer overflow is de controle van de pakket grootte op

specifieke poorten. Het is meestal mogelijk om de bron poort en de doel poort aan te geven, vragen van specifieke poorten of naar specifieke poorten kunnen worden afgewezen. String profielen in het algemeen zijn payload profielen. Payload profielen controleren de payload van een pakket, dat wil zeggen dat het string profiel van de lading wordt doorzocht op een specifieke string...
Wat zou er verder kunnen worden gedetecteerd met profielen? Om de lading te controleren op specifieke strings is niet altijd het beste. Om het profiel te laten zoeken naar flag (parameter) combinaties van de TCP header is een andere mogelijkheid. Als in een pakket zowel de SYN als de FIN bit zijn gezet, representeerd dit een anomaly welke een aanvaller zou kunnen gebruiken om specifieke eigenschappen van het os (besturingssysteem) (of het os zelf) te achterhalen. Zoals genoemd in het begin, zijn er specifieke poorten waarvan bekend is dat ze vaak door trojans worden gebruikt. Voorbeelden van zulke poorten zijn 31337 of 27374.

Misschien wordt het duidelijker als ik het proces uitleg met een voorbeeld. Laten we eens kijken naar de typische eigenschappen van een synscan aanval:

- verschillende bron IPs
- TCP bron en doel Poort is 21
- TOS (Type Of Service) 0
- IP id 39426
- SF set (SYN en FIN)
- verschillende volgnummers gezet
- verschillende ack (bevestiging) nummers gebruikt
- TCP Window grootte van 1028

In gevallen als dit zou het de functie van een profiel moeten zijn om het onderscheid te maken tussen "normaal" en "abnormaal" gedrag van een verbinding. Sommige IDS’en gebruiken speciale databases met informatie, zoals boven weergeven, zullen ze deze doorzoeken naar overeenkomsten.
In principe kunnen afwijkingen in het synscan voorbeeld worden gedetecteerd door profiel controle:

- Bron en Doel Poort zijn 21 (File Transfer Protocol - ftp).
- Gelijke bron en doel poort duiden niet uitsluitend op de dreiging van een aanval, alleen de waarschijnlijkheid er van.
- SF gezet, zoals boven genoemd, dit zou niet moeten gebeuren daar men geen verbinding opzet en afbreekt op hetzelfde moment.
- Ack nummer is ongelijk aan 0, zelfs als alleen SF en niet ACK zijn gezet. Als ACK niet gezet is, zou het ACK nummer 0 moeten zijn.
- IP ID is altijd 39426, zelfs al zou dit nummer niet constant moeten blijven (volgens de RFC), maar dit duidt niet uitsluitend op een synscan aanval - hetzelfde gaat op voor de constante window grootte...

De ontwikkeling van het profiel voor de detectie van een synscan aanval dient met meer rekening te houden dan de bovengenoemde criteria. Het doel van het profiel zou de detectie van zowel bekende als nieuwe versies van aanvallen moeten omvatten. Voor deze reden zouden generale en speciale karakteristieken moeten worden gecombineerd om de waarschijnlijkheid van de aanval detectie te

vergroten. Zelfs als het mogelijk zou zijn om een nieuw profiel voor iedere versie van een aanval te schrijven, zou dit ons meer bezig houden dan andere, belangrijker dingen. Daarom dienen we aandacht te besteden aan het profiel om zoveel mogelijk aanvallen (en versies ervan) te detecteren zonder dat we het profiel moeten aanpassen.

profielen zouden geschreven moeten worden om specifieke aanvallen te detecteren, maar er moeten algemene profielen zijn om afwijkingen te vinden. Een voorbeeld vna een profiel voor de detectie van een specifieke aanval is hierboven weergegeven (synscan aanval). Een algemener profiel zou bijvoorbeeld de volgende criteria kunnen controleren:

- ACK nummer ongelijk aan 0, zelfs al is ACK niet gezet
- abnormale flag combinaties in de TCP header (SYN en FIN) of andere (zie de beschrijving van de scans)
- ....

Profielen welke in het algemeen zoeken naar afwijkingen in protocollen worden Protocol analyse gebaseerde profielen genoemd, terwijl een andere groep bekend staat als "Packet Grepping".

Zwakheden

Hoewel de methode van lading profielen (inclusief string profielen) betrouwbaar lijkt, zijn er manieren om deze te omzeilen. Mogelijk schrijf ik zelf een paper over hoe je Snort regels om de tuin kunt leiden, maar hier zal ik me beperken tot het essentiële. Een profiel, zoals oven weergegeven, zoekt naar "*/etc/paswwd*", met *nocase*, hoofdletters worden genegeerd. Echter, als we */etc/passwd* niet direct maar indirect benaderen, bijvoorbeeld als de aanvaller gebruikt maakt van *’ /etc/blablabla/.../.../\passwd ’*, zou het profiel dan alarm slaan? Nee, omdat het zoekt naar */etc/passwd* (en andere hoofdletter/kleine letter versies). Een andere limiet van deze profielen is de detectie van de meest bekende aanvallen en het zoeken naar bekende zwakheden. Nieuwere versies van specifieke aanvallen worden vaak niet ontdekt... Andere profielen - welke gespecialiseerd zijn in specifieke aanvallen - of algemene profielen hebben het voordeel van het vinden van nieuwe aanvallen. Men dient echter voorzichtig te zijn met het maken van nieuwe profiel regels. Een profiel dat is gespecialiseerd in het detecteren van een specifieke aanval, zal falen in het vinden van een licht aangepaste variatie (in plaats van een constante IP ID waarde van 39426, kan de nieuwe aanval een variabele waarde gebruiken...). Bij het maken van algemene profielen (protocol analyse gebaseerde profielen) dienen we zeker te zijn dat de regels echt globaal zijn gedefinieerd, dat betekend dat ze in staat zijn om afwijkingen te vinden die anders niet of nauwelijks zouden voorkomen.

Een ander falen duikt op in nauwkeuriger onderzoek van de Unicode aanval (zie [4]). Hier is een typische beschrijving van een beveiligingsgat in MS IIS welke mogelijk was door de Unicode aanval:

"kort samengevat:
Een fout in Microsoft Internet Information Server (IIS) staat remote gebruikers toe om directory lijsten te tonen, bestanden te bekijken, bestanden te verwijderen en commando’s uit te voeren. Aanvallers kunnen de Unicode karakterset gebruiken om URLs samen te stellen om bronnen via IIS te benaderen, die anders onbereikbaar zouden zijn. Alle recente versies van IIS zijn getroffen door deze zwakheid.

Exploitatie van deze zwakheid is triviaal. ISS X-Force is op de hoogte van wijdverspreide exploitatie van deze zwakheid."

Een probleem voor IDS'sen zit in het feit dat karakters in UTF-8 verschillende codes voor hetzelfde resultaat kennen, bijvoorbeeld "A" : U+0041, U+0100,

U+0102, U+0104, U+01CD, U+01DE, U+8721. Daar MS IIS case sensitive (hoofdletter gevoellig) is, zijn er vele mogelijkheden om verschillende karakters weer te geven (er zijn bijvoorbeeld 83.060.640 verschillende mogelijkheden om AEIOU weer te geven).
Since MS IIS is case
Als een aanvaller, bijvoorbeeld, "http://victim/../../winnt/system32/cmd.exe" aanvraagt, genereert IIS een fout. Echter, door "../.." met een UTF-8 equivalent te vervangen wordt er geen fout gegenereerd: "http://victim/..%C1%9C../winnt/system32/cmd.exe". Verder maken de zogenoemde UN-escape UTF-8 codes het mogelijk om pagina's te openen waartoe we geen toegang zouden moeten hebben. Een NIDS heeft mogelijkheden om deze aanvallen te detecteren daar de gebeurtenis plaats vind op de applicatie laag - in zo'n geval zou de toepassing van een HIDS beter zijn. Encryptie vormt meestal een probleem voor sensors - dat feit wordt ondertussen vaak misbruikt. De voortgang van sommige "IDS-Producers" toont duidelijk de limieten van profielen, de beschikbare profielen detecteren sommige bekende Unicode aanvallen, maar worden na kleine aanpassing van de aanval waardeloos. Alleen NetworkICE heeft met succes profielen ontwikkeld die dit type aanvallen ontdekken (Snort en ISS Real Secure hebben profielen ontwikkeld die, echter, alleen de bekende Unicode aanvallen detecteerden).

Reacties

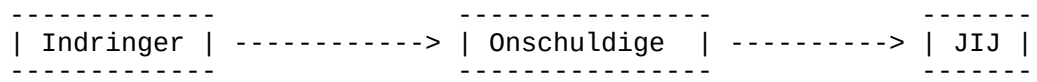
Zoals eerde uitgelegd, analyseren IDS'en de activiteiten op de PC en op het net - maar hoe bruikbaar zou een IDS zijn als het niet zou reageren en ons waarschuwen? Een IDS zou waardeloos zijn, een verspilling van machine performance. 'Response' (reactie) kan worden opgedeeld in Active Response (actieve reactie) en Passive Response (passieve reactie). We zullen de verschillen bekijken in de specifieke sectie.
Geïnteresseerde lezers kunnen ook kijken naar OPSEC

```
Secured by Check Point' appliances are security solutions that integrate
Check Point VPN-1/FireWall-1 technology onto our partners' hardware
platforms.
```

Dit systeem staat de integratie toe van bestaande systemen in Fire Wall-1. Een bijkomend voordeel is de wereldwijde herkenning (het heeft zo'n 300 partners). Als je een aanval herkent kun het IP adres van aanvaller uitsluiten (alleen als een mogelijke reactie) ... Als geïnteresseerd bent in OPSEC, lees "Deployment Platforms", hier zul je ooko de condities vinden om "lid" te worden van het systeem (partner worden).

Actieve Reactie

Actieve reactie betekend automatische reactie als de IDS een aanval (of een poging tot) herkend. Afhankelijk van de ernst van de aanval, bieden de meeste IDS'en meerdere opties voor de reactie.
1) Neem actie tegen de potentiële indringer. 2) Verzamel "simpelweg" aanvullende data (over de indringer en zijn aanval, resp. de implicaties). 3) Pas de configuratie aan
De eerste gechikte reactie zou de initiatie van stappen tegen de aanvaller zijn. Dit kan een variëteit van stappen omvatten, zoals de persoon's toegang uitschakelen of aanvallen tegen de indringer starten. Zoals uitgelegd bij de honeypots is het vaak niet alleen lastig om directe aanvallen tegen de indringer te starten maar vaak ook illegaal. In deze context duikt ook vaak het zogeheten "Third Party Effect" op. Wat is dit effect? Een grafische weergave van een Third Party Effect ziet er als volgt uit:



Het Third Party Effect betekend dat een onschuldig persoon (of onschuldig netwerk) met succes was aangevallen door de indringer, en vervolgens gebruikt door indringer om ons netwerk (en mogelijk andere netwerken) aan te vallen. Dus, wat is het probleem? Het probleem is dat ons netwerk een 'onschuldig' netwerk zou ontmaskeren als zijnde de indringer, en deze vervolgen aanvallen, alleen omdat de indringer dat netwerk gebruikte om ons netwerk aan te vallen. Als resultaat van onze aanval (onder de foute aanname dat we de indringer te pakken hebben) zou onherstelbare schade kunnen worden aangericht op het onschuldige netwerk. Als de indringer slim genoeg was om zijn sporen te verbergen, zullen wij aansprakelijk worden gehouden voor de schade, en niet de indringer.

De tweede optie (aanvullende informatie verzamelen) is minder problematisch. Zou er een potentiële aanval/inbraak gedetecteerd worden, wordt "alleen" aanvullende informatie over de gebruiker en zijn aanval worden verzameld. Als een IDS vastgesteld dat een specifieke gebruiker met succes zijn privileges heeft uitgebreid (of er een ander soort aanval plaatsvondt), wordt de observatie van die gebruiker uitgebreid, bijvoorbeeld het loggen van commando's (indien nog niet geactiveerd), waar de gebruiker op inlogt, wanneer logte hij in, wanneer en hoef vaak logt hij de komende paar dagen in, probeert hij om specifieke binaries over te zetten (FTP)... Op deze manier wordt er een profiel van de indringer samengesteld. Hiermee hebben we het voordeel dat we in staat zijn om gedetailleerde logs te analyseren en potentiële gaten te dichten, het stelt ons ook in staat eventueel juridische stappen te ondernemen.
Als een derde optie zie ik de modificatie van het systeem, de firewall, enz. Als de indringer specifieke IP adressen gebruikt, kunnen we de gebruiker uitsluiten van inloggen op het netwerk via dit IP. Natuurlijk kunnen we andere toegangspogingen van verdachte locaties blokkeren en loggen. In specifieke gevallen kunnen we simpelweg alle toegang tot het eigen netwerk blokkeren (of alle vragen aan specifieke TCP poorten, van specifieke netwerk interfaces...). Een andere mogelijkheid van Active Respons is om een TCP verbinding af te breken (ook bekend als TCP kill). Om de verbinding met een andere computer te verbreken, sturen we een RST (Reset Flag), dit "killed" de sessie. Normaal gesproken wordt een RST verstuurd als er een fout optreed in de verbinding..., in dit geval kan het gebruikt worden door een IDS (zoals ISS RealSecure) om de sessie met een een andere computer te beëindigen (voor Win-NT bestaat er een tool met dezelfde naam).

```
" tcpkill - kill TCP connections on a LAN
  .......
```

```
   tcpkill  kills specified in-progress TCP connections (use-
   ful for libnids-based applications which  require  a  full
   TCP 3-whs for TCB creation). "
```

Dit is een gedeelte uit ’man tcpkill’ ...
Zoals je kunt zien, zijn er echt omvangrijke mogelijkheden om te reageren op aanvallen. Het lijkt aantrekkelijk om een tegenaanval te starten, maar zou niet moeten worden uitgevoerd.

Passieve Reactie

Vergeleken met Active Response worden meestal alleen waarschuwingen gelogt, deze dienen gemonitord te worden door de admin/gebruiker. Hier zijn de opties om te reageren:

1) waarchuwingen, hints...logging
2) genereren van zogenoemde rapporten welke de systemen monitoren voor een bepaalde tijd en een account produceren.

Bijna iedere IDS is in staat om waarschuwingen te genereren of om indicaties te sturen naar de gebruiker/browser. Als het wordt geprobeerd - bijvoorbeeld - om een belangrijk systeembestand te wissen, specifieke services te starten (waarvan het gebruik niet toegestaan zou moeten zijn) ... een waarschuwing, het incident melden, kan gegenereerd worden, alsook wie er deel aan nam en op welke tijd. Ondertussen hebben steeds meer IDS’en de optie om zogenoemde rapporten te genereren. De status van het systeem kan gemonitord worden over uitgebreide perioden, activiteiten kunnen gelogd worden en een status rapport kan gegenereerd worden. Bijna alle IDS’en bieden de optie van passieve reactie.

Filter

Filters worden gebruikt om aanvallen aan de hand van hun handtekening te herkennen. Dit profiel is indirect gerelateerd aan de eerder genoemde profielen, Hier kijken we naar de typische karakteristieken van een aanval (zoals *doel/bron poorten*, *bron/doel IPs* ...) In een ander deel van deze sectie zal ik met N-code een aantal voorbeelden van filters voor bekende aanvallen introduceren. Aan het eind van het artikel is een pagina te vinden (advanced users guide - nfr) welke een uitleg biedt van N-code - als je er niet bekend mee bent.
land:

```
# Dit is een voorbeeld van hoe een land
# aanval te detecteren in N-code
 filter pptp ip () {
        if(ip.src == ip.dest)
        {
                record system.time,
                        eth.src, ip.src, eth.dst, ip.dest
                to land_recdr;
        }
 }
```

Omdat er onbekende variabelen zijn gebruikt, hierbij een korte uitleg :

- ip.src = het bron-IP adres
- ip.dest = the doel-IP adres
- eth.src = MAC adres van de "doel-machine"
- eth.dst = MAC adress van de "bron-PCs"
- record system.time = logt het punt in tijd wanneer de condition ip.src == ip.dest was gevonden

zoals je kunt zien, kent N-code ook de == operator, als je de Advanced User's Guide leest zul je ook andere overeenkomsten (met andere hogere talen) vinden, zo kent N-code de operators + , - , * ... of samengestelde operators als >=, != .... of zoals boven weergegeven ==. Xmas Scan (kerstboom scan): Zoals je je wellicht herinnerd van "Typen Aanvallen" zijn in een Xmas Scan alle flags gezet. Daardoor moet het aannemelijk zijn om te verifiëren of ze alle zijn gezet, of niet. Hiervoor moeten we de waarden van de individuele bits weten:
bits:

```
 Bit                    Value
 -------------------------------------
 F-FIN                  1
 S-SYN                  2
 R-RST                  4
 P-PSH                  8
 A-ACK                  16
 U-URG                  32
-------------------------------------
```

filter xmas ip() { if(tcp.hdr) { $dabyte = byte(ip.blob,13); if(!($dabyte ^ 63)) { record system.time, ip.src,tcp.sport,ip.dest, \ tcp.dport, "UAPRSF" to xmas_recorder; return; } } }
Hier komen weer een paar onbekende variabelen voor, deze zijn:

- tcp.hdr = if tcp.hdr == 0, het pakket bevat geen geldige TCP Header, als tcp.hdr == 1 is dit wel het geval
- tcp.dport = TCP doel poort
- tcp.sport = TCP bron poort
- ip.blob = inhoud van de lading van een pakket (zonder header)
- "UAPRSF" betekend dat URG,ACK,PSH,RST,SYN en FIN zijn gezet

$dabyte is een lokale variabele, toegewezen aan byte (ip.blob,13). Om de "byte expressie" uit te leggen, is hier een kleine demonstratie van de TCP code bits:

```
| Src Port | Dest Port | Seq Number | ACK Number | HDR Length | Flags |\
URG | ACK | PSH | RST | SYN | FIN | Win Size | Chksum | Urg Off | Opt |
```

We realiseren waarom 13 bytes in byte() zijn gespecificeerd, omdat 13 bytes voldoende zijn om de flags te bevatten. Voordat we kunnen begrijpen hoe byte werkt, eerst een paar opmerkingen over blob. Als je refereert aan Hoofdstuk 3 van de Advanced User Guide, blob is een "serie bytes van willekeurige grootte". Een "byte" geeft een byte van de gespecificeerde offset van een blob, de syntax ziet er als volgt uit: byte (str blob_om_te_zoeken, int offset). Het eerste argument specificeert de blob om te doorzoeken (hierboven: ip.blob), het tweede argument definieert de offset (in 'blob_om_te_doorzoeken') van de

gezochte byte. Met ’if(!($dabyte ^ 63))’ wordt gecontroleerd of alle flags zijn gezet, dit zou moeten resulteren in 63 als de waarden van alle flags worden opgeteld (32+16+8+4+2+1), als iemand het wil weten: met ^ wordt een bit voor bit XOR uitgevoerd.

Behalve de genoemde opties, biedt N-ocde veel omvangrijke mogelijkheden. Het is bijvoorbeeld mogelijk om te controleren:

- of het pakket een IP pakket is (ip.is)
- de lengte van het IP pakket (ip.len)
- het gebruikte protocol van het IP pakket (ICMP, TCP of UDP) (ip.protocol)
-
- het controle getal van een ICMP pakket (icmp.cksum)
- de inhoud van de lading van de ICMP pakket (in blob) (met icmp.blob)
- of het pakket een geldige ICMP header bevat (icmp.hdr)
- of het pakket een ICMP pakket is (icmp.is)
- het type ICMP pakket, zoals Echo Reply, Destination unreachable...
- ..... enz .....

Aanvullende informatie over N-code is te vinde in de Advanced User’s Guide Guide op:http://www.cs.columbia.edu/ids/HAUNT/doc/nfr-4.0/advanced/ advanced-htmlTOC.html

IN toekomstige versies van deze papers zullen meer filters worden besproken, controleer regelmatig op nieuwe versies ;)

Standaarden

In deze sectie zal ik verschillende "standaarden" introduceren, en een aantal lijsten/overeenkomsten welke worden gedeeld door veel tool "experts".

CVE

CVE staat voor Common Vulnerabilities en Exposures (veel voorkomende zwakheden), welke niets meer is dan een lijst met zwakheden in beveiligingen. Op het eerste oog lijkt het misschien grappig, maar kan later van pas komen. Verschillende tools gebruiken verschillende termen voor gevonden zwakheden, door CVE te gebruiken kan een uniforme beschrijving verschillende zwakheden worden gegeven, welke voor iedereen begrijpbaar is. Daardoor is het niet langer nodig dezelfde tools als anderen te gebruiken.

CVE levert een naam voor een specifieke zwakheid/blootstelling en een uniforme (en gestandiseerde) beschrijving, en voorkomt zo miscommunicaties tussen gebruikers van verschillende systemen. CVE beschijft zwakheden (vulnerabilities) als "problemen die meestal worden beschouwd als zwakheden in de context van alle redelijke beveiligings beleiden" en exposures ("blootstellingen") als "problemen die

alleen overtredingen zijn van bepaalde beveiligings beleiden". In CVE is het verschil tussen zwakheid en blootstelling fundamenteel. Voorbeelden van zwakheid zijn, bijvoorbeeld, phf, door iedereen beschrijfbare password bestanden... Voorbeelden van blootstellingen zijn het gebruik van programma's die kunnen worden aangevallen met brute kracht (brute force) of het gebruik van services die in het algemeen aangevallen kunnen worden. Bij definitie en met deze voorbeelden zou het eenvoudig moeten zijn om onderscheid te maken tussen zwakheden en blootstellingen (in CVE). Het fundamentele verschil zou kunnen zijn dat de aanvaller de mogelijkheid heeft om commando's uit te voeren als een andere gebruiker of om bestanden te lezen/schrijven zelfs al zou dit niet mogelijk moeten zijn (door bestands permissies).
In contrast, stellen blootstellingen een gebruiker in staat om aanvullende informatie over het syteem (en de status) te verkrijgen, deze activiteiten lopen in de achtergrond... Blootstellingen komen voort uit slechte beveiligingsinstellingen welke "gerepareerd" kunnen worden. Zwakheden kunnen gezien worden als gaten in de beveiliging van de "normale" beveiliging (welke de mogelijkheid zou moeten bieden om het risico van potentiële aanvallers te beperken door controle op permissies). Deze "lijst" zou bijgehouden moeten worden, maar niet iedere zwakeheid of blootstelling wordt onmiddelijk "geaccepteerd". Nadat een zwakheid/blootstelling is gedetecteerd ontvangt het eerst een "candidate number" (dit gebeurt door de CNA - Candidate Numbering Authority). In aansluiting zal deze worden gepubliceerd op het bord (door de CVE Editor) en bediscussieerd of de zwakheid/blootstelling geaccepteerd moet worden. Als het bord besluit om de kandidaat niet te accepteren (op het moment) wordt de reden hiervoor gepubliceerd op de website. Als de kandidaat wordt geaccepteerd, wordt hij toegevoegd aan de lijst (en zo officieel onderdeel van de CVE). Nu zou het duidelijker moeten zijn dat iedere (potentiële) zwakheid eerst een "kandidaat nummer" krijgt, omdat het eerst bediscussieerd moet worden of de kandidaat wordt geaccepteerd, of niet. De zwakheid krijgt het 'kandidaat nummer' om deze te onderscheiden van de officiële ingangen in de lijst. Iedere kandidaat heeft 3 basis velden (de "identificeren" hem):

- Number (nummer)
- Description (beschrijving)
- References (referenties)

In deze context is het nummer de eigenlijke naam van de kandidaat, samengesteld uit "jaar van onstaan", een toegevoegd nummer, welke onthuld welke sequentiële kandidaat in een jaar het is:

```
CAN-Year - sequential candidate of the year
```

Zoals eerder aangegeven wordt een geaccepteerde kandidaat toegevoegd aan de lijst. Als een resultaat wordt het 'CAN-YEAR- Candidate number', 'CVE-Year-Candidate number'. Een voorbeeld: 'CAN-2001-0078' wordt 'CVE-2001-0078' in de lijst.

Dat is het, voor meer informatie, zie de officiële web pagina van CVE.

Voorbeelden

In dit laatste deel worden een paar IDS'en voorgesteld.

Snort

Omdat Snort erg bekend is en veel opties biedt, zal ik deze in meer detail beschrijven dan de andere IDS voorbeelden. In principe kan Snort worden gebruikt in een van drie modi: Sniffer, Packet Logger en Network Intrusion Detection System. In Sniffer mode genereerd Snort pakketten op de console, in Packet Logger mode logt hij ze op de harde schijf en de Network Intrusion Detecion mode analyseert pakketten. Ik zal me vooral op de laatste modus concentreren, maar hier is een korte introductie in de Sniffer en Packet Logger mode:
In Sniffer mode kan een variëteit van pakket informatie worden gelezen, zoals TCP/IP pakket header:

```
[Socma]$ ./snort -v
```

Als output krijgen we alleen de IP/TCP/ICMP/UDP header. Er is een groot aantal opties, een paar zullen hier besproken worden.

```
-d = geeft de pakket data
-e = geeft de Data Link Layer weer.
```

Packet Logger Mode:
In tegenstelling tot de Sniffer mode kan de Packet Logger mode pakketten loggen naar de harde schijf. We hoeven alleen een directory toe te wijzen waarnaar Snort moet loggen en het zal automatisch naar Packet Logger mode gaan:

```
#loggingdirectory must exist:
[Socma]$ ./snort -dev -l ./loggingdirectory
```

Met "-l" gebruikt Snort soms het adres van de remote computer als de directory (om logs naartoe te schrijven), soms wordt het lokale host adres gebruikt. Om te loggen naar het thuis netwerk specificeren we het thuis netwerk in de opdrachtregel:

```
[Socma]$ ./snort -dev -l ./loggingdirectory -h 192.168.1.0./24
```

Een andere mogelijkheid is om te loggen in TCP-DUMP formaat:

```
[Socma]$ ./snort -l ./loggingdirectory -b
```

Nu wordt het volledige pakket gelogged, niet alleen specifieke secties, dit elimineert de noodzaak van verdere opties. Het is mogelijk om programma's als tcpdump te gebruiken om de bestanden te vertalen naar ASCII tekst, maar Snort kan het ook zelf:

```
[Socma]$ ./snort -dv -r packettocheck.log
```

Network Intrusion Detection Mode: om naar NIDS mode te gaan, kunnen we een commando als het volgende gebruiken:

```
[Socma]$ ./snort -dev -l ./log -h 192.168.1.0/24 -c snort.conf
```

In dit geval is snort.conf het configuratie bestand. Het wordt gebruikt om Snort te laten weten waar het zijn "regels" kan vinden om uit te maken of het een aanval is, of niet, of de vraag is toegestaan... De regels, zoals gedefinieerd in snort.conf zullen worden toegepast op het pakket om deze te analyseren. Als er geen specifieke uitvoer directory was opgegeven wordt de standaar /var/log/snort gebruikt. De uitvoer van snort is afhankelijk van de alert modus - afhankelijk hiervan is de informatie wat vroeger of wat later beschikbaar.

Modus	Hoe/wat wordt weergegeven
-A fast	Tiid, Bron en Doel IPs/poorten, het waarschuwings Bericht
-A full	Standaard instelling
-A unsock	verstuurd de Waaarschuwingen naar een UNIX socket
-A none	Stopt de Waarschuwingen

Zoals we hebben gezien, kunnen we met -b loggen in binary mode, met -N wordt packet logging afgebroken. Maar dit is niet alles, zo kan Snort bijvoorbeeld berichten sturen naar syslog. Standaard instelling hiervoor is LOG_AUTHPRIV en LOG_ALERT. OM berichten naar syslog te sturen, geven we alleen "-s" op, voorbeeld volgt. Verder hebben we de mogelijkheid om berichten te sturen naar smbclient of Win-pop-up waarschuwingen naar een Windows computer. Om deze "feature" te gebruiken, moeten we "-enable-smbalerts" opgeven tijdens de configuratie van Snort.

```
[Socma]$ ./snort -c snort.conf -b -M MYWINWORKSTATION
```

Hierbij een voorbeeld van het gebruik van de alert modes:

```
[Socma]$ ./snort -b -A fast -c snort.conf
```

Behalve de beschreven opties, zijn er andere als de volgende:

```
-D = start Snort in daemon mode
-u usersnort= start Snort met UID 'usersnort'
-g groupsnort = start Snort met GID 'groupsnort'
-d = log ook data van de applicatie laag
```

Snort biedt veel opties, als je een probleem tegenkomt, kun je gewoon "snort -h" gebruiken, of op de mailing lijsten kijken of jouw probleem elders ook is opgetreden. De volgende sectie gaat over Snort regels, als het je niet uitmaakt om de bestaande regels te begrijpen, of je eigen regels te schrijven, kun je deze sectie overslaan. Zoals ik aangeef aan het eind van dit deel (over snort) is de Snort Users Manual te vinden op www.snort.org, het is onze echte bron voor dit.

Snort Regels:
Voor een beter begrip van Snort is het belangrijk de Snort Rules te kennen. Snort gebruikt soms specifieke variabelen welke gedefinieerd kunnen worden met 'var':

```
var: <name> <wert>      var

MY_NET [192.168.1.0/24,10.1.1.0/24]
alert tcp any any -> $MY_NET any (flags:S;msg: "SYN packet";)
```

Er zijn andere manieren om de variabele naam op te geven:

```
$variable = definieert de Meta variable
$(variable) = hier wordt de waarde van de variabele 'variable' opgegeven
$(variable:-default) = als 'variable' niet gedefinieerd is, wordt de
waarde 'default' gegeven.
$(variable:?msg) = geeft de waarde van de variabele 'variable' of, indien
niet gedefinieerd, het bericht 'msg'.
```

Als je al eerder met shell programmeren hebt gewerkt, zou het volgende niet je onbekend moeten voorkomen:

```
[Socma]$ shelltest=we
[Socma]$ echo hello $shelltestlt
hello
[Socma]$ echo hello ${shelltest}lt
hello world
```

De toepassing van $(variable)in Snort en ${variable} in de shell is identiek. Er bestaan ook andere equivalenten (of vergelijkbare termen) in shell programmering:

```
[Socma]$ shelltest = bash
[Socma]$ echo ${shelltest:-nobash}
bash
[Socma]$ echo ${notdefined:-nobash}
nobash
```

De toepassing van de term '$(variable:-default)' verschilt alleen in het feit dat de shell { en } gebruikt, in plaats van (en). De laatste term bestaat ook in shell:

```
[Socma]$ shelltest = bash
[Socma]$ echo ${shelltest:?"then csh"}
bash
[Socma]$ echo ${notdefinedvariable:?"not defined or nil"}
not defined or nil
```

Het doel van deze korte excursie was om "kennis te associ&eum;ren", ik was in staat om me de syntax van Snort sneller te herinneren door in gedachten te refereren aan de termen van de shell die ik me herinnerde.
Veel opdrachtregel opties kunnen gezet worden het configuratie bestand. hiervoor wordt 'config' gebruikt:

```
        config <aanwijzing> [: <waarde> ]
```

De belangrijkste 'aanwijzingen' zijn:
alertfile = veranderd het bestand waarin de waarschuwingen worden opgeslagen

daemon = start het proces als daemon (-D)
reference_net = zet het thuis network (-h)
logdir = zet de logging directory (-l)
nolog = schakel logging uit
set_gid = veranderd GID (-g)
chroot = chroot'ed in de opgegeven directory (-t)
set_uid = zet UID (-U)

Als je bijvoorbeeld het alertbestand wil wijzigen in "mylogs" kun je als volgt verder gaan:

```
config alertfile : mylogs
```

Terug naar de eigenlijke regels (hier een voorbeeld van deel van een ftp.rules bestand):

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP EXPLOIT overflow";\
flags: A+; content:"|5057 440A 2F69|";\
classtype:attempted-admin; sid:340;rev:1;)
```

In principe bestaan Snort regels uit twee delen: de regel header en de regel opties. De regel header informeerd over twee dingen:

- bron en doel IP addressen
- protocol
- de acties die ondernomen moeten worden door de regel

In de ftp regel boven de header is het volgende deel:

```
Actie         bron ip           doel ip
   |              |                |
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
        |                |               |
       Protocol    vanaf iedere poort   Poort
```

Zoals je kunt zien eindigt de regel header bij het eerste (en beginnen de regel opties hierna. Er zijn meerdere mogelijke acties (in dit geval 'alert') welke kunnen worden gestart als de Snort regels iets verdachts opmerken:

- alert = afhankelijk van welke alert methode er wordt gebruikt (standaard is 'alert full'), wordt er een waarschuwing afgegeven en het bijbehorende pakket gelogged
- log = alleen het pakket wordt gelogt
- pass = resultaten in het pakket worden genegeerd
- activate = genereert een alert en keert terug naar een andere dynamische Snort regel (zodadelijk meer hierover)
- dynamic = de regel blijft inactief tot hij wordt geactiveerd door een andere regel, waarna deze werkt als 'log' (zie hierboven).

Het tweede veld (protocol - hier tcp) specifieceert het protocol om te analyseren. Mogelijkheden zijn: tcp, udp, icmp en ip (in de toekomst kunnen dit er meer worden ... zoals ARP, GRE).
In combinatie met het volgende veld (bron ip) komen we vaak de ! operator tegen (negatieve operator).

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

Als resultaat van de negatieven operator wordt ieder pakket dat niet afkomstig is van $EXTERNAL_NET gelogd. Er zijn meer mogelijkheden om een aantal IP adressen op te geven, zoals een lijst van IP adressen. De adressen worden gescheiden door een komma en ingesloten door [ ].

```
alert tcp ![ip address,ip address] any -> ....
```

Een ander alternatief is het gebruik van "any", dat ieder IP adres bevat.

```
log tcp any any -> ...
```

Het laatste deel van de regel header is de specificatie van de poorten, in ons voorbeeld ftp. Het is niet alleen mogelijk een specifieke poort te monitoren, maar ook een specifiek bereik (meerdere poorten). Hier zijn de opties:

```
:portnumber                     -> alle poorten kleiner of gelijk
                                aan portnumber
portnumber:                     -> alle poorten groter of gelijk
                                aan portnumber
fromportnumber:toportnumber     -> alle poorten tussen fromportnumber
                                en toportnumber
```

Natuurlijk is het mogelijk om de negatieve operator te gebruiken, met de consequentie dat alle poorten worden gemonitord behalve de opgegeven, bijvoorbeeld .

```
!:21            -> alle poorten dien niet kleiner dan of gelijk
                      aan 21
```

Iets dat nog niet uitgelegd is maar al wel de hele tijd gebruikt is de richting (direction) operator "->".

```
"bron" -> "doel"
```

Echter, er is een andere variant <> :

```
"bron" <> "doel"
```

Dit betekend dat Snort ook de bron zal doorzoeken als het doel voor het adres.
Zoals ik al noemde, is er de stap ’activate’, deze genereerd een alert en geeft een dynamische Snort regel.
Als een specifieke regel zijn acties compleet heeft kan deze een andere regel activeren. In principe is het verschil tussen normale regels en "geactiveerde" regels het feit dat alleen een specifiek veld geactiveert moet worden: "activates". Dynamische regels, echter werken als logs (zie boven), verschil: "activate by" moet worden opgegeven. Een ander veld moet worden opgegeven: "count". Nadat "activate rule" zijn

werk heeft gedaan, wordt de dynamische regel geactiveerd, maar alleen voor "count" pakketten (oftewel, 40 pakketten als count = 40).

```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags : PA;  \
  content : "|E8C0FFFFFF|\bin|;activates : 1; msg : "IMAP buf!";)
dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by : 1; \
  count : 50;)
```

Sommige van de opties, zoals de regel opties, zijn nog niet geïntroduceerd, deze zal ik nu uitleggen, maar deze zullen later duidelijker worden. Merk de velden *activates* en *activated_by* op in ons voorbeeld hierboven. De eerste regel roept een dynamische regel aan nadat de eerste regels zijn werk heeft voltooid, dit wordt ook aangegeven door het statement activated_by = 1 van de dynamische regel.

Nu het tweede deel van de Snort regels: de regel opties. Laten we weer de eerste ftp.rule nemen:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP EXPLOIT
overflow"; flags: A+;\
content:"|5057 440A 2F69|";classtype:attempted-admin;\
sid:340; rev:1;)
```

De eerste regel optie in dit geval (de regel header stopt bij de eerste "):

```
 (msg:"FTP EXPLOIT overflow";\
flags: A+; content:"|5057 440A 2F69|";\
classtype:attempted-admin; sid:340; rev:1;)
```

Er zijn 34 sleutel woorden, ik zal mijn uitleg beperken tot de belangrijkste en/of meest voorkomende. Voor een overzicht van alle mogelijke sleutelwoorden, zie de Snort Users Manual.

msg - geeft de alarm berichten en logt ze in de Packet Logger Mode
logto - logt de pakketten in een specifiek bestand
dsize - vergelijkt de pakket grootte met een specifieke waarde v
flags - controleert de TCP flags voor specifieke waarden
content - zoek naar een specifiek patroon/string in een pakket
content-list - zoek naar een specifiek patroon/string in een pakket
nocase - maakt geen onderscheid tussen grote en kleine letters in gezochte strings
react - actieve reactie (blokkeert websites)
sid - Snort regel id
classtype - sorteert de potentiële aanvallen in groepen
priority - zet de gevoeligheid
Ok, maar hoe werken de individuele regels? msg:
We komen regelmatig ’msg’ tegen als we de regels doorlezen, deze optie is verantwoordelijk voor het genereren van alarmen en de logging er van.

```
   msg:"<text>";
```

t "<text>" is het bericht dat wordt geschreven/weergegeven in alertfile

logto:
Ieder pakket, waarvoor de regel van toepassing is, wordt gelogged in een specifiek bestand.

```
    logto: "<filename>";
```

In dit geval is "<filename>" het bestand waar de bestanden zullen zullen worden gelogged.
dsize:
Dit wordt gebruikt om de grootte van een pakket aan te geven. Als we de grootte van de buffer van een bepaalde service kennen kan deze optie gebruikt worden om te beschermen tegen een mogelijke buffer overflow. Vergeleken met ’content’ is het iets sneller, en daardoor meer gebruikt om te testen op buffer overflows.

```
        dsize: [>|<] <size>;
```

De twee optionele operators > en < geven aan dat de pakket grootte groter resp kleiner zou moeten zijn dan de opgegeven grootte:
Dit controleert welke flags zijn gezet. Op moment zijn er 9 flags beschikbaar in Snort:

```
F       FIN
S       SYN
R       RST
P       PSH
A       ACK
U       URG
2       Bit 2 assigned
1       Bit 1 assigned
0       no TCP flags set
```

Er zijn additionele logische operators om criteria op te geven voor het testen van flags

```
+ ALL flag     = Sla toe bij alle opgegeven flags (ook anderen).
* ANY flag     = Sla toe bij alle opgegeven flags.
! NOT flag     = Sla toe als de opgegeven flags niet zijn gezet.
```

Meestal wordt het sleutelwoord ’flags’ op deze manier gebruikt:

```
        flags: <Flag waarde>;
```

De gereserveerde bits kunnen gebruikt worden om ongebruikelijk gedrag te detecteren, bijvoorbeeld pogingen om de vingerafdrukken van de IP stack te achterhalen.
content:
Een van de meest gebruikte sleutelwoorden (behalve ’msg’) is ’content’. Het kan gebruikt worden om de lading van pakketten voor te doorzoeken op bepaalde inhoud. Als de opgegeven inhoud wordt aangetroffen, worden voorgedefinieerde stappen ondernomen tegen de gebruiker. Na de detectie van de inhoud in de lading van een pakket, zal de de rest van de Snort regels worden uitgevoerd. Zonder ’nocase’ toe te passen, zal er onderscheid worden gemaakt tussen hoofd- en kleine letters. De inhoud van de lading zal doorzocht worden op binaries en tekst. Binaire data komt tussen | | en wordt weergegeven als byte code. De byte code geeft binaire informatie weer in de vorm van hexadecimale nummers. De negatieve operator (!) kan in deze context worden gebruikt, zo kunnen we

bijvoorbeerbeeld een alarm starten als een pakket een bepaalde tekst bevat.

```
content: [!] "<content>";
```

Het gebruik van ! is niet verplicht

```
 alert tcp any any -> 192.168.1.0/24 143 \
(content: "|90C8 C0FF FFFF|/bin/sh";\
msg: "IMAP buffer overflow";)
```

Zoals gedemonstreerd in deze regel, is de binaire data opgenomen tussen ||, hiermee kan ook normale tekst worden gebruikt. Zie ook de beschrijvingen van ’offset’ en ’depth’ in de Snort User Manual, deze worden vaak gebruikt in context met ’context’.

content-list:

Dit sleutelwoord werkt vergelijkbaar met ’content’, met het verschil dat er een aantal strings, om de pakketten op te doorzoeken, kunnen worden opgegeven in een bestand. Dit bestand, met de woorden waarnaar gezocht wordt, wordt gespecificeerd met ’content list’. Hiermee moet je in gedachten houden dat de strings verticaal worden uitgelijnd (iedere string op een eigen regel), bijvoorbeeld

```
"kinderporno"
"warez"
.....
```

In aansluiting, kunnen we - met behulp van ’ content-list: [!] "<bestand>" ’ dit bestand doorzoeken. Natuurlijk is de ! optioneel, het heeft hetzelfde effect als in ’content’...

nocase:
Deze regel speelt een belangrijke rol in context met ’content’ sleutelwoorden. Normaal gesproken worden hoofdletters gerespecteerd, met behulp van ’nocase’ wordt het onderscheid genegeerd:

```
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root";\
 nocase; msg: "FTP root user access attempt";)
```

Zonder het gebruik van ’nocase’ zou alleen gezocht worden naar ’USER root’, maar daar ’nocase’ was aangegeven, wordt er geen onderscheid gemaakt tussen hoofd- en kleine letters.
Als het doorzoeken van een pakket op specifieke content (met ’content’ of ’content_list’) resulteerd in een hit, kan ’react’ gebruikt worden voor de reactie. Zo kunnen bijvoorbeeld bepaalde pagina’s, opgevraagd door de gebruiker, geblokkeerd worden (porno pagina’s...). Door Flex Resp toe te passen kunnen verbindingen afgebroken worden of waarschuwingen verstuurd naar de browser. De volgende opties zijn mogelijk/legaal:

block - verbreekt de verbinding en stuurt een waarschuwing.
warn - stuurt een zichtbare waarschuwing (binnenkort beschikbaar)
Deze ’basis argumenten’ kunnen compleet gemaakt worden met additionele argumenten (zogenoemde

’additionele modifiers’):

msg - de text die verstuurt moet worden naar de gebruiker, als ’msg’ wordt gebruikt.
proxy : <portnummer> - de proxy wordt gebruikt om de notificatie te sturen (binnekort beschikbaar)

```
react : <basic argument [, additional modifier ]>;
```

Het ’react’ sleutelwoord wordt toegevoegd aan het eind van de regel opties, en kan als volgt gebruikt worden:

```
alert tcp any any <> 192.168.1.0/24 80 (content-list: "adults"; \
  msg: "This page is not for children !"; react: block, msg;)
```

sid:
’sid’ of Snort rules IDentification wordt gebruikt om ’speciale’ Snort regels te identificeren. Dit maakt het mogelijk/legaal voor "uitvoer plugins" om iedere regel te identificeren. Er zijn een aantal sid groepen:

```
< 100 = gereserveerd voor toekomstig gebruik
100-1 000 000 = regels die met Snort komen
> 1 000 000 = gebruikt voor lokale regels

sid-msg.map bevat een mapping van msg tags voor sid’s.
Het wordt gebruikt voor post processing om een waarschuwing toe te
wijzen aan een id.
sid:  <snort rules id>;

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80  \
 (msg: "WEB-IIS file permission canonicalization"; uricontent:\
 "/scripts"..%c1%9c../"; flags: A+;nocase;sid: 983;rev:1;)
```

classtype:
Met ’classtype’ kunnen we aanvallen in verschillende groepen onderbrengen. In de regels kunen we de prioriteit van een potentiële aanval vaststellen, alsook in welke groep hij thuishoort. Regels die een categorie hebben toegewezen gekregen in het configuratie bestand zullen automatisch een standaard prioriteit krijgen toegewezen.

```
classtype: <class name>;
```

Categoriëen van regels worden gedefinieerd in classifcation.config. De volgende syntax wordt gebruikt:

```
config classificatie: <class naam>,<class beschrijving>,\
      <standaard prioriteit>
```

In de volgende paragraaf - de beschijving van ’priority’ zullen we bekijken welke aanvals groepen er zijn.
In the following paragraph - the description of ’priority’ - we will learn what kind of groups of attacks there are.
priority:
Dit sleutelwoord wordt gebruikt om "beveiligings prioriteiten" aan onze regels toe te wijzen, om aan te

geven hoeveel schade een potentiëe aanval zou kunnen aanrichten. Hoe hoger de prioriteit hoe hoger het beveilings risisco. In vergelijking met de eerder uitgelegde ’class types’, zijn de prioriteiten makkelijker te begrijpen:

```
Class type                    Omschrijving                Prioriteit
-----------------------------------------------------------------
 not-suspicious               Alle "niet-verdachte" verkeer        0
 unknown                      onbekend verkeer                     1
 bad-unknown                  Potentiëel "slecht" verkeer     2
 attempted-recon              Poging tot Informatie Lek            3
 successful-recon-limited     Informatie Lek                       4
 successful-recon-largescale  Grootschalig Informatie Lek          5
 attempted-dos                Poging tot DoS-aanval                6
 successful-dos               Succesvolle DoS-aanval               7
 attempted-user               Poging om user privileges te krijgen 8
 unsuccessful-user            Onsuccesvolle poging om user         9
                              privileges te krijgen
 successful-user              Successvolle poging om user
                                     privileges te krijgen
 attempted-admin              Poging om admin privileges te krijgen 10
 successful-admin            Succesvolle poging tot admin privileges
```

Zoals eerder genoemd, representeren hogere prioriteiten hogere risico’s. Een gebruiker die admin privileges krijgt zou de meest serieuze aanval zijn.

```
 alert tcp any any -> any 80 (msg: "WEB-MISC phf Versuch";\
        flags: A+;content: "/cgi-bin/bash";priority:10;)
```

Het onderwerp van ’regels’ is -toegegeven- complex maar niet zo heel erg moeilijk. Bestuur een paar regels, lees de manual waar ze voor zijn en naar een tijdje zal het ’regel monster’ duidelijk worden ;) Bronnen voor Snort en documentatie zijn te vinden op http://www.snort.org. Hier zijn een aantal waardevolle .pdf’s te vinden, zoals de Snort USer Manual, welke de belangrijkste bron is voor deze beschrijving.

LIDS

Sinds Stealth’s paper (en bronnen) en de LIDS-Hacking-HOWTO weten we dat LIDS de beveiliging de beveiliging van een computer niet echt bevorderd, maar in sommige gevallen eerder een root kit is ;)

Laten we eens kijken naar het concept en dan naar de (denkbeeldige) kracht en zwakheid van LIDS. Het was ontwikkeld om - bijvoorbeeld - belangrijke systeem bestanden te beschermen en bepaalde processen te verbergen voor de gebruiker. In aanvulling, het zou niet moeten toestaan om modules toe te voegen, de nodige modules worden geladen tijdens het starten van het systeem. Eerder noemde ik al de LIDS-Hacking-HOWTO en Stealth’s paper, welke beide de werking en zwakheden van LIDS bespreken, ik zal mezelf beperken tot de belangrijkste features. Aan het einde van deze sectie zul je links aantreffen aar beide teksten.

De belangrijkste taak van LIDS is het beschermen van het bestandssysteem. Om in staat te zijn belangrijke bestanden/directories te beschermen worden deze gesorteerd in groepen:

- Read Only = Dit bestand/directory heeft alleen lees toegang, veranderingen zijn niet toegestaan
- Append Only = Het is alleen toegestaan om inhoud "toe te voegen" aan het bestand
- Exception = Deze bestanden worden niet beschermd
- Protect (un)mounting = Staat iemand toe (of niet) om een bestandssysteem te (un)mounten

Voor "echte" bescherming worden sommige system calls gemanipuleerd om zeker te weten dat de beschermingen worden volgehouden (bijvoorbeeld sys_open(), sys_mknod(), ...)
Verder, voorkomt LIDS dat specifieke processen gestopt (of zichtbaar) kunnen worden. Doel van de procedure is om te voorkomen dat de aanvaller specfieke processen ziet die hem zouden kunnen monitoren. Een ’ps -ax’ call zou ook niet onze processen moeten weergeven. Om een proces waarlijk te verbergen, wordt het gemarkeerd als ’PF_HIDDEN’. In het geval dat ps werkt om informatie te geven over de processen zal dit verhinderd worden doordat de processen zijn gemarkeert als ’PF_HIDDEN’. Dit op zich is niet voldoende om een proces veilig te verbergen omdat het nog steeds een entry heeft in het /proc bestandssysteem. LIDS is in staat om die functie te manipuleren om te voorkomen dat het proces opduikt in de /proc directory. Behalve dit is het mogelijk om de privileges van een proces te beperken. Als, bijvoorbeeld CAP_CROOT is gezet op 0, zal het proces geweigerd worden gebruikt te maken van chroot (zie /usr/src/linux/include/linux/capatibilites.h).
In aansluiting heeft LIDS de mogelijkheid om te draaien in een of twee beveiligings opties: ’security’ of ’none_securtity’. De globale variabele ’lids_load’ wordt hiervoor gebruikt. De standaard waarde is ’1’, dit betekend dat LIDS draait in ’security’ mode - De restricties zijn dus ingeschakeld. Als ’security=0’ is gezet tijdens de start (LILO prompt) wordt ’none_security’ geactiveerd. Als een resultaat worden alle beveiligings controles, restricties ... gedeactiveerd. Met ’lids_load=0’ functioneerd de computer alsof LIDS niet was geïnstalleerd. Een andere mogelijkheid is om de beveiligings optie te veranderen is het programma ’lidsadm -S’ online, hiervoor dient een wachtwoord opgegeven te worden.

LIDS biedt ok de mogelijkheid om firewall regels te beschermen, dit kan door CONFIG_LIDS_ALLOW_CHANGE_ROUTES te activeren en CAP_NET_ADMIN uit te schakelen. Als iemand de firewall regels wil veranderen, dient CAP_NET_ADMIN geactiveerd te worden om te voorkomen dat iedereen de regels kan veranderen. Ook kan deze Sniffer worden uitgeschakeld en een poort scanner kan geïntegreerd worden in de kernel ...
kernel. .

LIDS biedt ook verschillende "respons opties" (zie sectie over respons), bijvoorbeeld om waarschuwingen te sturen via pager of SMS naar de beheerder.

Er zijn vele opties, Stealth legt dit uit in zijn paper over het misbruiken van LIDS: http://www.securitybugware.org/Linux/4997.html. De LIDS Howto kan gevonden worden op: http://www.lids.org/lids-howto/lids-hacking-howto.html.

COLOID

COLOID is het acronym voor Collection of LKMs for Intrusion Detection, en is een tijdje geleden door mij opgestart.

Toen het duidelijk werd dat delen van het project niet goed werkten en niet up-to-date waren, is het project tijdelijk gestopt. Echter, ik zou graag nog even uitweiden over de oorspronkelijke plannen: Met de eerste module (prev_exec) wilden we - onder meer - tijdelijk de uitvoering van bepaalde binaries blokkeren (in mijn sourcecode gebruikte ik de GNU compiler GCC), op specifieke tijden. Het is gedefinieerd in de source wanneer een uitvoeren verboden zou moeten zijn. - de tijd kan tot op de minuut worden aangegeven. Als een gebruiker probeert in die tijd GCC te starten, zal dit worden geblokkeerd, GCC wordt niet gestart. Zou een gebruiker GCC op een "toegestane" tijd starten, zullen de argumenten worden gecontroleerd en een zoektocht naar .c bestanden vindt plaats. Het doel van deze procedure is het doorzoeken van de source (iemand wil compileren) op "gevaarlijke functies". Defaults waren ’scanf’ en ’strcpy’ ... enz. Het is mogelijk om meer functies toe te voegen. Zoals ik eerder aangaf, doorzoekt LKM de source en als het een of meer van de functies aantreft, wordt de uitvoering van GCC geblokkeerd. Aansluitend wordt er een entry in een log bestand weggeschreven en een ’biep’ gegenereerd.

Tot zo ver werkte de module, maar het concept was niet algemeen genoeg en kon eenvoudig omzeilt worden.

De tweede module was ’anom_detection’ welke de Anomaly Detection gebruikt die eerder genoemd is. Eigenlijk behoren deze twee LKMs tot deze sectie:
1) Anomaly_Detection.c welke de database genereerd van normale gebruikers activiteiten
en
2) Misuse_Detect.c welke het gedrag van de gebruiker controleerd (door de database als benchmark te gebruiken) op afwijkingen ten op zichte van het normale gedrag (gelogged in de database).

Het plan was om LKM de volgende criteria te laten controleren:

Hoe vaak voerde de gebruiker de volgende commando’s uit:

- su
- login
- chmod
- chown
- insmod
- ps
- lsmod
- rm
- last
- lastlog
- ftp

Wanneer voerde de gebruiker de volgende commando’s uit:

- login
- su

Of wanneer heeft de gebruiker de PC normaal gestart en wanneer begon hij een shutdown
Overig:
Hoe vaak (probeerde) hij de volgende bestanden te openen:

- /etc/passwd
- /etc/group
- /etc/shadow
- /etc/ftpusers
- /etc/ftpgroups
- /etc/ftpaccess
- /etc/hosts.allow
- /etc/hosts.deny
- /etc/inetd.conf
- .....

Hoe vaak voerde hij programma's uit met de SUID bit set?
We moeten goed opletten welke bestanden we monitoren (hoe vaak opende hij ze). Als we teveel bestanden kiezen, kost dit te veel performance, normaal werk is dan nauwelijks nog mogelijk.

Er bestonden andere, kleinere LKMs zonder complete source, de source of de twee bovengenoemde modules is beschikbaar op mijn web pagina, mogelijk wil iemand hier iets mee doen ....

Ter afsluiting

Zou iemand nog meer ideeën hebben over de inhoud van deze paper, ik ben bereikbaar op: Socma(Q)gmx.net . Voor verdere stimulatie, lofbetuigingen, kritiek... stuur me een email
Referenties (behalve degene genoemd in de tekst):

1. http://online.securityfocus.com/infocus/1524
2. http://online.securityfocus.com/infocus/1534
3. http://online.securityfocus.com/infocus/1544
4. http://online.securityfocus.com/infocus/1232
5. http://www.entercept.com/products/entercept/whitepapers/downloads/systemcall.pdf
6. http://www.computec.ch/dokumente/intrusion_detection/
   angriffsmoeglichkeiten_auf_okenas_stormwatch/
   angriffsmoeglichkeiten_auf_okenas_stormwatch.doc

Site onderhouden door het LinuxFocus editors team © Klaus Müller "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Vertaling info: de --> -- : Klaus Müller <Socma(at)gmx.net> de --> en: Jürgen Pohl <sept.sapins(Q)verizon.net> en --> nl: Guus Snijders <ghs(at)linuxfocus.org>

2005-01-14, generated by lfparser_pdf version 2.51