IDS – Intrusion Detection System, part II

by Klaus Müller
<Socma(at)gmx.net>

About the author:
Klaus Müller a.k.a. 'Socma' ist momentan noch Schüler und beschäftigt sich v.a. mit Linux Programmierung und sicherheitsrelevanten Themen.

Abstract:

Während sich der erste Teil v.a. auf die typischen Angriffe gegen Intrusion Detection Systeme konzentrierte, wird dieser zweite Teil Methoden vorstellen, mit denen ihr diese Angriffe entdecken und wie ihr darauf reagieren könnt. Dazu zählen z.B. die Verwendung von Signaturen und Filtern. Abschließend wird noch Snort und LIDS vorgestellt.

Analysemöglichkeiten

Es wurde bereits besprochen, gegen welche Attacken sich IDSs schützen sollten und welche verschiedenen Systeme man unterscheidet. Jetzt aber zur Art und Weise der Analyse, also wie ein IDS bestimmt, ob ein Angriff vorlag oder nicht, bzw. ob der Angriff erfolgreich war oder nicht.

Grundsätzlich unterscheidet man hier Misuse Detection und Anomaly Detection. Bei der Misuse Detection werden bestimmte Muster definiert, die einen Angriff als solchen enttarnen. Diese Muster werden auch Signaturen genannt, und werden in einem extra Teil noch intensiver besprochen. Für jetzt reicht es zu wissen, dass man z.B. Signaturen definieren kann, die im Netzwerk−Verkehr nach bestimmten Strings suchen (z.B. /etc/passwd) und gegebenfalls die "Anfrage" nach dieser Datei verbieten und Alarm auslösen. Der Vorteil der Misuse Detection besteht darin, dass die Wahrscheinlichkeit von false alarms relativ gering ist, da man anhand von Signaturen recht genau einstellen kann, nach welchen Dingen gesucht werden soll.... Der Nachteil ist allerdings auch recht offensichtlich, denn neue Attacken werden nicht sehr häufig erkannt, da sie nicht definiert sind (siehe Kapitel über Signaturen). Die andere Möglichkeit stellt die sogenannte Anomaly Detection dar. Das bedeutet einfach nur, dass ein Profil von normalen Useraktivitäten erzeugt wurde und wenn das Verhalten des Users zu sehr von diesem Profil abweicht, ein Alarm ausgelöst wird.

Erster Schritt bei dieser Analyse stellt also das Erstellen eines Profils (einer Datenbank) "normaler" Useraktivitäten dar. Hierbei können die verschiedensten Dinge protokolliert werden : Wie oft führt er bestimmte Kommandos aus ? Wann führt er bestimmte Kommandos aus ? Wie oft öffnet er bestimmte

Dateien ? .... Hier ein kleines Beispiel : – User 'beispiel' führt durchschnittlich 3 mal täglich /bin/su aus (dieser Wert stünde im Profil) Nun kommt der User 'beispiel' und führt an einem Tag auf einmal 7 mal su aus, also mehr als doppelt so oft wie sonst. Anomaly Detection würde dieses "anormale" Verhalten enttarnen und den Admin warnen, dass User 'beispiel' plötzlich 7 mal su ausgeführt hat, dabei ist der protokollierte , "normale" Durchschnitt bei 3. Die Nachteile dieses Verfahrens wurde mir v.a. klar, als ich selber mit der Umsetzung (siehe Beispiele am Ende – COLOID) begann, denn die Methode, um eine Datenbank normaler Useraktivitäten zu erzeugen, ist recht rechenintensiv. Beobachtet man z.B., wie oft der User 10 Dateien geöffnet hat, so müsste bei jedem open() überprüft werden, ob es sich um eine der 10 Dateien handelt, und falls ja der entsprechende Counter hochgezählt werden. Dennoch stellt es eine sehr gute Möglichkeit dar, auch neue Angriffstechniken aufzudecken, da diese wahrscheinlich als "anormal" gelten werden. Außerdem kann der Admin schließlich auch selbst definieren, welche Abweichung als "anormal" gelten soll, z.B. eine Abweichung von 10% oder erst eine Abweichung von 75%.... Bei der Verwendung dieser Methode müsst ihr allerdings aufpassen, dass die Erstellung des Userprofils in einem "sicheren" Netzwerk abläuft, da sonst das Verhalten des Angreifers als normal gilt und das Verhalten des richtigen Users als anormal.

Allgemein umfasst die Anomaly Detection folgende Dinge:

- Threshold Detection = dieser Bereich benutzt v.a. Counter, die zählen wie oft was ausgeführt, geöffnet, gestartet ... wurde. Diese statische Analyse kann noch durch die sogenannte Heuristische Schwellenanalyse erweitert werden.
- Rule–Based Detection = hier werden bestimmte Regeln definiert, sollte die Verwendung von diesen Regeln abweichen wird ein Alarm ausgelöst.
- Static Measure = das Verhalten des Users/Systems entspricht einem Muster, das entweder vordefiniert wurde oder sonst wie erstellt wurde. Oft wird hier ein Programm/LKM... eingefügt, das normale Benutzeraktivitäten protokolliert und so das Muster definiert

Heuristische Schwellenwertanalyse heisst hier, dass der Counter (wie oft etwas ausgeführt werden darf) nicht von Anfang an statisch festgelegt wird, sondern dynamisch. Führt ein User normalerweise /bin/login 4 mal aus, so würde der Counter evtl. auf 5 gesetzt....
Eine Untergruppe der Anomaly Detection stellt die Protocol Anomaly Detection dar, eine relativ neue Technik, die vom Prinzip her genauso funktioniert wie die Anomaly Detection. Jedes Protokoll hat ein "vordefiniertes" Verhalten (siehe entsprechende) RFC's , Ziel der Protocol Anomaly Detection ist es, festzustellen, ob das Verhalten des Protokolls wie vordefiniert ist oder nicht. Es basieren mehr Angriffe auf Protocol Misuse als man vielleicht denken mag, daher ist diese Untergruppe durchaus wichtig für IDSs. Bei einem Blick zurück in den Teil über Scanning, lassen sich sicherlich einige
Ansätze zur Protocol Anomaly Detection finden.

- Überprüfen, ob kein Flag gesetzt ist (NULL Scan)
- Überprüfen, ob alle Flags gesetzt sind (XMas Scan)
- Überprüfen, ob "unsinnige" Kombinationen von Flags gesetzt sind, wie SF
- .....

In den entsprechenden rfc's findet ihr die richtigen Spezifizierungen, auch welches Verhalten nicht eintreten sollte, bzw. welches Verhaltenn bei einem bestimmten Ereignis auftreten sollte. Desweiteren gibt es noch die Application Anomaly Detection (eigentlich kommt es Application Based IDSs nahe). In einigen Texten las ich Ansätze in diese Richtung, so dass ich mich auch damit beschäftigt habe. Natürlich hat auch ein Programm ein "normales" Verhalten, d.h. wie reagiert es auf Ereignis X .. Y , was macht es wenn der User eine fehlerhafte Eingabe macht, ... Oft werden bestehende Binaries (v.a. ps, netstat...) durch eigene ersetzt, um (im Falle von ps) bestimmte Prozesse zu verstecken. Mittels Application Anomaly Detection könnte man nun evtl. "anormales" Verhalten des Programms erkennen. Einige Application Based IDSs funktionieren sicherlich in diese Richtung, doch mit dieser Art der IDSs habe ich am wenigsten Erfahrung .

Abschließend noch zu einer neuen Methode der ID−Systeme:Intrusion Prevention. Intrusion Prevention wird von einigen neuen ID−Systemen eingesetzt und unterscheidet sich von den bisher besprochenen Methoden der Intrusion Detection. Anstatt durch die Analyse von Logfiles / des Traffics.... Angriffe später zu entdecken, wird versucht, Angriffe gar nicht erst zuzulassen.

Im Gegensatz zu klassischen IDSs werden hier keine Signaturen benutzt, um Angriffe zu erkennen. Im folgenden wird kurz erläutert, was IPSs machen, die Funktionsweise sollte anhand von kleinen Beispielen verständlich sein:

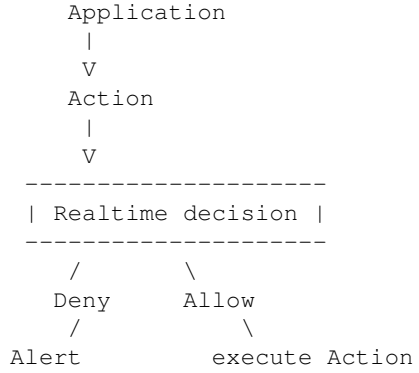
- Monitor application behavior
- Create application rules
- Alert on violations
- Correlate with other events
- System Call Interception
- .....

'Monitor application behavior' kommt den Application−Based−IDSs nahe,d.h. das Verhalten einer Applikation wird untersucht (und protokolliert), so z.B. auf welche Dateien es normalerweise zugreift, mit welchen Programmen es interagiert, auf welche Ressourcen es zurückgreift....Ähnlich wie bei der Anomaly Detection wird also erst einmal versucht herauszufinden, was ein Programm normalerweise macht, bzw. was es machen darf.

Der dritte Punkt ('Alert on violations') sollte eigentlich keiner Erklärung bedürfen, es heisst nämlich nur, dass bei Abweichungen (sprich bei Erkennung einer Attacke) ein Alert ausgelöst werden kann. Dies kann bedeuten, dass einfach "nur" geloggt wird oder aber das Resourcen gesperrt werden....

Im zweiten Schritt ('Create application rules') wird auf den Informationen, die auf den Recherchen in Teil 1 ('Monitor application behavior') beruhen, ein sogenanntes Application Ruleset aufgestellt. Dieses Ruleset gibt Auskunft darüber, was eine Applikation machen darf (auf welche Ressourcen es zurückgreifen darf...) und daraus folgt dann auch, was eine Applikation nicht machen darf.

'Correlate with other events' bedeutet, dass man andere Sensoren....über die Vorgänge informiert, durch das Zusammenarbeiten der Sensoren...kann so besser sichergestellt werden, dass Angriffe nicht zugelassen werden.



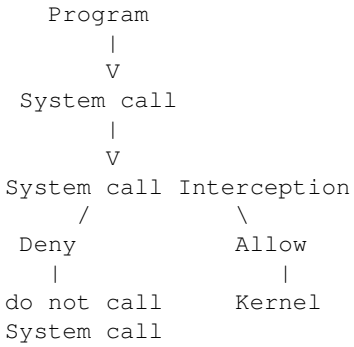
Dieses vereinfachte Schema soll lediglich noch einmal die Vorgänge verdeutlichen. Bevor eine Aktion

durchgeführt wird, wird erst einmal eine 'Realtime decision' durchgeführt (d.h. die Aktion mit dem Application Ruleset verglichen). Sollte die Aktion nicht gestattet sein (z.B. weil das Programm auf Dateien zugreifen will / ändern will, obwohl es eigentlich gar nicht auf diese Systemdateien zugreifen darf), wird ein Alert erzeugt. In den meisten Fällen werden die anderen Sensoren (oder eine zentrale Konsole) davon auch noch informiert. Dadurch soll verhindert werden, dass andere Rechner im Netzwerk bestimmte Dateien ausführen/öffnen... Sollte die Aktion allerdings dem Ruleset entsprechen, wird es genehmigt und die entsprechende Aktion letztendlich ausgeführt...

Nun aber zum vorerst letzten Punkt, der 'System call interception'. Manipulierte Systemaufrufe (durch z.B. sog. Rootkits) sind heutzutage immer öfter vorzufinden. Der Ansatz bei der Interception der Systemaufrufe ist recht einfach : Bevor ein Systemaufruf "zugelassen" wird, wird er erst einmal genauer überprüft. Überprüfung heisst z.B., dass man folgende Fragen checkt (siehe auch [5]):

- wer hat den Systemcall aufgerufen (welches Prog...) ?
- unter welcher User – Autorität läuft der Prozess (root...) ?
- auf was versucht der Systemcall zuzugreifen ?

Somit kann auch hier kontrolliert werden, ob versucht wird wichtige Konfig/Systemdateien zu ändern, da man "lediglich" überprüfen muss, ob der Systemcall dem vordefinierten Ruleset entspricht oder nicht.



Intrusion Prevention ist verglichen mit anderen Methoden eine recht neue Methode, so dass es in Zukunft sicherlich auch noch mehr Informationsmaterial zum Thema geben wird.

Zum Abschluss sei v.a. noch auf OKENA verwiesen, ein sehr leistungsfähiges IPS. In der Whitepapers Sektion unter www.okena.com findet ihr zusätzliches Material zu StormWatch... Wer sich für die Schwächen von StormWatch interessiert, sollte [6] lesen.

Signaturen

Hier werden nun die Signaturen der IDSs behandelt, im zweiten Unterteil dann noch deren Schwächen...

Konzept

Anhand von Signaturen können bekannte Angriffe erkannt werden, eine Signatur guckt also nach einem bestimmten Muster im Verkehr. Dieses Muster können verschiedenste Dinge sein, z.B. Strings oder auffällige Header (mit anormalen Flagkombinationen), Ports die dafür bekannt sind, dass sie von Trojanern benutzt werden.... Die meisten Angriffe haben gewisse Merkmale, z.B. dass eben bestimmte Flags gesetzt sind oder dass bestimmte Strings im Payload enthalten sind, durch Signaturen wird nun versucht, anhand von diesen Merkmalen einen Angriff zu entdecken.

Beginnen möchte ich mit den sog. Payload Signaturen. Hier zur Anschauung einmal ein Teil des Payloads eines Pakets:

```
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...
```

Wie ihr später in der Beschreibung zu den Snort−Rules sehen werdet, bestehen hier einige Möglichkeiten, was man machen kann. Häufig wird der Inhalt des Payloads nach bestimmten Strings durchsucht (in Snort mit 'content' oder 'content−list'), möchte jemand z.B. eine Passwortdatei (z.B. /etc/passwd) ziehen, so besteht die Möglichkeit, den Payload zu durchsuchen (nach /etc/passwd) und falls das Paket diesen String enthält, Maßnahmen dagegen einzuleiten, z.B. so :

```
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;)
```

Eine etwas andere Möglichkeit besteht darin, alle Pakete zu melden, die nicht einen bestimmten String enthalten.

Ein weiterer Ansatz besteht darin, die Größe der Pakete (an bestimmte Ports) zu überwachen, um vor möglichen Buffer Overflows zu schützen. Allgemein betrachtet besteht also die Möglichkeit, den Source Port und Destination Port zu bestimmen und Zugriffe von bestimmten Ports, bzw. auf bestimmte Ports ganz zu unterbinden. Allgemein betrachtet, gehören string signatures zu den payload signatures. Bei den Payload Signatures handelt es sich um Signaturen, die den Payload eines Pakets untersuchen, wie z.B. bei den string signatures der Payload nach einem bestimmten String durchsucht wird.
Doch was kann man noch anhand von Signaturen entdecken ? Nur den Payload nach bestimmten Strings durchsuchen, ist sicherlich nicht immer das "berauschenste". Eine andere Möglichkeit besteht darin, die Signatur nach den Flagkombinationen des TCP Headers gucken zu lassen. Sollte in einem Paket sowohl das SYN als auch das FIN Bit gesetzt sein, so liegt hier eine Anormalität vor, die vom Angreifer dazu genutzt werden könnte, bestimmte Eigenschaften des Betriebssystems herauszufinden (oder das Betriebssystem selbst herauszufinden). Wie anfangs bereits erwähnt, gibt es auch bestimmte Ports, die dafür berühmt sind, dass sie von Trojanern genutzt werden. Beispiele für solche Ports wären 31337 oder 27374.

Vielleicht wird es verständlicher, wenn ich das ganze an einem Beispiel erkläre. Für dieses Beispiel schauen wir uns einmal typische Merkmale eines synscan−Angriffs an:

- verschiedene Source IPs
- TCP Src und Dest Port ist 21
- Type of service 0
- IP id 39426
- SF gesetzt (SYN und FIN)
- verschiedene Sequenznummern gesetzt
- verschiedene Acknowledgement Nummern gesetzt
- TCP Window Größe von 1028

Ziel einer Signatur sollte es in einem solchen Falle sein, "normale" Merkmale einer Verbindung von "annormalen" zu unterscheiden. Manche IDSs besitzen auch extra Datenbanken, die eben solche Informationen wie oben gegeben enthalten, dann wird auf Übereinstimmung überprüft.
Prinzipiell kann man für das synscan−Beispiel schon Anormalitäten feststellen, die sich per Signatur überprüfen lassen:

- Source und Destination Port sind 21 (File Transfer Protocol − ftp) .
- Gleiche Source und Destination Port Number bedeuten nicht zwangsläufig, dass eine Attacke vorliegt, nur ist es wahrscheinlich.
- SF gesetzt, wie oben bereits gesagt, dürfte dieses nicht vorkommen, da man keine Verbindung anfordern und direkt wieder beenden kann
- Acknowledgement Nummer ist ungleich 0, obwohl nur SF und nicht ACK gesetzt. Wenn ACK nicht gesetzt ist, sollte die Acknowledgement−Nummer auf 0 stehen
- IP ID ist immer 39426, obwohl (nach RFC) diese Nummer nicht konstant bleiben dürfte, muss aber nicht zwangsläufig auf einen synscan−Angriff hindeuten, genauso wie die konstante Window Größe...

Bei der Entwicklung einer Signatur, zum Erkennen eines synscan−Angriffs, muss man nicht nur die oben genannten Merkmale des Angriffs beachten. Ziel der Signatur sollte es sein, sowohl alte Versionen als auch zukünftige Versionen dieses Angriffs zu erkennen. Deshalb sollte man sowohl möglichst allgemeine Merkmale mit speziellen Merkmalen des Angriffs verbinden, so dass die Wahrscheinlichkeit einer Erkennung steigt. Zwar könnt ihr für jede neue Version eines Angriffs eine neue Signatur schreiben, allerdings wäre man dann wohl den ganzen Tag damit beschäftigt, neue Signaturen zu schreiben, anstatt sich mit anderen, wichtigeren Dingen zu beschäftigen. Darum solltet ihr darauf achten, dass man mit der Signatur möglichst viele Angriffe (und Versionen) erkennt, ohne jedes Mal die Signatur ändern zu müssen.

Grundsätzlich sollte man sowohl Signaturen zur Erkennung von bestimmten Attacken als auch allgemeine Signaturen (z.B. zur Erkennung von Anormalitäten) schreiben. Ein Beispiel für eine mögliche Signatur, die einen speziellen Angriff entdeckt, könnt ihr oben sehen (oben war es der Angriff von synscan). Eine mögliche allgemeine Signatur könnte z.B. nach folgenden Dingen gucken:

- acknowledgement nummer ungleich 0, obwohl ACK nicht gesetzt
- anormale Flagkombinationen im TCP Header (SYN und FIN) oder andere (siehe Beschreibung der Scans)
- ....

Signaturen, die allgemein nach Anormalitäten von Protokollen suchen, werden im allgemeinen als Protocol

analysis based signatures bezeichnet, während die andere Gruppe als "Packet Grepping" bekannt ist.

Schwächen

Zwar scheint die Methode der Payload Signatures (also auch die der string signatures) recht "sicher", allerdings gibt es Möglichkeiten, diese zu umgehen. Evtl. schreibe ich mal ein Paper dazu, wie man Snort Rules umgehen kann, daher beschränke ich mich hier auf das Wesentliche. Eine Signatur, wie die, die ihr dort oben sehen könnt, guckt nach "/etc/passwd", durch nocase wird Groß/Kleinschreibung nicht beachtet. Doch was ist, wenn wir nicht auf direktem Wege die /etc/passwd saugen, sondern über einen Umweg ? Würde der Angreifer stattdessen '/etc/blabla/shit/../../.\passwd ' nehmen, würde die Signatur dann immer noch Alarm schlagen ? Nein, denn sie sucht, wie oben bereits erwähnt, nur nach /etc/passwd (und anderen Groß/Klein geschriebenen Varianten). Ein weiteres Problem bei diesen Signaturen ist, dass sie meist nur bekannte Angriffe entdecken, nach bekannten Viren suchen. Neuere Versionen bestimmter Attacken werden meistens nicht entdeckt.... Die anderen Signaturen, die sich auf spezielle Angriffe spezialisieren oder allgemeine Signaturen haben zwar den Vorteil, dass sie neue Angriffe eher entdecken, allerdings muss man bei der Formulierung seiner Signatur−Regeln "aufpassen". Eine Signatur, die sich auf einen bestimmten Angriff spezialisiert (auf dessen Entdeckung), wird fehlschlagen bei der Entdeckung einer etwas veränderten Variante (anstatt einen konstanten IP ID Wert von 39426 zu haben, hat die neue Attacke einen variablen Wert). Bei der Formulierung von allgemeinen (protocol analysis based signatures) Signaturen, muss man darauf achten, dass man die Regeln auch wirklich allgemein formuliert, d.h. die Regeln müssen auf Anormalitäten hinweisen, die so nicht auftreten können bzw. dürfen .

Eine weitere Schwäche offenbart sich bei näherer Betrachtung der Unicode Attacke (siehe [4]). Beispielhaft hier die Beschreibung einer Sicherheitslücke im MS IIS,die durch die Unicode Attacke ermöglicht wird/wurde:

"Synopsis:
A flaw exists in Microsoft Internet Information Server (IIS) that may allow remote users to list directory contents, view files, delete files, and execute arbitrary commands. Attackers may use the Unicode character set to craft URLs to access resources via IIS that would normally be inaccessible. All recent versions of IIS are affected by this vulnerability. Exploitation of this vulnerability is trivial. ISS X−Force is aware of widespread exploitation of this vulnerability. "

Das Problem für IDSs besteht darin, dass die Zeichen in UTF−8 verschiedene Codes haben, die doch im gleichen Zeichen resultieren. Z.B. "A": U+0041, U+0100,

U+0102, U+0104, U+01CD, U+01DE, U+8721. Da MS ISS nicht auf Groß/Klein − schreibung achtet(e) , waren wiederum mehrere Möglichkeiten vorhanden, verschiedene Zeichen darzustellen (es existieren z.B. 83,060,640 verschiedene Möglichkeiten AEIOU darzustellen..).

Ruft ein Angreifer nun z.B. "http://victim/../../winnt/system32/cmd.exe" würde IIS noch einen Fehler erzeugen, wenn man allerdings die Stelle "../.." durch ein UTF−8 Äquivalent ersetzt, wird kein Fehler erzeugt : "http://victim/..%C1%9C../winnt/system32 /cmd.exe". Außerdem können sog. unescape UTF−8 Codes dazu führen, dass man Seiten aufrufen kann, die man eigentlich nicht aufrufen darf. Ein NIDS hat es schwer, diese Angriffe zu erkennen, da sich das ganze auf dem Application Layer abspielt, in einem solchen Fall wäre der Einsatz eines HIDSs sinnvoller. Verschlüsselung stellt im allgemeinen ein Problem für Sensoren da, diese Tatsache wird mittlerweile auch immer öfter ausgenutzt. Das Vorgehen einiger "IDS−Hersteller" zeigte auch deutlich die Grenzen der Signaturen, die verfügbaren Signaturen erkannten zwar einige bekannte Unicode−Attacken, bei kleinen Veränderungen des Angriffs wurde die Signatur allerdings wieder wertlos. Lediglich NetworkICE hat zur damaligen Zeit erfolgreich die Signaturen entwickelt, die diese Attacken entdeckten (Snort und ISS RealSecure entwickelten auch Signaturen, allerdings entdeckten diese −wie

gesagt– nur die bekannten Unicode–Attacken).

Response

Wie bereits im Verlauf des Textes erwähnt wurde, untersuchen IDSs die Vorgänge auf dem PC / im Netzwerk, doch was wäre ein IDS, wenn es nichts machen würde, um darauf aufmerksam zu machen / darauf zu reagieren ? Dann wäre ein IDS wohl nicht viel Wert, bzw. nur Verschwendung von Rechenperformance. Grundsätzlich unterscheidet man bei 'Response' einerseits Active Response und andererseits Passive Response. Im jeweiligen Abschnitt sollten die Unterschiede eigentlich deutlich werden.
Dem interessierten Leser sei auf OPSEC (http://www.opsec.com) verwiesen.

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

Dieses System erlaubt es also bestehende Sicherheits–Systeme in die FireWall–1 zu integrieren. Ein zusätzlicher Vorteil liegt darin, dass es weltweit anerkannt wird (und ca. 300 Partner hat). Wenn ihr einen Angriff entdeckt, könntet ihr die IP Adresse des Angreifers durch die Firewall "sperren" (nur als eine mögliche Response Option). Solltet ihr an OPSEC interessiert sein, so solltet ihr euch "Deployment Platforms" durchlesen, dort stehen auch die Voraussetzungen, um dem System "beizutreten" (Partner zu werden).

Active Response

Active Response bedeutet, dass automatisch darauf reagiert wird, wenn das IDS einen Einbruch(sversuch) meldet. Je nachdem, was für ein Angriff gemeldet wird (also auch abhängig davon, wie schwerwiegend der Angriff war) bieten die meisten IDSs verschiedene Optionen, um darauf zu reagieren:
1) Aktionen gegen den potentiellen Einbrecher einleiten 2) "Lediglich" zusätzliche Informationen sammeln (über den Angreifer und dessen Attacke, bzw. deren Auswirkungen) 3) Konfiguration ändern
Die erste mögliche Reaktion wäre hier Aktionen gegen den Einbrecher einzuleiten. Diese Aktionen können natürlich alles mögliche bedeuten, Zugriff für diese Person sperren oder gar Attacken gegen diesen Einbrecher einzuleiten. Wie allerdings schon bei Honeypots erwähnt, ist es oft nicht nur schwierig, seinerseits Attacken gegen den Angreifer zu steuern, sondern oft auch illegal. In diesem Zusammenhang taucht öfter der sog. "Third Party Effect" auf. Was ist dieser Effekt eigentlich? Anschaulich erklärt, ist ein Third Party Effect nichts anderes als z.B. so was :

```
------------                ------------                -------
| Intruder | ------------> | Innocent  | ----------> | YOU |
------------                ------------                -------
```

Der Third Party Effect heisst also einfach, dass eine unschuldige Person (ein unschuldiges Netzwerk) von dem Angreifer erfolgreich attackiert wurde, danach nutzt er dieses Netzwerk, um von dort aus unser Netzwerk (und vielleicht auch andere) anzugreifen. Doch wo liegt hier das Problem? Das Problem besteht nun darin, dass unser Netzwerk 'Innocent' als Angreifer enttarnen würde, anschließend würde es Attacken gegen 'Innocent' einleiten, obwohl 'Intruder' dieses Netzwerk attackiert hat, um gegen uns Attacken einzuleiten, und

nicht 'Innocent', der der eigentliche Täter war. Als Folge unseres Angriffs (mit der falschen Gewissheit, wir würden "nur" den Angreifer attackieren) würde sicherlich nicht zu unterschätzender Schaden bei 'Innocent' entstehen. Wenn 'Intruder' noch geschickt genug war, seine Spuren (also die Spuren seines eigentlichen Angriffs) zu verwischen, werden wir für diesen Schaden verantwortlich gemacht, nicht 'Intruder'.

Die zweite Möglichkeit (also zusätzliche Informationen zu sammeln) ist da schon unproblematischer. Sollte ein möglicher Angriff/Einbruch festgestellt werden, werden nun "lediglich" zusätzliche Informationen über den User und dessen Attacke eingeholt. Wenn ein IDS feststellt, dass ein bestimmter User erfolgreich seine Rechte erweitert hat (oder sonst ein "Angriff" vorliegt) könnte dieser User in Folge dessen genauer observiert werden, z.B. Protokollieren der eingegebenen Kommandos (falls nicht eingestellt), von wo aus hat sich der User eingeloggt, wie lange bleibt er angemeldet, wann hat er sich eingeloggt, wie oft (und wann) loggt er sich die nächsten Tage ein, probiert er irgendwelche bestimmten Binaries zu ftp'en ... , es wird so gesehen also ein Profil des Angreifers erstellt. Dies hat den Vorteil, dass man später die ausführlichen Logs genau analysieren kann, um mögliche Schwachstellen in der Konfiguration zu schließen, außerdem ist es eher möglich, gerichtliche Schritte gegen den Angreifer einzuleiten. Als dritte Möglichkeit sehe ich die mögliche Änderung der bestehenden Konfigurationen des Systems, der Firewall etc. Wenn der Angreifer bestimmte IP Adressen benutzen sollte, so könnte man verbieten, dass sich ein User mit dieser IP mit dem Netzwerk connecten darf. Natürlich könnte man auch sonstige Zugriffe, die vom (vermuteten) Ort aus kommen, blocken (und auch protokollieren). In Ausnahmefällen könnte man vorerst auch einfach mal jeglichen Zugriff auf das eigene Netzwerk unterbinden (oder jeglichen Zugriff auf bestimmte Ports, von bestimmten Netzwerk Interfaces ....) Eine weitere Möglichkeit der Active Response stellt das Beenden einer TCP−Connection dar (TCP−Kill genannt). Um eine Verbindung zu einem anderen Rechner zu beenden, schickt man ihm ein RST (Reset−Flag), so dass die Session "gekillt wird". Normalerweise wird RST nur gesendet, wenn ein Fehler in der Verbindung aufgetreten ist, in diesem Fall kann es aber von einem IDS (wie z.B. ISS RealSecure) dazu "gebraucht" werden Sessions mit einem anderen Rechner zu schließen (es existiert auch ein gleichnamiges Tool , für Win−NT).

```
" 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). "
```

Dies ist ein Ausschnitt aus 'man tcpkill' ....
Wie ihr seht, besteht hier eine recht umfassende Möglichkeit, auf Angriffe zu reagieren. Gegenattacken einzuleiten scheint zwar auf den ersten Blick verführerisch, sollte allerdings möglichst nicht angewandt werden.

Passive Response

Im Unterschied zur Active Response, werden hier meist nur Warnungen... geloggt , die dann der Admin/User...kontrollieren muss. Hier bestehen folgende Möglichkeiten zu reagieren:

1) Warnungen , Hinweise ... loggen
2) Erzeugen von sog. Reports, die den Zustand des Systems über gewisse Zeit beobachten und so einen Bericht anfertigen.

Nahezu jedes IDS besitzt die Möglichkeit Warnungen zu generieren oder Hinweise an User/Browser...zu verschicken. Wenn z.B. versucht wird eine wichtige Systemdatei zu löschen, bestimmte Services zu starten

(deren Benutzung verboten sein sollte),...könnte eine Warnung erzeugt werden, die über den Vorfall informiert, wer daran beteiligt war und auch den Zeitpunkt. Mittlerweile besitzen auch immer mehr IDSs die Möglichkeit sog. Reports zu erzeugen. Der Zustand des Systems wird hier über einen längeren Zeitraum hin überwacht, Vorgänge protokolliert und dann ein Statusreport erzeugt. Die Möglichkeit der Passive Response bieten eigentliche alle IDSs...

Filter

Ein Filter wird dazu benutzt einen Angriff anhand seiner Signatur zu erkennen. Diese Signatur hat indirekt etwas mit den bereits besprochenen Signaturen zu tun , da auch hier typische Merkmale eines Angriffs (wie Dest/Src Ports, Dest/Src IPs...) untersucht werden. Im weiteren Teil dieses Kapitels , werde ich anhand von N−Code einige Beispiele für Filter gegen bekannte Attacken vorstellen und erklären, sollte euch N−Code unbekannt sein, so könnt ihr am Ende dieses Teils eine Seite finden (advanced users guide – nfr) die eine übersicht über N−Code liefert.
land:

```
# Dies ist ein Beispiel, wie man in N-Code
# eine land Attacke erkennen kann
 filter pptp ip () {
        if(ip.src == ip.dest)
        {
                record system.time,
                        eth.src, ip.src, eth.dst, ip.dest
                to land_recdr;
        }
 }
```

Da hier unbekannte Variablen verwendet wurden, hier kurz eine Erklärung :

- ip.src = die Source−IP Adresse
- ip.dest = die Destination−IP Adresse
- eth.src = MAC Adresse der "Ziel−Machine"
- eth.dst = MAC Adresse des "Source−PCs"
- record system.time = protokolliert die Zeit (den Zeitpunkt) zu dem die Bedingung ip.src == ip.dest erfüllt war

Wie ihr seht, kennt auch N−Code den Operator ==, wenn ihr euch den Advanced User's Guide durchlest, werdet ihr feststellen das auch andere Gemeinsamkeiten (mit Hochsprachen...) bestehen, z.B. kennt auch N−Code die Operatoren + , − , *... oder zusammengesetzte Operatoren wie >=, != .... oder wie oben ==. Xmas Scan: Wie ihr aus "Angriffsarten" wissen solltet, sind bei einem Xmas Scan alle Flags gesetzt. Daher scheint es plausibel zu überprüfen, ob denn nun alle gesetzt sind oder nicht. Dafür benötigt man allerdings noch die Werte der einzelnen, gesetzten Bits :

```
 Bit                    Wert
 ------------------------------------
 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; } } }
Auch hier kommen einige unbekannte Variablen vor, die ich erstmal erklären werde:

- tcp.hdr = wenn tcp.hdr == 0, dann beinhaltet das Paket keinen gültigen TCP Header, bei tcp.hdr == 1 schon
- tcp.dport = TCP Destination Port
- tcp.sport = TCP Source Port
- ip.blob = Inhalt des Payloads eines Pakets (ohne Header)
- "UAPRSF" bedeutet das URG,ACK,PSH,RST,SYN und FIN gesetzt sind

Bei $dabyte handelt es sich um eine lokale Variable, der byte(ip.blob,13) zugewiesen wird. Zur Erklärung des "byte−Ausdrucks" hier eine kleine Darstellung der 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 |
```

Nun erkennt man auch warum man hier 13 Bytes in byte() spezifiziert, denn 13 Bytes reichen aus, um die Flags zu erhalten. Bevor man verstehen kann, wie byte arbeitet, erst mal eine Anmerkung zu blobs. Wie ihr im Chapter 3 des Advanced User Guides nachlesen könnt ist ein blob eine "an arbitrarily sized sequence of bytes". 'byte' gibt ein Byte vom spezifierten Offset eines Blobs zurück, die allgemeine Syntax sieht so aus: byte (str blob_to_search, int offset); Durch das erste Argument wird der Blob spezifiert , der durchsucht werden soll (oben ip.blob) und das zweite Argument gibt den Offset (in 'blob_to_search') des gesuchten Bytes an. Durch 'if(!($dabyte ^ 63))' wird nun noch überprüft ob alle Flags gesetzt sind, man erhält 63 wenn man die Werte der Flags (32+16+8+4+2+1) addiert (sollte es jemanden interessieren, durch ^ wird ein bitweises XOR durchgeführt)

Insgesamt bietet N−Code wirklich viele und umfassende Möglichkeiten, neben den bereits genannten. Z.B. ist auch noch möglich:

- zu überprüfen ob es sich bei dem Paket um ein IP Paket handelt (ip.is)
- die Länge des IP Paketes ermitteln (ip.len)
- das verwendete Protokoll desIP Pakets (ICMP,TCP oder UDP) (ip.protocol)
- die Checksumme eines ICMP Paketes zu ermitteln (icmp.cksum)
- den Inhalt des Payloads der ICMP Pakete (als blob) zu ermitteln (mit icmp.blob)
- zu testen ob das Paket einen gültigen ICMP header enthält (icmp.hdr)
- zu testen ob das Paket überhaupt ein ICMP Paket ist (icmp.is)
- den Typ des ICMP Paketes zu ermitteln, also Echo Reply, Destination unreachable....
- ..... etc .....

Weitere Informationen zu N−Code könnt ihr im Advanced User's Guide einholen:
http://www.cs.columbia.edu/ids/HAUNT/doc/nfr−4.0/advanced/ advanced−htmlTOC.html

In zukünftigen Versionen dieses Papers, werden wohl wesentlich mehr Filter beschrieben werden, also checkt mal regelmäßig nach neuen Releases dieses Papers ;)

Standards

In diesem Abschnitt werde ich euch verschiedene "Standards" vorstellen, also Listen/Vereinbarungen....die von vielen ools/"Experten".... gemeinsam verwendet werden.

CVE

CVE steht für Common Vulnerabilities and Exposures und ist eigentlich nichts anderes als eine Liste , die Namen für vulnerabilites/exposures vorsieht. Dies mag sich zwar auf den ersten Blick komisch anhören, kann allerdings später wichtig werden. Verschiedene Tools benutzen meist verschiedene Ausdrücke für die gefundenen Vulnerabilites...., durch die Verwendung von CVE kann man so eine einheitliche Beschreibung verschiedenster Vulnerabilites/Exposures verwenden, die jeder versteht. Man ist also nicht mehr davon abhängig, dass der andere auch dieses Tool verwendet....

CVE stellt also einen Namen für eine bestimmte Vuln/Exposure bereit, zusätzlich aber auch noch eine Beschreibung, durch die einheitliche (und standardisierte) Beschreibung, tauchen weniger Missverständnisse zwischen Usern verschiedener Systeme auf. CVE definiert hierbei vulnerability als : "those problems that are normally regarded as vulnerabilities within the contect of all reasonable security policies" und exposures als "problems that are only violations of some reasonable security policies". Die Unterscheidung zwischen vulnerability und exposures ist elementar in CVE. Als Beispiel für eine vulnerability , lassen sich z.B. phf, world−writeable password files.... nennen, während ein Beispiel für exposures z.B. die Verwendung von Programmen (die man bekannterweise per Bruteforce angreifen kann) wäre, oder auch die Verwendung von Services , die im allgemeinen auch oft angegriffen werden. Durch die Defintion und diese Beispiele sollte es eigentlich möglich sein, zwischen vulnerabilites und exposures (in CVE) zu unterscheiden. Der grundsätzliche Unterschied besteht v.a. darin , dass vulnerabilites beinhalten das ein Angreifer Möglichkeiten besitzt Kommandos als anderer User auszuführen oder gar Dateien zu lesen/beschreiben, obwohl dies (aufgrund der file−permissions) nicht möglich sein sollte. Exposures hingegen beinhalten, dass ein User weitere Informationen über das System (und dessen Zustand) herausfinden kann, seine Aktivitäten dabei versteckt ablaufen.....Exposures entstehen also aufgrund von falschen Sicherheitseinstellungen, die man allerdings "beheben" kann. Als Vulnerability wird hingegen eher eine Sicherheitslücke verstanden, die unter den "gewöhnlichen" Sicherheitssystemen auftritt. Außerdem soll hier normalerweise auch die Möglichkeit gegeben sein, die Bedrohung durch potentielle Angreifer minimieren zu können (in dem man Permissions checkt....). Doch irgendwie muss die "Liste" auch aktuell bleiben, allerdings wird nicht jede vulnerability oder exposure sofort "aufgenommen". Wenn eine vulnerability / exposure gefunden wird, wird ihr zuerst eine "candidate number" zugewiesen (dies geschieht durch die CNA − Candidate Numbering Authority). Außerdem wird sie auf dem Board vorgeschlagen (vom CVE Editor) und danach drüber geredet ob man die vulnerability /exposure aufnehmen will. Sollte das Board zum Schluss kommen, dass man den Kandidaten (vorerst) nicht aufnehmen will, so wird auf ihrer Website noch der Grund angegeben. Sollte der Kandidat allerdings akkzeptiert werden, wird er in die Liste aufgenommen (und ist somit nun offiziell Teil von CVE). Nun sollte es auch verständlicher sein, das jeder (potentiellen) vulnerability.... erstmal ne "candidate number" zugewiesen wird, da erst danach darüber diskuttiert werden muss, ob der Kandidat aufgenommen werden soll oder nicht. Der vulnerability... wird hier erstmal eine 'candidate number' zugewiesen, damit man zwischen Kandidaten und offiziellen Einträgen in der Liste unterscheiden kann. Jeder Kandidat besitzt 3 grundlegende Felder (die ihn "identifizieren"):

- Number
- Description
- References

Die Nummer ist sogesehen der eigentliche Name des Kandidaten, wobei sich diese Nummer aus "Erscheinungsjahr" und einer weiteren Zahl zusammensetzt , die angibt der wievielte Kandidat es dieses Jahr war:

```
CAN-JAHR-wievielter Kandidat des Jahres
```

Wenn nun ein Kandidat vom Board akkzeptiert wird, wird er wie bereits gesagt in die Liste aufgenommen. Damit ist auch verbunden , dass aus 'CAN−YEAR−Candite number' ein 'CVE−YEAR−Candidate number' wird. D.h. an einem Beispiel: Existiert ein Kandidat mit 'CAN−2001−0078' und wird dieser Kandidat akkzeptiert, wird er in die Liste aufgenommen als 'CVE−2001−0078'.

Soviel dazu, die offizielle Seite zu CVE ist http://cve.mitre.org/ , dort könnt ihr natürlich zusätzliche Informationen finden...

Beispiele

In diesem letzten Teil werden einige IDSs vorgestellt :

Snort

Da Snort sehr weit verbreitet ist und viele Optionen bietet, werde ich es etwas ausführlicher als die anderen Beispiele beschreiben. Grundsätzlich kann sich Snort in drei Modes befinden : Sniffer, Packet Logger und Network Intrusion Detection System. Im Sniffer Modus gibt Snort Pakete auf der Konsole aus, im Packet Logger Modus loggt es sie auf der Festplatte... und im Network Intrusion Detection Modus können Pakete analysiert werden. Zwar gehe ich hauptsächlich auf den letzten Modus ein, aber dennoch hier kurz eine kleine Einführung in Sniffer u. Packet Logger Mode: Sniffer Mode:
Im Sniffer Modus könnt ihr euch verschiedene Paketinformationen ausgeben lassen, z.B. TCP/IP Paket Header:

```
[Socma]$ ./snort −v
```

Als Ausgabe werdet ihr lediglich die IP/TCP/ICMP/UDP Headers ausgegeben bekommen. Es gibt eine Vielzahl von Optionen, von denen hier einige aufgezählt werden.

```
-d = gibt auch noch die Paket−Daten aus
-e = zeigt auch noch den Data Link Layer
```

Packet Logger Mode:
Im Unterschied zum Sniffer Mode, könnt ihr im Packet Logger Mode die Pakete auch auf der Festplatte protokollieren. Ihr müsst lediglich ein Verzeichnis angeben, in das Snort loggen soll und schon schaltet es automatisch in Paket Logger Modus:

```
#loggingdirectory muss existieren:
[Socma]$ ./snort −dev −l ./loggingdirectory
```

Bei Angabe von "–l" kann es manchmal passieren das Snort die Adresse des Remote–Computers als Verzeichnis holt (in das geloggt wird) und manchmal nimmt es die lokale Host Adresse. Um relativ zum Heimnetzwerk loggen zu können, müsst ihr auf der Kommandozeile spezifizieren, welches das Heimnetzwerk ist:

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

Eine weitere Möglichkeit besteht darin, das ganze im TCP–DUMP Format zu loggen:

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

Nun wird auch das gesamte Paket geloggt, nicht nur bestimmte Sektionen, dadurch fällt die Angabe von weiteren Optionen weg. Zwar kann man Programme wie tcpdump benutzen um die File wieder in ASCII –Text zu "übersetzen" , allerdings kann Snort dies auch erledigen:

```
 [Socma]$ ./snort –dv –r packetzumuntersuchen.log
```

Network Intrusion Detection Mode: Um in den NIDS Modus zu schalten , könnt ihr ein Kommando wie das folgende verwenden:

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

In diesem Fall ist snort.conf die Konfigurationsdatei. Diese wird benötigt damit Snort weiss woher es seine "Regeln" herbekommt, also wann ein Angriff vorliegt oder nicht, ob eine Anfrage gestattet werden soll.... Die in snort.conf festgelegten Regeln werden dann auf die Pakete angewandt und analysiert. Wird kein Output–Verzeichnis festgelegt, wird standardmäßig /var/log/snort dazu verwendet. Die Ausgabe von Snort hängt auch von den Alert–Modes ab, je nachdem welchen Alert–Modus man verwendet bekommt man mehr oder schneller seine Informationen:

Modus	wie/was wird ausgegeben
–A fast	Zeitpunkt, Source u. Destination IPs/ports, die eigentliche Alert Nachricht
–A full	Standardeinstellung
–A unsock	sendet die Warnungen an einen UNIX socket
–A none	stellt die Alerts ab

Wie wir wissen kann durch –b im Binary Mode geloggt werden, durch –N wird Paket Logging ganz abgeschaltet. Doch das ist noch längst nicht alles, z.B. kann Snort auch Nachrichten an Syslog schicken. Standardeinstellung ist hier LOG_AUTHPRIV und LOG_ALERT. Zum Senden von Nachrichten an Syslog müsst ihr lediglich "–s" angeben, Beispiele dazu gleich. Des weiteren besteht die Möglichkeit auch Nachrichten an den smbclient oder WinPopUp Warnungen an einen Windows Rechner zu schicken. Um dieses "Feature" nutzen zu können müsst ihr bei der Konfiguration von Snort "–enable–smbalerts" angeben.

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

Oder hier ein Beispiel , das die Anwendung der Alert Modes zeigt:

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

Neben den bereits gesagten Optionen gibt es noch einige, wie die folgenden:

```
-D = starte Snort im Daemon Mode
-u usersnort= starte Snort mit der UID 'usersnort'
-g groupsnort = starte Snort mit der GID 'groupsnort'
-d = protokolliere auch die Daten aus der Applikationsschicht mit
```

Snort bietet viele Optionen, bei Problemen könnt ihr einfach mal "snort –h" eintippen oder in Mailinglisten danach gucken ob euer Problem nicht schon mal aufgetaucht ist. Der folgende Abschnitt handelt von den Snort–Rules, solltet ihr kein Interesse daran haben die bestehenden Rules zu verstehen oder gar eigene zu schreiben, so könnt ihr diesen Abschnitt überspringen. Wie ich am Ende (des Teils über Snort) nochmals erwähnen werde, könnt ihr das SnortUsersManual unter www.snort.org downloaden, dieses dient hier als eigentliche Quelle.

Snort Rules:
Zum besseren Verständnis von Snort ist auch ein Verständnis der Snort Rules notwendig. Snort benutzt manchmal bestimmte Variablen, die ihr durch Verwendung von 'var' definieren könnt:

```
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";)
```

Es gibt allerdings mehrere Möglichkeiten den Variablennamen anzugeben:

```
$variable = definiert die Meta Variable
$(variable) = hier wird der Wert der Variable "variable" eingesetzt
$(variable:-default) = wenn 'variable' definiert ist, wird hier der
 wert von 'variable' eingesetzt, ist 'variable' nicht definiert wird
 hier der Wert von 'default' eingesetzt
$(variable:?msg) = setzte den Wert der Variable 'variable' ein
 oder (wenn nicht definiert) gib die Nachricht 'msg' aus
```

Solltet ihr euch schon mal etwas mit Shellprogrammierung befasst haben, kommen euch diese Dinge sicherlich nicht fremd vor:

```
[Socma]$ shelltest=we
[Socma]$ echo hallo $shelltestlt
hallo
[Socma]$ echo hallo ${shelltest}lt
hallo welt
```

Hier ist also die Verwendung von $(variable) in Snort und ${variable} in der Shell gleichbedeutend. Auch für die anderen gibt es Äquivalente (oder ähnliche Ausdrücke) in der Shellprogrammierung:

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

Also auch die Verwendung des Ausdrucks $(variable:–default)' unterscheidet sich nur in der Tatsache das auf der Shell { und } anstatt (und) benutzt werden. Der letzte Ausdruck existiert auch auf der Shell:

```
[Socma]$ shelltest = bash
[Socma]$ echo ${shelltest:?"dann eben csh"}
bash
[Socma]$ echo ${nichtdefiniertevariable:?"nicht definiert oder null"}
nicht definiert oder null
```

Dieser kleine Exkurs sollte lediglich "Wissen verknüpfen", zumindest ich konnte mir die Schreibweisen in Snort so schneller einprägen, da ich einfach an die (mir bekannten) Dinge auf der Shell gedacht habe.
Viele Kommandozeilenoptionen...können in der Konfigurationsdatei eingestellt werden. Hierzu wird 'config' verwendet :

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

Die wichtigsten 'directiven' wären :
alertfile = ändert die Datei in die Alerts gespeichert werden
daemon = Prozess als Daemon starten (–D)
reference_net = setzt das Heimnetzwerk (–h)
logdir = setzt das Logging Verzeichnis (–l)
nolog = Logging wird ausgeschaltet
set_gid = ändert die GID (–g)
chroot = chroot'ed in das angegebene Verzeichnis (–t)
set_uid = setzt die UID (–U)

Wenn ihr z.B. die alertfile ändert wollt in z.B. "mylogs", macht ihr das so:

```
config alertfile : mylogs
```

Nun aber zurück zu den eigentlichen Regeln (hier eine beispielhafte ftp.rules (Ausschnitt)):

```
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;)
```

Grundsätzlich bestehen Snort–Rules aus zwei Teilen : dem rule header und den rule options. Hierbei gibt der rule header Auskunft über folgende Dinge:

- Source u. Destination IP Adressen
- Protokoll
- die Aktionen die von der Regel eingeleitet werden

In der obigen ftp–rule wäre der rule header der folgende Teil:

```
Aktion       source ip         destination ip
   |             |                  |
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
        |               |               |
       Protokoll       From any port   Port
```

Hier ran sieht man , dass der rule header bis zur ersten (geht und danach die rule options beginnen. Es gibt mehrere mögliche Aktionen (in diesem Falle alert) die eingeleitet werden können, sollte die Snort−Rule etwas auffälliges feststellen:

- alert = je nachdem welche alert−methode (s.o. − standard ist alert full) verwendet , wird ein Alert erzeugt und das entsprechende Paket geloggt
- log = hierbei wird einfach nur das Paket geloggt
- pass = führt dazu das das Paket ignoriert wird
- activate = erzeugt einen Alert und kehrt dann zu einer anderen dynamischen Snort−Rule zurück (mehr dazu gleich)
- dynamic = führt dazu, dass die Regel solange "inaktiv" bleibt , bis sie von einer anderen Regel aktiviert wurde, danach arbeitet es wie log (s.o.)

Das zweite Feld (Protokoll − hier tcp) spezifiziet welches Protokoll "beobachtet" (analysiert) werden soll. Möglich sind hier : tcp ,udp , icmp und ip (allerdings ist es nicht unwahrscheinlich das in Zukunft auch andere wie ARP,GRE....verfügbar sind).
Im Zusammenhang mit dem nächsten Feld (source ip) trifft man öfters auch auf den ! − Operator (Negationsoperator).

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

Als Folge der Verwendung des Negationsoperators wird nun mehr jedes Paket das nicht von $EXTERNAL_NET kommt geloggt. Des weiteren besteht die Möglichkeiten mehrere IP Adressen anzugeben, also Listen von IP Adressen. Dafür müsst ihr lediglich die entsprechenden Adressen mit Komma voneinander trennen und mit [ ] umklammern.

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

Eine weitere Alternative stellt die Verwendung von "any" dar, womit jede IP Adresse einbezogen ist.

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

Der letzte Teil des rule headers ist die Spezifikation des Ports, im obigen Beispiel also ftp. Man kann nicht nur einen bestimmten Port ueberwachen, sondern auch bestimmte Bereiche (also auch mehrere Ports). Hier die verschiedenen Möglichkeiten:

```
:portnumber                    -> alle ports kleiner gleich
                               portnumber
portnumber:                    -> alle ports größer gleich
                               portnumber
fromportnumber:toportnumber    -> alle ports zwischen fromportnumber
                               und toportnumber (und diese mit
                               einbezogen)
```

Natürlich lässt sich auch hier wieder der Negationsoperator angeben, was zur Folge hätte das alle außer die angegebenen Ports "überwacht" werden würden, z.B.

```
!:21            -> alle ports die nicht kleiner gleich 21 sind
```

Etwas was noch nicht direkt erläutert, die ganze Zeit aber verwendet wird, ist der Direktionsoperator "–>".

```
"Source" -> "Destination"
```

Doch es gibt auch eine andere Variante, <> :

```
"Source" <> "Destination"
```

Dies führt dazu das Snort sowohl Source als auch Destination nach der Adresse.... durchsucht.
Wie ich bereits eben gesagt habe, gibt es die Aktion 'activate' , die dazu führt das ein Alert erzeugt wird und dann zu einer anderen dynamischen Snort–Regel zurückkehrt.
Sollte eine bestimmte Regel die jeweiligen Aktionen durchgeführt haben kann es eine andere aktivieren.
Grundsätzlich unterscheiden sich normale Regeln und "activate rules" nur in der Tatsache das ein bestimmtes Feld bei activate rules angegeben werden muss : "activates". Dynamische Regeln hingegen arbeiten wie log (s.o.) nur das hier "activated_by" angegeben wird. Des weiteren existiert ein weiteres Feld, dass angegeben werden muss :"count". Wenn die "activate rule" ihre Arbeit getan hat, wird die dynamische regel "aufgerufen", allerdings nur für "count" Pakete (also wenn count = 40, dann für 40 Pakete).

```
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;)
```

Einige Optionen (also die rule options) sind noch nicht bekannt, daher werde ich jetzt auf diese eingehen, später erscheint euch diese Regel dann auch sinnvoller. Beachtet im obigen Beispiel bitte die Felder activates und bei der dynamischen Regel activated_by (und count). Die erste Regel ruft nach erledigter Arbeit die dynamische Regel auf , dies sieht man auch daran das in der dynamischen Regel activated_by : 1 steht.

Nun aber zum zweiten Teil der Snort–Rules, den rule options. Nimmt man nochmals die erste ftp.rule:

```
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 diesem Falle wäre die rule option (der rule header geht wie gesagt nur bis zur ersten ")") :

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

Zwar gibt es 34 Keywords, allerdings werde ich nur die (wichtigsten und/oder gebräuchlichsteen) erläutern. Derjenige der eine Übersicht über alle möglichen Keywords haben möchte, sollte ins SnortUsersManual schauen, dort stehen alle erklärt.

msg – gibt die alert–messages aus und loggt die Meldungen im Paket Logger Modus
logto – loggt die Pakete in einer bestimmten Datei
dsize – vergleicht die Paketgröße mit einem anderen Wert

flags – überprüft die TCP Flags nach bestimmten Werten
content – sucht nach nem bestimmten Muster/String in einem Paket
content–list – sucht nach mehreren Mustern/Strings in einem Paket
nocase – Groß–Kleinschreibung des gesuchten Strings werden nicht beachtet
react – aktive Response (blockt Webseiten)
sid – Snort rule id
classtype – teilt die potentiellen Angriff in bestimmte Gruppen von Angriffen ein
priority – regelt die "Strenge"
So weit so gut, doch was genau machen die einzelnen rule options ? msg:
Auf 'msg' trifft man sehr häufig beim Durchstöbern der rules, da diese Option dafür verantwortlich ist, dass Alerts ausgegeben werden (oder auch das geloggt wird).

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

Hierbei ist "<text>" die jeweilige Nachricht die in die alertfile geschrieben wird/ausgegeben wird.....
logto:
Jedes Paket, auf das die Rule zutrifft, wird in einer speziellen File geloggt.

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

In diesem Falle ist "<filename>" die Datei, in die diese Paket geloggt werden.
dsize:
Hiermit wird die Größe des Pakets bestimmt. Wenn man weiss das ein bestimmter Service einen Buffer (einer bestimmten Größe) kann man diese Option verwenden, um möglichen Bufferoverflows entgegen zu wirken. Verglichen mit 'content' ... ist es um einiges schneller, daher wirds zum Testen von BufferOverflows auch eher benutzt.

```
dsize: [>|<] <größe>;
```

Die beiden optionalen Operatoren > und < drücken aus, dass die Größe des Pakets kleiner,bzw. größer als dieser Wert sein sollte.. flags:
Hiermit wird getestet welche Flags gesetzt sind. Momentan sind 9 Flags in Snort verfügbar:

```
F      FIN
S      SYN
R      RST
P      PSH
A      ACK
U      URG
2      Bit 2 reserviert
1      Bit 1 reserviert
0      keine TCP Flags gesetzt
```

Daneben gibt es auch noch logische Operatoren, die zusätzliche Kriterien zum Testen der Flags spezifieren können:

```
+ ALL flag     = Treffer bei allen spezifierten (und auch anderen)
               Flags
* ANY flag     = Treffer bei allen spezifierten Flags
! NOT flag     = wenn die spezifierten Flags nicht  gesetzt sind
```

Allgemein wird das 'flags' Keyword so verwendet:

```
flags: <Flag Wert>;
```

Die reservierten Bits können verwendet werden um ungewöhnliches Verhalten, wie z.B. IP stack fingerprinting Versuche zu erkennen.
content:
Einer der wohl (neben msg) am häufigsten gebrauchten Keywords ist 'content'. Hiermit kann der Payload Pakete nach bestimmten Inhalt durchsucht werden. Sollte der spezifiezierte Inhalt gefunden werden, werden bestimmte Aktionen gegen den User eingeleitet. Wenn der Inhalt (der nach content spezifiert wird) in dem Payload des Pakets vorkommt, wird der Rest der jeweiligen Snort−Rule ausgeführt. Ohne Angabe von nocase (s.u.) wird auch Groß/Kleinschreibung beachtet. Der Inhalt , nach dem der Payload durchsucht werden soll, kann sowohl binäre daten, als auch text enthalten. Binäre Daten werden durch | | eingeschlossen und als Bytecode dargestellt. Der Bytecode stellt die binären Informationen als hexadezimale Zahlen dar... Auch im Zusammenhang mit diesem Keyword kann man den Negationsoperator (!) geschickt anwenden, so dass man z.B. einen alert... ausgibt wenn ein Paket nicht einen bestimmten Text...enthält.

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

Die Angabe von ! ist wie gesagt optional, also nicht zwingend.

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

Wie in dieser Regel zu sehen ist, werden die binären Daten durch die | | umschlossen (wie oben bereits erwähnt), danach lässt sich normaler Text spezifieren. Ihr solltet euch auch einmal die Beschreibung von 'offset' und 'depth' im SnortUsersManual anschauen, da diese auch öfters in Verbindung mit 'content' gebraucht werden.

content−list:

Dieses Keyword arbeitet so ähnlich wie 'content' mit dem Unterschied das man mehrere Strings.... angeben kann, nach denen das Paket durchsucht werden soll. Man schreibt die entsprechenden Hexazahlen,Strings.... in eine Datei und spezifiziert dann bei Verwendung von 'content−list' die Datei in der die Wörter stehen. Zu beachten ist hier, das die Strings ... untereinander (jeder String in einer Zeile) stehen muss, z.B.

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

Danach kann man durch z.B. 'content−list: [!] "<dateiname>";' diese Datei "durchsuchen". Auch hier kann man natürlich optional ! angeben, hat aber auch die gleiche Wirkung wie bei 'content'.

nocase:
Diese rule option spielt eine wichtige Rolle im Zusammenhang mit 'content' keywords. Diese achten normalerweise auf Groß/Kleinschreibung, durch Verwendung von 'nocase' wird aber nicht mehr explizit auf Groß/Kleinschreibung kontrolliert:

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

Ohne Verwendung von 'nocase' würde nur nach 'USER root' durchsucht, da aber 'nocase' angegeben wurde zählt die Groß/Kleinschreibung nicht mehr... react:
Wenn man ein Paket nach einem bestimmten Inhalt durchsucht (durch 'content' oder 'content_list') und einen Treffer feststellt, kann 'react' dazu verwendet werden darauf zu reagieren. Normalerweise werden z.B. bestimmte Seiten die ein User besuchen will geblockt (Pornoseiten...). Durch Flex Resp ist es möglich Verbindungen zu beenden oder Warnungen an den Browser zu schicken. Folgende Optionen sind möglich/gültig:

block – schließt die Verbindung und sendet einen Hinweis raus
warn – sendet eine sichtbare Warnung (bald verfügbar)
Diese sog. 'basic arguments' können dann noch durch weitere Argumente (sog. 'additional modifiers') ergänzt werden:

msg – der text der durch das Keyword 'msg' gesendet wird, wird in den Hinweis an den User einbezogen
proxy : <portnummer> – hier wird der proxy port dazu verwendet den Hinweis zu versenden (auch bald verfügbar)

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

Das 'react' Keyword wird am Ende der rule options eingefügt, und kann z.B. so verwendet werden:

```
alert tcp any any <> 192.168.1.0/24 80 (content-list: "adults"; \
  msg:"Diese Seite ist nicht für Kinder gedacht !"; react: block, msg;)
```

sid:
'sid' oder ausformuliert Snort rules IDentification wird benutzt um "einzigartige" Snort rules zu identifizieren. Dadurch ist es "output plugins" erlaubt/möglich jede Regel leicht zu identifizieren. Allerdings gibt es verschiedene Bereiche von Sid's:

```
< 100 = für die Zukunft reserviert
100-1 000 000 = Regeln die zusammen mit Snort "kommen"
> 1 000 000 = wird für lokale Regeln verwendet

sid-msg.map enthält ein mapping der msg tags zu sid's. Diese wird von
post-processing benutzt um einer id eine Warnung zuzuordnen.

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:
Durch 'classtype' könnt ihr die Attacken in diverse Gruppen von Attacken unterteilen. In den Regeln könnt ihr dann selbst bestimmen welche Priorität ein potientieller Angriff haben soll, bzw. zu welcher Gruppe der Attacken er gehört .Regeln die in der Konfigurationsfile eine Einstufung "haben", bekommen dort automatisch auch eine Standard–Priorität zugewiesen.

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

Die Einstufungen der Regeln werden in der classification.config definiert. Dort gilt folgende Syntax:

```
config classification: <class name>,<class beschreibung>,\
      <standard priorität>
```

Welche Gruppen von Attacken es gibt, könnt ihr im nächsten Teil sehen, der Beschreibung von 'priority'.
priority:
Durch dieses Keyword könnt ihr euren Regeln eine "Security−Stufe" zuweisen, wie schwer also ein potentieller Angriff sein würde. Je höher die Priorität der Regel, desto größer das potentielle Sicherheitsrisiko. Verbunden mit den bereits besprochenen 'class types' lassen sich die Prioritäten besser verstehen:

Class type	Beschreibung	Priorität
not-suspicious	Jeglicher "unauffälliger" Verkehr	0
unknown	Unbekannter Verkehr	1
bad-unknown	Potentiell "schlechter" Verkehr	2
attempted-recon	Attempted Information Leak	3
successful-recon-limited	Information Leak	4
successful-recon-largescale	Large Scale Information Leak	5
attempted-dos	Versuchte DoS-Attacke	6
successful-dos	Erfolgreiche DoS-Attacke	7
attempted-user	Versuch User-Privilegien zu erhalten	8
unsuccessful-user	Nicht erfolgreicher Versuch User-Privilegien zu erhalten	7
successful-user	Erfolgreicher Versuch User-Privilegien zu erhalten	9
attempted-admin	Versuch Admin Rechte zu bekommen	10
successful-admin	Erfolgreicher Versuch Admin Rechte zu bekommen	11

Wie bereits erwähnt bedeuten höhere Prioritäten auch höhere Sicherheitsrisiken, sollte es einem User gelingen Admin Rechte zu bekommen ist dies die schwerwiegenste Attacke.

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

Das ganze Thema 'Rules' ist zugegebenermaßen sehr umfassend, doch nicht soo schwer. Schaut euch einfach diverse rules an, guckt im Manual oder hier nach was die einzelnen bedeuten und nach ner gewissen Zeit erscheinen euch auch die "Rulemonster" verständlich ;) Bezugsquelle für Snort und Dokumentationen findet ihr unter http://www.snort.org . Dort findet ihr einige interessante .pdfs , z.B. das Snort Users Manual, die Hauptquelle für diese Beschreibung von Snort.

LIDS

Spätestens seit stealth's Paper (und Sources) und dem LIDS−Hacking−HOWTO wissen wir , dass LIDS nicht wirklich die Sicherheit auf dem Rechner erhöht, sondern in manchen Situationen eher ein Rootkit ist ;)

Aber erst einmal zum Konzept und anschließend zu den (scheinbaren) Stärken und Schwächen von LIDS. Ursprünglich wurde es entwickelt um z.B. wichtige Systemdateien zu schützen und bestimmte Prozesse für die User unsichtbar zu machen. Außerdem sollte es nicht gestattet sein einfach so Module einzubinden, die

notwendigen Module werden beim Starten des Systems eingebunden... Da es (wie schon gesagt) ein LIDS−Hacking−HOWTO u. Stealth's Paper gibt , die die Arbeitsweise und Schwächen von LIDS erläutern, werde ich hier nur kurz die wichtigsten Dinge erwähnen. Den Link zu den beiden Texten findet ihr am Ende dieses Abschnitts.

Eine Hauptaufgabe von LIDS ist es, das Dateisystem zu schützen. Um (wichtige) Dateien schützen zu können, werden die entsprechenden Dateien / Verzeichnisse erst einmal in Gruppen unterteilt:

- Read Only = Die Datei / das Verzeichnis ist nur lesbar, es sind keine Änderungen erlaubt
- Append Only = In solchen Dateien....ist es nur erlaubt Inhalt "anzuhängen"
- Exception = diese Dateien .... sind nicht geschützt
- Protect (un)mounting = erlaubt / verbietet jemandem ein Dateisystem zu (un)mounten

Um "wirklichen" Schutz bieten zu können, werden einige Systemaufrufe manipuliert, die sicherstellen das die Protections eingehalten werden. (z.B. sys_open(), sys_mknod(), ...)
Desweiteren verhindert LIDS , dass bestimmte Prozesse gekillt werden können (oder sichtbar sind). Sinn u. Zweck des ganzen ist es zu verhindern, dass der Angreifer gewisse Prozesse sieht, die ihn möglicherweise beobachten könnten. Ein 'ps −ax..' Aufruf sollte also unsere Prozesse nicht anzeigen. Damit man den Prozess auch wirklich verstecken kann, wird er als 'PF_HIDDEN' markiert, wenn ps dabei ist die Informationen über die Prozesse auszugeben und sieht das ein Prozess mit 'PF_HIDDEN' markiert ist, wird es diesen Prozess nicht anzeigen. Doch das alleine genügt noch nicht um den Prozess wirklich zu verstecken, denn momentan hätte er immer noch einen Eintrag im Proc−Filesystem (/proc), also manipuliert LIDS auch diese "Funktion", damit der Prozess nicht im /proc Verzeichnis erscheint. Daneben besteht noch die Möglichkeit die Rechte eines Prozesses zu beschränken, anhand von Capatibilites. Wenn z.B. CAP_CHROOT auf 0 gesetzt ist, hat der Prozess nicht mehr die Möglichkeit chroot anzuwenden (siehe auch /usr/src/linux/include/linux/capatibilites.h).
Außerdem besitzt LIDS die Möglichkeit in 2 Sicherheitsstufen zu laufen, 'security' und 'none_security'. Um zwischen 'security' und 'none_security' zu unterscheiden, gibt es die globale Variable 'lids_load'. Standardmäßig ist diese auf '1' voreingestellt, dies bedeutet das standardmäßig LIDS im 'security' Modus läuft, d.h. die Einschränkungen...gelten. Wenn man beim Start 'security=0' angibt (LILO−Prompt) so wird auf 'none_security' geschaltet, was bedeutet das jegliche Sicherheitsüberprüfungen, Einschränkungen....abgeschaltet sind. Mit lids_load == 0, läuft der Rechner also so, als wäre LIDS gar nicht installiert. Des weiteren besteht die Möglichkeit die Sicherheitsstufe durch 'lidsadm −S' online zu ändern , dann muss allerdings ein Passwort spezifiert werden.

Doch LIDS bietet auch die Möglichkeit die Firewall Regeln zu schützen, man sollte dafür aber CONFIG_LIDS_ALLOW_CHANGE_ROUTES anschalten und CAP_NET_ADMIN "ausschalten". Möchte jemand die Firewall Regeln ändern, muss er zuerst CAP_NET_ADMIN wieder anschalten, dadurch wird verhindert das jeder die Regeln ändern kann. Zusätzlich besteht noch die Möglichkeit Sniffer "abzuschalten" und einen Port Scan detector im Kernel einzubauen.

Auch LIDS bietet diverse "Response−Options" (siehe Kapitel über Response), z.B. die Benachrichtigung über den Pager, per SMS an den Administrator zu verschicken.

Alles in allem sehr viele Möglichkeiten, doch Stealth hat in seinem Paper auch gezeigt wie man LIDS missbrauchen kann um es zu einem verfügbar: http://www.securitybugware.org/Linux/4997.html Das LIDS Howto könnt ihr hier runterladen : http://www.lids.org/lids−howto/lids−hacking−howto.html

COLOID

COLOID ist die Abkürzung für Collection of LKMs for Intrusion Detection und wurde vor einiger Zeit von mir ins Leben gerufen.
Da sich herausgestellt hat, das Teile des Projekts uneffektiv und nicht "zeitgemäß" arbeiteten, wurde das Projekt vorübergehend beendet. Nichtsdestotrotz möchte ich kurz auf ursprünglich geplante Features eingehen : Das erste Modul (prev_exec) sollte u.a. die Ausführung bestimmter Binaries (in meinem Source habe ich den Gnu C Compiler – GCC – genommen) zu bestimmten Zeiten verhindern. Es wird also im Source definiert wann die Ausführung verboten sein soll, wobei möglich ist auf Minuten genau den Zeitabschnitt anzugeben. Führt jetzt ein User GCC zu dieser Zeit aus, wird lediglich die Ausführung 'geblockt', d.h. GCC wird nicht ausgeführt. Sollte der User GCC zur "erlaubten" Zeit ausführen werden die Argumente überprüft und nach .c – Files durchsucht. Sinn u. Zweck des Ganzen ist es, den Source (den jemand kompilieren möchte) nach "gefährlichen Funktionen" zu durchsuchen. Voreingestellt waren hierbei 'scanf' und 'strcpy' .... etc ..... , allerdings ist es auch möglich weitere Funktionen...hinzuzufügen. Wie bereits gesagt durchsuchte nun das LKM den jeweiligen Source und sollte es auf eine der Funktionen treffen wurde die Ausführung von GCC verhindert. Zusätzlich wird in eine Logfile geschrieben und ein 'beep' erzeugt .

Das Modul funktionierte zwar soweit, allerdings war der Ansatz nicht umfassend genug und konnte leicht umgangen werden.

Das zweite Modul war 'anom_detection' , das die bereits erläuterte Anomaly Detection verwendete. Eigentlich waren es zwei LKMs die zu diesem Bereich gehören :
1) Anomaly_Detection.c das die Datenbank normaler Useraktivitäten erzeugt
und
2) Misuse_Detect.c das später (mit der Datenbank als "Fundament") kontrolliert ob das Verhalten der User von dem normalen (in der Datenbank protokollierten) Verhalten abweicht.

Geplant war, dass dieses LKM folgende Dinge "überprüft":

Wie oft hat der User die folgenden Programme/Kommandos ausgeführt :

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

Wann hat er die folgenden Programme/Kommandos gestartet:

- login
- su

Bzw. wann hat er normalerweise den PC gestartet u. shutdown ausgeführt....
Sonstiges:
Wie oft hat er (versucht) die folgenden Dateien zu öffnen:

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

Wie oft hat er Programme ausgeführt, bei denen das SUID−Bit gesetzt war ? ....
Aufpassen muss man hier nur bei der Spezifizierung welche Dateien man beobachten möchte (wie oft er sie geöffnet hat). Wählt man hier zu viele Dateien, so leidet die Rechnerperformance zu sehr, als dass ansatzweise vernünftiges arbeiten am PC kaum möglich wäre.

Es gab noch andere kleinere LKM's bei den noch kein kompletter Source verfügbar war, der Source der beiden oben erwähnten Module ist auf meiner Seite verfügbar, vielleicht kann der ein oder andere ja noch was mit anfangen....

Abschließende Worte

Solltet ihr noch irgendwelche Ideen zum Inhalt des Papers ... haben, schreibt mir bitte ne Mail: Socma(Q)gmx.net . Für weitere Anregungen, Lob, Kritik.... könnt ihr mich natürlich auch anschreiben.
Referenzen (neben denen, die bereits im Text erwähnt wurden):

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