IDS – Intrusion Detection System, Teil I

by Klaus Müller
<Socma(at)gmx.net>

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

Abstract:

Dies ist der erste Artikel in einer Serie über Intrusion Detection Systeme. Im diesem ersten Teil werden nicht nur typische IDS–Architekturen vorgestellt, sondern vor allem typische Angriffe gegen Intrusion Detection Systeme behandelt und analysiert.

Einleitung

Bevor ich mit dem eigentlichen Text anfange, vorab einige Dinge, damit keine Missverständnisse entstehen: IDS steht für Intrusion Detection System, im Verlauf des Textes verstehe ich unter IDS auch ein System, das Einbrüche erkennt und Gegenmaßnahmen einleitet (wobei das Einleiten von Gegenmaßnahmen ein optionales Feature ist). Anfangs werde ich erst einmal eine Übersicht über die verschiedenen Arten von IDS geben, um anschließend näher auf verschiedene Taktiken eingehen zu können. Im letzten Teil werden dann diverse Programme wie Snort, bofh ... und mein eigenes Projekt COLOID besprochen.
Vorab einige Erläuterungen zu Begriffen, die ich nicht direkt im Text erklären werde :

- false positive = Falscher Alarm, also ein Alarm, dass ein Angriff stattgefunden hat, obwohl dies nicht der Fall war
- false negative = Das IDS erkennt einen Angriff nicht als solchen und filtert ihn nicht

IDS–Übersicht

Sinn dieses Artikels ist, die verschiedenen Arten von Intrusion Detection Systemen vorzustellen, gleichzeitig auch ihre Vor– und Nachteile. Vorab aber erstmal ein paar einleitende Worte zum Sinn und Zweck von Intrusion Detection Systemen. Ein IDS beobachtet sozusagen die Vorgänge, die auf dem jeweiligen Rechner

oder im jeweiligen Netzwerk vorgehen, anhand von verschiedenen Methoden, die ich hier vorstellen werde, wird versucht herauszufinden, ob die Sicherheit der Rechner bedroht ist (ein "Angriff" vorliegt), um anschließend Aktionen dagegen einzuleiten. Da meistens zusätzlich Log−Files erstellt werden, gehört es auch zu einer Hauptaufgabe diese zu analysieren und bei Anzeichen eines Einbruchs oder einem illegalen Versuch eines Users seine Rechte zu erweitern, zu reagieren.
Grundsätzlich lassen sich folgende drei Arten von Aufgabenbereichen spezifizieren:

- Information Sources
- Response
- Analysis

Response und Analysis werden später nochmals genauer behandelt, die verschiedenen Arten der "Information Sources" gleich. Aufgrund der umfassenden Möglichkeiten einen PC oder ein Netzwerk "anzugreifen", gibt es auch verschiedene Arten von IDS (diese lassen sich natürlich auch miteinander kombinieren), die sich grundsätzlich darin unterscheiden, wo sie kontrollieren und was sie kontrollieren (daher Information Sources − Quellen).

Host−Based Intrusion Detection

Host−Based Intrusion Detection Systeme analysieren Informationen auf einem einzelnen Host. Da sie sich normalerweise nicht um den gesamten Netzwerkverkehr kümmern müssen, sondern "lediglich" die Vorgänge auf dem eigenen PC, können sie meist präzisere Angaben über die Art des Angriffs geben. Außerdem sieht man hier direkt die Auswirkungen auf dem jeweiligen PC, also ob ein bestimmter User erfolgreich einen Angriff gestartet hat. Host−Based IDS benutzen meist zwei verschiedene Arten, Auskunft über die Vorgänge zu geben : System Logs und Operating System Audit Trails. Beide haben ihre Vorteile, so sind Operating System Audit Trails meist exakter, ausführlicher und können daher besser Auskunft über die Vorgänge geben, während System Logs meistens nur die allerwichtigsten Informationen liefern und daher wesentlich kleiner sind. System Logs lassen sich aufgrund ihrer Größe natürlich besser kontrollieren und analysieren, doch wie gesagt, beides hat seine Vor− und Nachteile.

Vorteile:

- ihr "seht" die Auswirkungen eines Angriffs und könnt daher besser darauf reagieren
- ihr könnt Trojanische Pferde... besser erkennen, da die zur Verfügung stehenden Informationen / Möglichkeiten sehr groß sind
- sie können Attacken erkennen, die Network−Based IDS nicht erkennen, da dort der Verkehr oft verschlüsselt ist
- man kann Vorgänge auf seinem Rechner exakt verfolgen

Nachteile:

- sie können schlechter Scans erkennen
- sie sind anfälliger gegenüber DoS−Attacken
- die Analyse der Operating System Audit Trails ist aufgrund ihrer Größe sehr zeitaufwendig
- sie strapazieren die Rechnerleistung des jeweiligen PCs (teilweise) immens

Beispiele:

- Tripwire [http://www.tripwire.com/products/index.cfml]
- SWATCH [http://freshmeat.net/redir/swatch/10125/url_homepage/swatch]
- DragonSquire [http://www.enterasys.com/ids/squire/]

- Tiger [http://freshmeat.net/redir/tiger-audit/30581/url_homepage/tiger]
- Security Manager [http://www.netiq.com/products/sm/default.asp]

Network Intrusion Detection (NIDS)

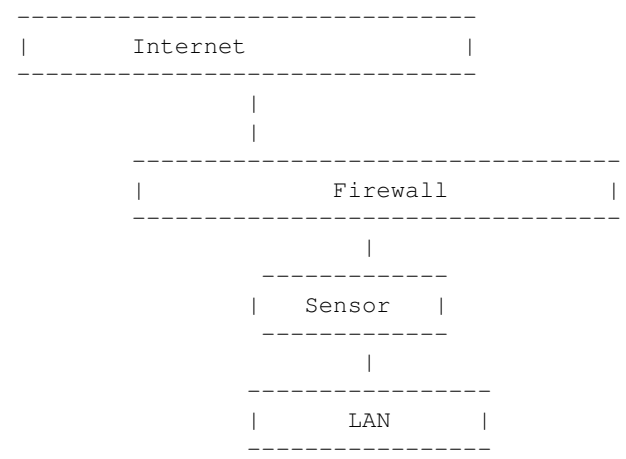
Hauptaufgabe eines Network IDS ist das Analysieren und Auswerten der Pakete, die über das Netzwerk transferiert werden. Um die Pakete nach "auffälligem" Inhalt wie z.B. /etc/passwd zu untersuchen, werden Signatures erzeugt, doch darauf wird während des Textes noch eingegangen. Meist werden sogenannte Sensoren (oft einzelne Hosts) verwendet, die eben nichts anderes machen als den Verkehr zu analysieren und gegebenenfalls Alarm auszulösen. Da dies ihre einzige Aufgabe ist, besteht die Möglichkeit diese Sensoren besser zu schützen. In diesem Zusammenhang taucht auch öfter der sogenannte "Stealth Mode" auf, also dass die Sensoren praktisch "unsichtbar" agieren, um es dem Angreifer schwieriger zu machen, sie zu lokalisieren oder anzugreifen.

"Stealth mode can be used for network sensors to make the promiscuous mode network interface card (NIC) invisible because it simply doesn't have an IP address in this mode. This is achieved by using a separate NIC in each network sensor machine to communicate with the console over, usually, a physically isolated secure network. Stealth mode makes it more difficult for a hacker to attack the network sensor itself."

Dieser Abschnitt (aus der Beschreibung zu RealSecure entnommen) verdeutlicht nochmals, was es heisst, wenn sich ein Sensor im Stealth Mode befindet. Der Sensor switcht also in den Promiscuous Mode (vereinfacht gesagt, der Modus in dem das Netzwerkgerät den gesamten Verkehr des Netzwerks mitliest) und besitzt keine eigene IP, dadurch soll es dem Angreifer möglichst schwer gemacht werden, den Sensor zu lokalisieren. Dieser Zustand wird übrigens von Paket-Sniffern wie tcpdump genutzt...

Grundsätzlich kann man die Sensoren in folgende Bereiche platzieren:

Innerhalb der Firewall (vereinfacht):



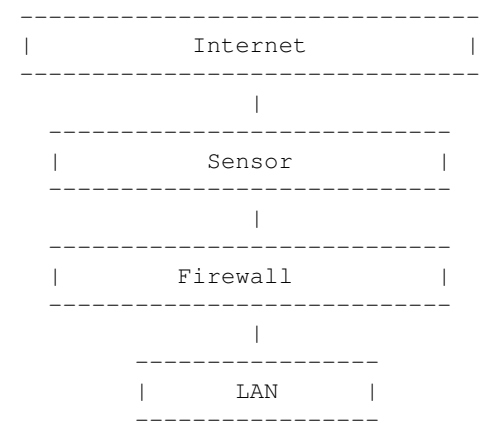
Diese Darstellung ist nur eine mögliche und soll nur verdeutlichen, dass die Sensoren nicht zwischen DMZ und Firewall stehen. Sollte euch der Begriff DMZ unbekannt sein, er steht für Demiltarized Zone und ist ein Bereich, der von allen Seiten geschützt wird.

Wenn die Sensoren innerhalb der Firewall sitzen, ist es für sie besser möglich festzustellen, ob eventuell eine Firewall falsch konfiguriert wurde, außerdem "erfährt" man, ob ein potentieller Angriff durch kam oder nicht.

Erfahrungsgemäß erzeugen Sensoren, hier auch weniger False Positives, da sie "unauffälliger" agieren und deshalb nicht so viel Traffic auf sie zu kommt (womit die Wahrscheinlichkeit eines falschen Alarms sinkt).

Sensoren, die innerhalb der Firewall platziert sind, dienen der Einbruchserkennung, sollen eure Sensoren diesen Zweck erfüllen, solltet ihr sie demnach innerhalb der Firewall platzieren...

Außerhalb der Firewall (vereinfacht):



Sensoren werden häufig, wie in der Darstellung gezeigt, außerhalb der externen Firewall platziert. Der Grund hierfür liegt darin, dass der Sensor so den ganzen Verkehr aus dem Internet empfängt und analysieren kann. Wenn man den Sensor hier platziert, ist noch nicht garantiert, dass wirklich alle Angriffe gefiltert und erkannt werden, z.B. bei TCP−Attacken. In diesem Fall würde man versuchen, die Attacke zu erkennen, indem man sogenannte Signatures verwendet (mehr dazu siehe in dem Teil über Signaturen). Nichtsdestotrotz wird von vielen Experten dieser Standort bevorzugt, da er den Vorteil hat, die Attacken (die auf die Firewall...) ausgeübt werden, zu protokollieren und analysieren, dadurch kann der Admin sehen, wo er eventuell die Konfiguration der Firewall umstellen sollte.

Sensoren, die außerhalb der Firewall platziert sind, dienen der Angriffserkennung, (Unterschied zur Platzierung innerhalb der Firewall!). Sollen eure Sensoren also die Angriffe erkennen, solltet ihr sie außerhalb der Firewall platzieren...

− Außerhalb und innerhalb der Firewall

Nun, diese Variante verbindet die beiden zuvor genannten Varianten. Allerdings ist hier die Gefahr groß, dass man die Sensoren und/oder die Firewall falsch konfiguriert, d.h. man kann nicht einfach die Vorteile der beiden Varianten zusammenrechnen und dieser Variante hier zuschreiben. Diese Möglichkeiten, die Sensoren zu platzieren, sind nicht die einzigen, man könnte sie natürlich noch anders platzieren, doch die oben genannten Standorte sind wohl die am häufigsten verwendeten.

Vorteile:

- die Sensoren können gut geschützt werden, da sie "nur" den Verkehr überwachen
- man kann besser Scans erkennen − anhand von Signaturen... kann man den Verkehr "filtern" (naja, dass es nicht immer so ist, wird noch gezeigt)

Nachteile:

- die Wahrscheinlichkeit von sogenannten false negatives (also Angriffen, die nicht als Angriffe enttarnt werden) ist hoch, da es schwierig ist, das gesamte Netzwerk zu kontrollieren
- meistens müssen sie auf verschlüsselter Ebene operieren, wodurch die Analyse der Pakete erschwert wird
- im Gegensatz zum Host−Based IDS sehen sie nicht die Auswirkungen eines Angriffs

Beispiele:

- NetRanger [http://www.cisco.com]
- Dragon [http://www.securitywizards.com]
- NFR [http://www.nfr.net]
- Snort [http://www.snort.org]
- DTK [http://all.net/dtk/dtk.html]
- ISS RealSecure [http://www.uh.edu/infotech/software/unix/realsecure/index.html]

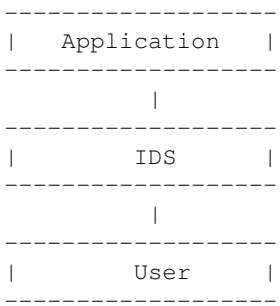
Obwohl ich hier zwischen HIDS und NIDS unterscheide, wird der Unterschied zwischen beiden ID−Systemen immer geringer, da mittlerweile auch HIDS die Grundfunktionalität eines NIDS haben. Bekannte ID−Systeme wie ISS RealSecure bezeichnen sich auch nicht nur als NIDS, sondern als "host−based and network−based IDS". In Zukunft wird der Unterschied zwischen beiden Systemen also immer geringer werden (so dass die beiden Systeme mehr und mehr "zusammenwachsen").

Network Node Intrusion Detection (NNIDS)

Grundsätzlich arbeitet dieser neue Typ (NNIDS) so wie die typischen NIDS, d.h. man nimmt sich Pakete aus dem Netzwerkverkehr und analysiert diese. Allerdings betrifft das nur Pakete, die an den Network Node adressiert sind (daher der Name). Ein anderer Unterschied zwischen NNIDS und NIDS besteht darin, dass NIDS im Promiscuous Mode laufen, während NNIDS nicht im Promiscuous Mode laufen. Dadurch, dass längst nicht jedes Paket analysiert wird, wird die Performance des Systems nicht zu sehr darunter leiden, bzw. solche Systeme laufen in der Regel sehr schnell.

Application Based Intrusion Detection

Application Based IDS sind eine Untergruppe der Host−Based IDS, dennoch erwähne ich sie hier getrennt. Es überwacht die Interaktion zwischen User und Programm, wobei hauptsächlich zusätzliche Log−Files erzeugt werden, um Auskunft über die Vorgänge zu geben. Da man direkt zwischen User und Programm operiert, kann man hier sehr leicht auffälliges Verhalten "rausfiltern". Sinnbildlich könnte man sich ABIDS so vorstellen:



Vorteile:

- man arbeitet auf unverschlüsselter Ebene, im Gegensatz zu z.B. Network Based IDS, dadurch ist eine Analyse besser möglich
- man kann "auffällige" Kommandos, die der User auf/mit dem Programm ausüben will, schnell erkennen und verhindern

Nachteile:

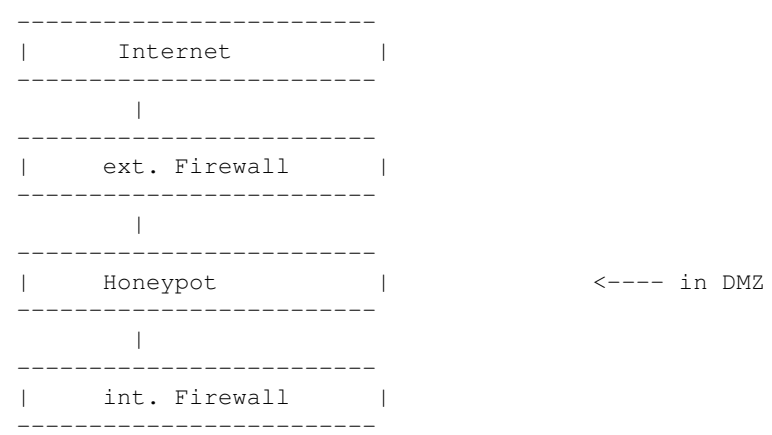
- kein Schutz und nur wenig Möglichkeiten, z.B. Trojanische Pferde zu erkennen (da man nicht im Kernelspace agiert)
- Die von dieser Art der IDS erzeugten Log−Files sind leichte Angriffsziele für "Angreifer" und nicht so sicher wie z.B. Operating System Audit Trails

Stack Based Intrusion Detection

Auch eine neuere Art der IDS ist das sogenannte Stack Based Intrusion Detection System (SBIDS). Momentan stehen mir allerdings nicht genügend Informationen zur Verfügung, die es mir ermöglichen würden, über diesen IDS−Typ ausreichend zu informieren.

Honeypot

Für Honeypots gibt es viele Assoziationen, damit keine Missverständnisse auftauchen, hier eine recht gute Definition aus der SANS Institute's Intrusion Deception FAQ : "Honeypots are programs that simulate one or more network services that you designate on your computer's ports. An attacker assumes you're running vulnerable services that can be used to break into the machine. A honeypot can be used to log access attempts to those ports including the attacker's keystrokes. This could give you advanced warning of a more concerted attack". In manchen Fällen ist ein Honeypot einfach eine "Box". Nach aussen hin erscheint sie ungeschützt, dabei protokolliert sie den Verkehr und analysiert ihn auch. Dadurch, dass Honeypots ungeschützt erscheinen und "normalerweise" keine Verbindung zu ihnen hergestellt werden sollte, wird jede Verbindung zum Honeypot als verdächtig erachtet.



Das interne Netzwerk muss also nicht offengelegt werden, da ein Honeypot normalerweise in der DMZ (DeMilitarized Zone) platziert wird. Hauptaufgabe eines Honeypots besteht meist darin, den Verkehr zu

analysieren, z.B. wann bestimmte Prozesse gestartet wurden, welche Dateien verändert wurden..., so dass man recht gut ein Profil von potentiellen Angreifern erstellen kann.

Doch Honeypot ist nicht gleich Honeypot, so dass man zwischen drei "Varianten" unterscheidet, die auf unterschiedliche Weise die Sicherheit des Systems erhöhen:

Prevention: Verhinderung der Attacke−Detection: Melden, dass eine Attacke vorliegt (möglichst auch Lokalisierung des Angreifers, was für ein Angriff, welche Dienste sind betroffen...?) Reaction: Die (Gegen)reaktion, Detection alleine bringt nichts, wenn man nichts dagegen unternimmt. Padded Cell Systems bieten besondere Möglichkeiten, die Gegenmaßnahmen betreffend. Reaction beinhaltet also was man oder das IDS macht, wie man auf einen Angriff reagiert.

Später werden die verschiedenen Response Options von IDS nochmal geklärt... Außerdem lassen sich Honeypots noch in zwei größere Kategorien einteilen, die Research und die Production Honeypots. Production Honeypots sollen dabei helfen (potentielle) Risiken in einem Netzwerk zu lindern, währenddessen Research Honeypots dem Sammeln und Erfragen von Informationen über den/die Angreifer dienen.

Prevention:
Honeypots sind sicherlich nicht die geeignetsten Systeme zur Verhinderung von Angriffen. Wie ihr später noch bei Padded Cell Systems sehen werdet, kann ein falsch programmierter und konfigurierter Honeypot Angriffe noch erleichtern, wobei dies wohl für den gesamten Themenbereich IDS gilt.

Detection:
Hier liegt wohl der große Vorteil von Honeypots, da sie ausgezeichnet sein können, um Angriffe zu erkennen. Hierbei können sie vor allem System Logs analysieren und auswerten. Die Platzierung des Honeypots spielt eine entscheidene Rolle, bzw. ein Admin kann nur dann von Honeypots profitieren, wenn sie richtig konfiguriert und platziert sind. Oft werden sie zwischen wichtige Server platziert, um eventuelle Scans auf das gesamte Netzwerk zu erkennen oder in die Nähe eines wichtigen Servers, um mit Hilfe von Port Redirection illegalen Zugriff auf den Server feststellen zu können (z.B. jemand versucht über bestimmte Ports auf den Server zuzugreifen, er wird umgeleitet zum Honeypot, da dieser Zugriff nicht gestattet sein sollte, wird eine Warnung ausgegeben.)

Reaction:
Welche Möglichkeiten ein Honeypot hat, auf Ereignisse zu reagieren, seht ihr in dem Teil über Responses (weiter unten). Bei der Verwendung/Entwicklung eines Honeypots sollte man darauf achten, dass der Rechner möglichst attraktiv für einen Angreifer scheint, d.h. es sollte nicht schwer sein, den Rechner anzugreifen. Grundsätzlich sollten nur wenige Änderungen an der Default−Konfiguration gemacht werden, da zu viele Änderungen nur auffällig wirken könnten. Nichtsdestotrotz sollte der eigentliche Sinn nicht vergessen werden, also Kontrollieren des Traffics, Protokollieren der Aktivitäten, ... Ein Ansatz von dem ich schon öfter gehört habe, ist es, den Honeypot in ein eigenes Subnetz hinter eine Firewall zu platzieren. Diese besitzt meist Alarm−Fähigkeiten und kann daher auch Warnungen ausgeben. Da die Logs begehrte Ziele der Angreifer sind, sollte man sie nicht auf dem Rechner selbst protokollieren, sondern an einen anderen Server schicken. Manchmal werden auch Sniffer benutzt, um den Verkehr nach bestimmten Zeichen zu durchsuchen und den Verkehr zu "protokollieren". Sollte man genug Informationen gesammelt haben, sollte man die Logs untersuchen und nach Schwächen des Netzwerks schauen, bzw. diese bereinigen. Durch die Fülle an Informationen sollte es leichter sein, Sicherheitslücken zu schließen und bei schwereren Angriffen auch gegen den Angreifer vorzugehen. Allerdings muss man auch beachten, dass Honeypots nicht ganz legal sind, bzw. müsst ihr bei der Verwendung von Honeypots aufpassen. Das Problem besteht darin, dass der Honeypot als "Einladung zum Angriff" aufgefasst werden könnte, und wenn man merkt, dass der Rechner angegriffen wurde (und man keine Gegenmaßnahmen einleitet), dies zusätzlich als "grobe Fahrlässigkeit" ausgelegt werden könnte. Manche gerichtlichen Argumentationen gegen Honeypots klingen schon banal, dennoch solltet ihr euch vorher erkundigen, wie das Gesetz es bei euch sieht. Wird von eurem Netzwerk aus ein

anderes Netzwerk angegriffen und dort entsteht Schaden, werdet ihr (wahrscheinlich) dafür bezahlen müssen, aus bereits genannten Gründen.
Weitere Recherchen ergaben allerdings, dass der Einsatz von Honeypots in z.B. Europa kein Problem darstellt (solltet ihr einen Paragraphen finden, der dies widerlegt, schreibt mir bitte. Bei meiner Suche habe ich nämlich keinen solchen Paragraphen gefunden...)

Padded Cell Systems

Diese Systeme arbeiten normalerweise mit einem der anderen, bereits genannten Systeme zusammen. Sollte von dem verwendeten IDS gemeldet werden, dass ein Angriff vorliegt, so wird der entsprechende Angreifer zu einem "padded cell host" weitergeleitet. Sobald sie einmal in diesem Padded Cell Host sind, können sie keinen Schaden mehr anrichten, da die gesamte Umgebung eine simulierte, eine "Scheinumgebung" ist. Wichtig dabei ist, dass diese Umgebung so realistisch wie möglich dargestellt werden muss, bzw. der Angreifer muss denken, dass der Angriff erfolgreich war (sozusagen). In diesem Padded Cell Host besteht die Möglichkeit jegliche Aktivität des Angreifers zu protokollieren, zu analysieren und mitzuverfolgen. Das Problem bei der Verwendung von Padded Cell Systems ist, dass es u.U. nicht legal ist, es in dem jeweiligen Land einzusetzen (da eventuell entsprechende Gegen−Aktionen des IDS als "Angriff" empfunden werden könnten, obwohl sie nur dem Sammeln von Informationen über den richtigen Angreifer dienen).

Vorteile:

- einmal in dem Padded Cell Host, können die Angreifer keinen Schaden mehr anrichten, da es nur eine "Scheinumgebung" ist
- der Admin kann die Aktionen des Angreifers direkt verfolgen / protokollieren, um mehr Informationen zur Attacke und dem Ziel der Attacke zu bekommen, somit ist es auch leichter Gegenmaßnahmen einzuleiten

Nachteile:

- eventuell könnte die Verwendung von Padded Cell Systems nicht legal sein (hier gilt wiederum das Gleiche wie für Honeypots, in Europa sollte es eigentlich kein Problem sein)
- die Umsetzung eines solchen Systems ist sehr schwer und erfordet einiges an Können, schließlich muss die gesamte Umgebung richtig simuliert werden. Sollte der Admin irgendwo einen kleinen Fehler gemacht haben, könnte dieses System durchaus weitere Sicherheitslücken eröffnen

Angriffsarten

Bevor man ein IDS entwickelt oder benutzt, sollte man die potentiellen Gefahren spezifieren, also auch welche Angriffe man zu erwarten hat. Trotz der vielen verschiedenen Möglichkeiten lassen sich Angriffe und ihr Ziel dennoch in vier Kategorien einteilen:

1)Confidentiality
Der Angriff hat zur Folge, dass sich die Vertrauensverhältnisse gegenüber dem bestimmten User (meist zu seinen Gunsten ;) geändert haben, also z.B., dass man sich nicht mehr authentifizieren muss gegenüber einem bestimmten Programm... Solche Fälle werden auch heute noch missbraucht, z.B., wenn zwischen zwei PCs ein "Vertrauensverhältnis" besteht, d.h. der User des einen PC kann sich ohne Passworteingabe beim anderen einloggen (oder sonst was). Erreicht ein Angreifer ein vergleichbares Ergebnis mit seinem Angriff, so ist es manchmal sehr schwer solche "missbrauchten" Vertrauensverhältnisse aufzudecken.

2)Integrity
Wenn der User wichtige Konfigurations− und Systemdateien verändert, bestehende Binaries durch seine eigenen ersetzt... Oft werden bestehende Binaries... (wie z.B. /bin/login) durch eigene Binaries ersetzt. Dort muss dann der entsprechende User nur ein (im Source) festgelegtes Passwort eingeben und bekommt (meist) root−Rechte. Solche Angriffe dienen also auch der Erweiterung seiner eigenen Rechte, z.B. durch das Einbauen einer Login−Backdoor. Zwar gibt es Tools wie Tripwire, doch längst nicht jeder benutzt es. Außerdem werden in der Konfiguration und Benutzung oft verheerende Fehler gemacht, die Tripwire in solchen Fällen zu einem weiteren Risiko machen.

3)Availability
Hierdurch wird die Erreichbarkeit des Systems beeinträchtigt. Dies könnte sich äussern in dem Verbot, dass bestimmte User sich überhaupt einloggen dürfen oder dass man sich nur zu bestimmten Zeiten einloggen darf... Ziel ist es meist, dass man selbst "ungestört" arbeiten kann und von keinem User entdeckt wird.

4)Control
Z.B. wenn der Angriff der "Übernahme" des Systems gilt, also Kontrolle über Dateien, Programme... Hierbei solltet ihr beachten, dass es einem Angreifer natürlich auch die anderen drei Möglichkeiten eröffnet. Sollte er die totale Kontrolle über das System haben, kann er verändern, einschränken ... wie er will.

Bevor ich nun auf die verschiedenen Angriffsarten (DoS, DDoS, Scans, ...) zu sprechen komme, erstmal ein kleiner Ausflug in die Welt der Integrity Checker. Tools wie Tripwire zählen zu den Host−Based IDS, zu der Aufgabe eines Integrity Checkers gehört es, die Integrität diverser Dateien zu überprüfen und gegebenenfalls eine Warnung auszugeben, wenn man Änderungen an einer Datei erkennt. Das Vorgehen bei der Verwendung von Integrity Checkern ist meist das gleiche, daher ist die Beschreibung eines bestimmten Integrity Checkers wohl nicht notwendig...

1) Erstellen einer Datenbank
Der erste Schritt nach der Installation eines Integrity Checkers wie Tripwire ist das Erstellen einer Datenbank. Dieser Schritt sollte (muss) auf jeden Fall zu einem Zeitpunkt vollzogen werden, in dem der Zustand des Systems keinesfalls fragwürdig ist. Erstellt man eine Datenbank zu einem späteren Zeitpunkt (und der Angreifer hat vielleicht schon bestehende Binaries ersetzt), würde die Verwendung eines Integrity Checkers den eigentlichen Sinn nicht erfüllen. In einem solchen Fall würden die ersetzten Binaries als "Originale" betrachtet werden. Sollte der Admin später diese Binaries wiederum durch die eigentlichen "Orignale" ersetzen, würde ein Alarm ausgelöst... Die meisten Tools bieten umfangreiche Möglichkeiten, die Dateien / Verzeichnisse zu spezifizieren, von denen man eine Hashsumme... erstellen möchte. Man könnte also vereinfacht ausgedrückt sagen, dass ein Fingerabdruck vom System angefertigt wird. (Der Modus heisst übrigens bei Tripwire "Database Initialization Mode")

2) Check der Integrität
Nachdem der Admin eine Datenbank (bzw. den Fingerabdruck) des Systems hat, kann er zu jeder Zeit sein System erneut überprüfen. Mit der erstellten Datenbank als Fundament, ermittelt er erneut die Hashsumme... vergleicht diese mit der Datenbank, und wenn die Werte differenzieren, wird ein Alert ausgegeben. Tripwire bietet hier noch mehr Möglichkeiten, lest dafür einfach mal die Manpage zu Tripwire oder eine zugehörige Dokumentation.

Diese beiden Schritte sind eigentlich beim Umgang mit den meisten Integrity Checkern vorzufinden. Tripwire bietet noch die Möglichkeit, die Datenbank zu aktualisieren (man hat neue Tools installiert...) oder die Policy File zu aktualisieren, das EMail Notification System zu testen...

Welche Fehler kann man nun beim Umgang mit Integrity Checkern machen?

Der erste und mit Abstand gröbste Fehler, den ein Admin machen kann, ist, eine MD5−Hashsummendatenbank zu erstellen, zu einem Zeitpunkt, in dem bestehende Binaries mit großer Wahrscheinlichkeit bereits ersetzt wurden. Geht ein Admin davon aus, dass ein Angriff erfolgreich war und erstellt danach von diesem "kompromittierten" System die Hashsummen−Datenbank, würde die vom Angreifer ersetzte Binary (z.B. Login−Backdoor) als "richtige" Binary gelten. Man sollte also möglichst schnell nach der Installation die Datenbank erstellen. Ein weiterer kapitaler Fehler beim Umgang mit Integrity Checkern (wie z.B. Tripwire) besteht darin, die Hashsummen−Datenbank auf der Festplatte zu lassen. Auf den ersten Blick erscheint es ganz normal, die Datenbank auf der Festplatte gespeichert zu lassen, doch wenn man schon die Datenbank auf der Festplatte lässt, sollte man zumindest darauf achten, dass die Partition / das Medium read−only ist. Es muss also sicher gestellt werden, dass niemand Schreibzugriff auf unsere Datenbank hat. Wäre es dem Angreifer erlaubt, in die Datenbank zu schreiben, könnte er diese nach Belieben verändern. Solltet ihr bereits mit Tripwire gearbeitet haben, wisst ihr sicherlich auch, dass man per Konfigurationsdatei einstellen kann, von welchen Dateien die Integrität überprüft werden soll. Doch was, wenn der Angreifer diese Konfigurationsdatei lesen/beschreiben kann? Er könnte die "Suchpfade" ändern, so dass das Verzeichnis mit den geänderten Binaries gar nicht erst durchgescant wird. Am besten lasst ihr also die Konfiguration auch nicht auf der Platte und packt sie wiederum auf ein read−only Medium. Einigen wird es umständlich erscheinen, die Dateien auf ein read−only Medium zu setzen, allerdings müsst ihr wohl hier etwas mehr Zeitaufwand in Kauf nehmen, zu Gunsten höherer Sicherheit (ein interessanter Aspekt bei Benutzerfreundlichkeit ist ja, dass viele Sicherheitslücken erst dadurch eröffnet werden/wurden, weil man Programme benutzerfreundlich gestalten wollte. In unserem Fall heisst das also, dass man etwas an Benutzerfreundlichkeit einspart, allerdings einen höheren Sicherheitsstandard besitzt). Tripwire (und andere Integrity Checker) sind sicherlich leistungsfähige Programme, die die Sicherheit erhöhen können und es potentiellen Angreifern schwerer machen können, erfolgreich anzugreifen, doch bringt das beste Tool nichts, wenn die sonstigen Sicherheitseinstellungen des Systems nicht stimmen. Verlangt also nicht von Integrity Checkern, dass sie euch die gesamte Arbeit abnehmen, kümmert euch selbst zusätzlich um die Sicherheit eurer Kiste...

Eine neuere Form der Integrity Checker sind die sogenannten Realtime Integrity Checker. Im Gegensatz zu den "normalen" Integrity Checkern überprüfen sie Realtime die Integrität der Datei. Hier ein kurzes Schema:

1) Der erste Schritt ist auch hier das Erstellen einer Hashsummen−Datenbank
2) Bevor jemand ein Programm ausführen kann, wird die Hashsumme der Datei ermittelt. Wenn der Wert mit dem in der Datenbank nicht übereinstimmt, wurde die Binary ersetzt. Liegt ein solcher Fall vor, wird die Ausführung verhindert
Dieses Konzept ist nur ein mögliches Konzept für Realtime Integrity Checker (zumindest habe ich mir dieses Konzept mal so aufgestellt). Im Unterschied zu Integrity Checkern, verlässt man sich also nicht darauf, dass der Admin regelmäßig die Dateien überprüft, es wird also versucht die Korrektheit der Binary vor Ausführung zu überprüfen, um gegebenenfalls die Binary gar nicht erst auszuführen...

Neben den eben aufgeführten Kategorien (Integrity, Control, ...) lassen sich die Angriffe natürlich auch anders aufteilen, z.B. wie angegriffen wird, bzw. mit welchen Mitteln. Auch hier gibt es natürlich viele Möglichkeiten, doch die häufigsten Attacken gegenüber IDS sind wohl:

− Scans
Heutzutage gehören Scans zu alltäglichen "Attacken", das Problem hierbei besteht darin, dass das IDS nicht zu viele False Positives erzeugen sollte. Meistens dienen Scans dem Einholen von (weiteren) Informationen über einen Host / ein Netzwerk (z.B. um anschließend weitere Attacken zu starten). Welche Informationen ein Angreifer über das eigene Netzwerk einholen kann, sieht man beispielweise am folgenden Scan−Ergebnis (nmap − nur ein kleines Beispiel, zeigt noch längst nicht alle Möglichkeiten von nmap):

```
...
Host (192.168.0.0) seems to be a subnet broadcast address (returned 1
extra pings).
```

```
Skipping host. Interesting ports on playground.yuma.net 192.168.0.1):
Port           State          Protocol               Service
22             open           tcp            ssh
111            open           tcp            sunrpc
635            open           tcp            unknown
1024           open           tcp            unknown
2049           open           tcp            nfs

TCP Sequence Prediction: Class = random positive increments
                          Difficulty=3916950 (Good luck!)
Remote operating system guess:
       Linux 2.1.122 - 2.1.132; 2.2.0-pre1 - 2.2.2

Interesting ports on vectra.yuma.net (192.168.0.5):
Port           State          Protocol               Service
13             open           tcp            daytime
21             open           tcp            ftp
22             open           tcp            ssh
23             open           tcp            telnet
37             open           tcp            time
79             open           tcp            finger
111            open           tcp            sunrpc
113            open           tcp            auth
513            open           tcp            login
514            open           tcp            shell

TCP Sequence Prediction: Class = random positive increments
                          Difficulty = 17719 (Worthy challenge)
Remote operating system guess: OpenBSD 2.2. - 2.3

Nmap run completed -- 256 IP addresses (2 hosts up) scanned in 6 seconds
```

Hauptsächlich kann man vor allem also folgendes herausfinden:

- welches Betriebssystem?
- welche Versionen von bestimmten Programmen laufen
- welche Dienste laufen, die man eventuell "angreifen" kann
- welche Ports offen sind
- ...

Viele der verschiedensten Scan−Techniken führen zusammen zu einer Masse von Informationen, durch die es dem Angreifer möglich gemacht wird, seinen Angriff einzuleiten. Zwar werde ich jetzt hier nicht alle Scan−Techniken beschreiben/erwähnen, allerdings werden einige häufig benutzte schon genannt (bei Fragen zu den Protokollen... guckt einfach mal in die entsprechenden RFC):

Ping Scanning:
Ping Scans werden dazu verwendet, um herauszufinden, welche Hosts online sind. Dafür schickt man dem entsprechenden Host (oder den Hosts) ein ICMP Datagramm vom Typ 8 (also Echo Request) und erwartet ein ICMP−Antwort−Datagramm vom Typ 0 (also Echo Reply). Manchmal wird allerdings nicht nur ein Echo Request geschickt, sondern zusätzlich noch ein ACK, da ICMP manchmal gesperrt wird. Wird ein RST zurückgesendet, ist der Host online.

Nmap Parameter: −sP

TCP Scanning (Vanilla) :
Beim TCP Scanning wird (meist) auf alle Ports versucht ein connect() anzuwenden, nachdem der Three−Way−Handshake durchlaufen wurde, also eine Verbindung zu den Ports aufgestellt wurde (bzw. die Verbindung mit den Ports verlief erfolgreich), wird der Rückgabewert von connect() überprüft. Der

Rückgabewert gibt dem Angreifer hierbei Auskunft darüber, ob der verwendete Port (oder die Ports) offen oder geschlossen sind. Ziel eines TCP Scans ist es also herauszufinden, ob Ports offen/geschlossen sind.

Nmap Parameter: −sT

Die folgenden Nmap−Outputs sind von einem meiner Rechner direkt nach einer Standardinstallation durchgeführt worden. Ich habe absichtlich keine Änderungen (an der Konfiguration) vorgenommen:

```
 [Socma]$ nmap -sT localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
```

Ein Ausschnitt eines Tcpdump−Traces (dieses Scans):

```
 02:10:15.804704 Diablo > Diablo: icmp: echo request
                          4500 001c 2db8 0000 3501 5a27 7f00 0001
                          7f00 0001 0800 fc95 fb69 0000
 02:10:15.814704 Diablo > Diablo: icmp: echo reply (DF)
                          4500 001c 0000 4000 ff01 7dde 7f00 0001
                          7f00 0001 0000 0496 fb69 0000
 02:10:15.814704 Diablo.58725 > Diablo.http: . ack 110306597 win 3072
                          4500 0028 d223 0000 2a06 c0aa 7f00 0001
                          7f00 0001 e565 0050 ad48 0003 0693 2525
                          5010 0c00 e718 0000
 02:10:15.814704 Diablo.http > Diablo.58725: R 110306597:110306597(0)
         win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0050 e565 0693 2525 0000 0000
                          5004 0000 a070 0000
 02:10:16.114704 Diablo.1727 > Diablo.821: S 196002918:196002918(0)
 win 32767 <mss 16396,sackOK,timestamp 213509[|tcp]> (DF)
                          4500 003c 8663 4000 4006 b656 7f00 0001
                          7f00 0001 06bf 0335 0bae c466 0000 0000
                          a002 7fff 739c 0000 0204 400c 0402 080a
                          0003 4205
 02:10:16.114704 Diablo.821 > Diablo.1727: R 0:0(0) ack 196002919
        win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0335 06bf 0000 0000 0bae c467
                          5014 0000 d7c4 0000
 02:10:16.114704 Diablo.1728 > Diablo.440: S 195504823:195504823(0)
 win 32767 <mss 16396,sackOK,timestamp 213509[|tcp]> (DF)
                          4500 003c 68b2 4000 4006 d407 7f00 0001
                          7f00 0001 06c0 01b8 0ba7 2ab7 0000 0000
                          a002 7fff 0ecf 0000 0204 400c 0402 080a
                          0003 4205
```

UDP Scanning:

Sinn und Zweck von UDP Scans sind ähnlich dem von TCP Scans, man will offene UDP Ports finden. Ein Scan läuft hier allerdings anders ab, da UDP ein verbindungsloses Protokoll ist (und TCP ein verbindungsorientiertes). Beim UDP Scanning wird ICMP "ausgenutzt", schickt man nun an die jeweiligen Ports 0 Byte UDP Packete, um dann auf die "Antwort" von ICMP zu warten. Wenn gemeldet wird, dass der Port nicht erreichbar ist (oder wie es richtig heisst "Port Unreachable" / Code−Wert 3), bedeutet dies, dass der Port geschlossen ist. Sollte der jeweilige Admin..., z.B. in seiner /etc/inetd.conf bestimmte Dienste deaktiviert haben und man versucht auf den entsprechenden Port ein Paket zu schicken, so hätte dies die Fehlermeldung "Port Unreachable" zur Folge...

Nmap Parameter: −sU

```
 [Socma]$ nmap -sU localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1459 ports scanned but not shown below are in state: closed)
Port       State       Service
111/udp    open        sunrpc

Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

Der zugehörige Tcpdump−Trace:

```
10:41:55.954397 Diablo > Diablo: icmp: echo request
                        4500 001c cc8f 0000 2801 c84f 7f00 0001
                        7f00 0001 0800 8471 738e 0000
10:41:55.954397 Diablo > Diablo: icmp: echo reply (DF)
                        4500 001c 0000 4000 ff01 7dde 7f00 0001
                        7f00 0001 0000 8c71 738e 0000
10:41:55.964397 Diablo.63793 > Diablo.http: . ack 994287972 win 2048
                        4500 0028 79e3 0000 2506 1deb 7f00 0001
                        7f00 0001 f931 0050 06d8 0003 3b43 a164
                        5010 0800 cccd 0000
10:41:55.964397 Diablo.http > Diablo.63793: R 994287972:994287972(0)
     win 0 (DF)
                        4500 0028 0000 4000 ff06 7dcd 7f00 0001
                        7f00 0001 0050 f931 3b43 a164 0000 0000
                        5004 0000 dbb4 0000
10:41:56.274397 Diablo.63773 > Diablo.15: udp 0
                        4500 001c 8a0b 0000 3011 02c4 7f00 0001
                        7f00 0001 f91d 000f 0008 08af
10:41:56.274397 Diablo > Diablo: icmp: Diablo udp port 15
     unreachable (DF) [tos 0xc0]
                        45c0 0038 0000 4000 ff01 7d02 7f00 0001
                        7f00 0001 0303 fb18 0000 0000 4500 001c
                        8a0b 0000 3011 02c4 7f00 0001 7f00 0001
                        f91d 000f
10:41:56.274397 Diablo.63773 > Diablo.1459: udp 0
                        4500 001c 6c2f 0000 3011 20a0 7f00 0001
                        7f00 0001 f91d 05b3 0008 030b
10:41:56.274397 Diablo > Diablo: icmp: Diablo udp port 1459
     unreachable (DF) [tos 0xc0]
                        45c0 0038 0100 4000 ff01 7c02 7f00 0001
                        7f00 0001 0303 fb18 0000 0000 4500 001c
                        6c2f 0000 3011 20a0 7f00 0001 7f00 0001
                        f91d 05b3
```

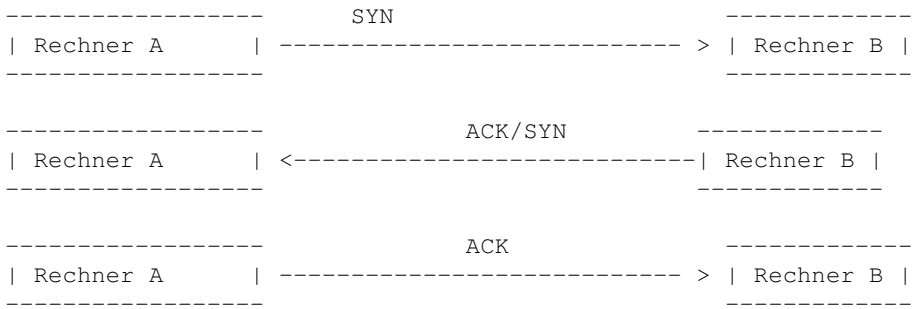
Eine etwas andere Variante eines UDP Scans (UDP recvfrom() and write() Scan) besteht darin, jeden Port zweimal zu scannen. Die eben besprochene Methode benutzt ICMP mit "Port Unreachable", allerdings erhält nur root diese Meldung. Scant man zwei mal einen geschlossenen Port erhält man nach dem zweiten Scan ein "Error 13 : Try Again"...

ACK Scanning:
Durch Senden eines ACK−Pakets an Ports an die Firewall wird herausgefunden, welche Ports gefiltert werden und welche nicht gefiltert werden. Erhält man ein RST zurück, bedeutet es, dass der entsprechende Port "ungeschützt" ist, bzw. nicht gefiltert wird, ansonsten bekommt man eine ICMP Error Message zurück. Zwar wird nicht direkt herausgefunden, welche Ports offen sind, allerdings gewinnt man so genauere Informationen über die Firewall (und deren Einstellungen).

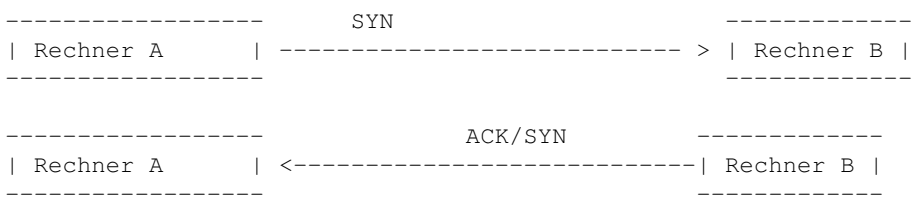
Nmap Parameter : −sA

```
 [Socma]$ nmap -sA localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
All 1558 scanned ports on Diablo (127.0.0.1) are: UNfiltered

Nmap run completed -- 1 IP address (1 host up) scanned in 6 seconds

 Der Tcpdump-Trace:

10:45:51.864397 Diablo > Diablo: icmp: echo request
                         4500 001c 1617 0000 3901 6dc8 7f00 0001
                         7f00 0001 0800 113d e6c2 0000
10:45:51.864397 Diablo > Diablo: icmp: echo reply (DF)
                         4500 001c 0000 4000 ff01 7dde 7f00 0001
                         7f00 0001 0000 193d e6c2 0000
10:45:51.864397 Diablo.53119 > Diablo.http: . ack 2682022466 win 3072
                         4500 0028 0dda 0000 3206 7cf4 7f00 0001
                         7f00 0001 cf7f 0050 0650 0003 9fdc 6a42
                         5010 0c00 c590 0000
10:45:51.864397 Diablo.http > Diablo.53119: R 2682022466:2682022466(0)
     win 0 (DF)
                         4500 0028 0000 4000 ff06 7dcd 7f00 0001
                         7f00 0001 0050 cf7f 9fdc 6a42 0000 0000
                         5004 0000 d7ef 0000
10:45:52.164397 Diablo.53099 > Diablo.14: . ack 2457451592 win 3072
                         4500 0028 218d 0000 3206 6941 7f00 0001
                         7f00 0001 cf6b 000e 5938 4710 9279 bc48
                         5010 0c00 e74d 0000
10:45:52.164397 Diablo.14 > Diablo.53099: R 2457451592:2457451592(0)
     win 0 (DF)
                         4500 0028 0000 4000 ff06 7dcd 7f00 0001
                         7f00 0001 000e cf6b 9279 bc48 0000 0000
                         5004 0000 93a2 0000
10:45:52.164397 Diablo.53099 > Diablo.imap3: . ack 2457451592 win 3072
                         4500 0028 a075 0000 3206 ea58 7f00 0001
                         7f00 0001 cf6b 00dc 5938 4710 9279 bc48
                         5010 0c00 e67f 0000
```

Stealth Scanning (NULL, XMAS, FIN, SYN, ...): Beim Stealth Scanning wird u.a. der Three−Way−Handshake "missbraucht". Zwar ist es eigentlich Vorraussetzung zu wissen, wie dieser abläuft, da man Stealth Scans aber besser daran erklären kann, führe ich hier noch mal kurz den Three−Way−Handshake vor:



Das Problem der TCP−Scans ist, dass sie recht auffällig sind (da jedesmal der Three−Way−Handshake durchlaufen wird). Beim Stealth Scanning passiert stattdessen folgendes:



Diese Darstellung sieht zwar so aus wie beim Three−Way−Handshake, allerdings mit einem wesentlichen Unterschied: In der hier gezeigten Darstellung würde keine Verbindung zwischen A und B bestehen, bzw. Rechner B würde denken, die Verbindung würde stehen, obwohl die Verbindung erst dann steht, wenn A ein weiteres ACK an B schickt (man spricht hier auch von einem "half−open" Port). Der oben gezeigte SYN Scan impliziert, dass der Port auf dem Zielrechner offen ist (aufgrund des ACK/SYN), wäre er geschlossen, so würde man ein RST/ACK zurückerhalten.

Nmap Parameter : −sS

```
 [Socma]$ nmap -sS localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 3 seconds

Tcpdump-Trace:

10:47:41.674397 Diablo > Diablo: icmp: echo request
                         4500 001c 8f08 0000 3501 f8d6 7f00 0001
                         7f00 0001 0800 99a9 5e56 0000
10:47:41.674397 Diablo > Diablo: icmp: echo reply (DF)
                         4500 001c 0000 4000 ff01 7dde 7f00 0001
                         7f00 0001 0000 a1a9 5e56 0000
10:47:41.674397 Diablo.58038 > Diablo.http: . ack 1561498752 win 3072
                         4500 0028 afe5 0000 3206 dae8 7f00 0001
                         7f00 0001 e2b6 0050 82b0 0003 5d12 9480
                         5010 0c00 4e85 0000
```

```
10:47:41.674397 Diablo.http > Diablo.58038: R 1561498752:1561498752(0)
     win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0050 e2b6 5d12 9480 0000 0000
                          5004 0000 dd44 0000
10:47:41.984397 Diablo.58018 > Diablo.1488: S 2803535203:2803535203(0)
     win 3072
                          4500 0028 a4f5 0000 3206 e5d8 7f00 0001
                          7f00 0001 e2a2 05d0 a71a 8d63 0000 0000
                          5002 0c00 88ef 0000
10:47:41.984397 Diablo.1488 > Diablo.58018: R 0:0(0) ack 2803535204
     win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 05d0 e2a2 0000 0000 a71a 8d64
                          5014 0000 94dc 0000
```

Nun kommen weitere Scans ins Spiel: FIN Scanning, NULL Scanning und XMAS Scanning. Beim FIN Scanning wird lediglich dem "Ziel−Rechner" eine FIN−Message geschickt, obwohl keine Verbindung zwischen beiden besteht. Bei einem geschlossenen Port wird nun RST zurückgegeben, ansonsten passiert nichts.

Nmap Parameter : −sF

```
 [Socma]$ nmap -sF localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 6 seconds

Tcpdump-Trace:

10:48:28.704397 Diablo > Diablo: icmp: echo request
                          4500 001c b29d 0000 3401 d641 7f00 0001
                          7f00 0001 0800 a1a7 5658 0000
10:48:28.704397 Diablo > Diablo: icmp: echo reply (DF)
                          4500 001c 0000 4000 ff01 7dde 7f00 0001
                          7f00 0001 0000 a9a7 5658 0000
10:48:28.704397 Diablo.52201 > Diablo.http: . ack 2873378189 win 4096
                          4500 0028 cbeb 0000 2b06 c5e2 7f00 0001
                          7f00 0001 cbe9 0050 9020 0003 ab44 458d
                          5010 1000 54a3 0000
10:48:28.704397 Diablo.http > Diablo.52201: R 2873378189:2873378189(0)
     win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0050 cbe9 ab44 458d 0000 0000
                          5004 0000 f4d2 0000
10:48:29.004397 Diablo.52181 > Diablo.233: F 0:0(0) win 4096
                          4500 0028 10c6 0000 2b06 8108 7f00 0001
                          7f00 0001 cbd5 00e9 0000 0000 0000 0000
                          5001 1000 d522 0000
10:48:29.004397 Diablo.233 > Diablo.52181: R 0:0(0) ack 1 win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
```

```
                        7f00 0001 00e9 cbd5 0000 0000 0000 0001
                        5014 0000 e50e 0000
```

Besonders interessant (vor allem bei der praktischen Umsetzung einer Protocol Anomaly Detection) sind die NULL und Xmas Scans. Der Xmas Scan trägt seinen Namen, aufgrund der Tatsache das dort alle Flags gesetzt sind, also: SYN, ACK, FIN, URG, PUSH. Wie beim FIN Scanning wird auch hier RST zurückgegeben, wenn der Port geschlossen ist.

Nmap Parameter : –sX

```
 [Socma]$ nmap -sX localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds

Tcpdump-Trace:

10:44:24.004397 Diablo > Diablo: icmp: echo request
                        4500 001c ffcc 0000 2a01 9312 7f00 0001
                        7f00 0001 0800 103d e7c2 0000
10:44:24.004397 Diablo > Diablo: icmp: echo reply (DF)
                        4500 001c 0000 4000 ff01 7dde 7f00 0001
                        7f00 0001 0000 183d e7c2 0000
10:44:24.004397 Diablo.36398 > Diablo.http: . ack 718216305 win 2048
                        4500 0028 2e28 0000 2906 65a6 7f00 0001
                        7f00 0001 8e2e 0050 9220 0003 2acf 1c71
                        5010 0800 41f0 0000
10:44:24.004397 Diablo.http > Diablo.36398: R 718216305:718216305(0)
     win 0 (DF)
                        4500 0028 0000 4000 ff06 7dcd 7f00 0001
                        7f00 0001 0050 8e2e 2acf 1c71 0000 0000
                        5004 0000 dc1f 0000
10:44:24.304397 Diablo.36378 > Diablo.1996: FP 0:0(0) win 2048 urg 0
                        4500 0028 7651 0000 2906 1d7d 7f00 0001
                        7f00 0001 8e1a 07cc 0000 0000 0000 0000
                        5029 0800 13d3 0000
10:44:24.304397 Diablo.1996 > Diablo.36378: R 0:0(0) ack 1 win 0 (DF)
                        4500 0028 0000 4000 ff06 7dcd 7f00 0001
                        7f00 0001 07cc 8e1a 0000 0000 0000 0001
                        5014 0000 1be7 0000
```

Die andere Möglichkeit, der NULL Scan, bedeutet, dass kein Flag gesetzt ist. Sollte der Port geschlossen sein, wird wiederum RST zurückgesendet.

Nmap Parameter : –sN

```
 [Socma]$ nmap -sN localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
```

```
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds

Tcpdump-Trace:

10:43:37.594397 Diablo > Diablo: icmp: echo request
                          4500 001c 2ecf 0000 2c01 6210 7f00 0001
                          7f00 0001 0800 8f87 6878 0000
10:43:37.594397 Diablo > Diablo: icmp: echo reply (DF)
                          4500 001c 0000 4000 ff01 7dde 7f00 0001
                          7f00 0001 0000 9787 6878 0000
10:43:37.604397 Diablo.34607 > Diablo.http: . ack 1932747046 win 4096
                          4500 0028 ee0f 0000 3706 97be 7f00 0001
                          7f00 0001 872f 0050 5b20 0003 7333 6126
                          5010 1000 ead5 0000
10:43:37.604397 Diablo.http > Diablo.34607: R 1932747046:1932747046(0)
     win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0050 872f 7333 6126 0000 0000
                          5004 0000 5605 0000
10:43:37.904397 Diablo.34587 > Diablo.408: . win 4096
                          4500 0028 e3bb 0000 3706 a212 7f00 0001
                          7f00 0001 871b 0198 0000 0000 0000 0000
                          5000 1000 192f 0000
10:43:37.904397 Diablo.408 > Diablo.34587: R 0:0(0) ack 0 win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0198 871b 0000 0000 0000 0000
                          5014 0000 291b 0000
```

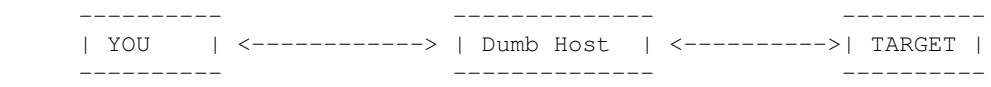
Man benötigt also nicht den kompletten Three−Way−Handshake, dadurch ist Stealth Scanning (wie schon gesagt) "unauffälliger" als TCP Scanning. IDS sollten auf jeden Fall in der Lage sein, diese Anormalitäten (Xmas und NULL) zu erkennen...

FTP Bounce:
In manchen FTPD kann das PORT Kommando dazu missbraucht werden, eine beliebige Verbindung vom Ftp Server zu einer anderen Maschine herzustellen. Doch erst einmal ein kleiner Überblick, wie das ganze "normalerweise" abläuft. Zuerst stellt der Client eine Verbindung zum Ftp Server (Port 21) her, der FTP Server "erstellt" eine zweite Verbindung zurück zum Client (damit er auch Daten zurücksenden kann). Für diese zweite Verbindung wird das PORT−Kommando verwendet. Das interessante hieran ist, dass das Kommando sowohl IP als auch Port (den es zu öffnen gilt) des Client enthält. Anschließend stellt der Server eine Verbindung her, wobei der Source Port 20 und der Destination Port, der Port ist, der bei PORT spezifiziert wurde. Der Angriffspunkt ist nun das PORT−Kommando, bei dem man die IP und den Port des (angeblichen) Client manipulieren kann, um eben eine Verbindung zum Opfer herzustellen, anstatt zu unserem PC. Nachdem also IP und Port manipuliert wurden, kann der eigentliche Verkehr durch "list" oder "get" initialisiert werden. Jetzt überprüft man die Antwort von FTP, denn wenn wir ein "425: Can't build data connection: Connection refused" erhalten, ist der spezifierte Port geschlossen. Erhalten wir hingegen ein "150 : File status okay about to open data connection" oder "226: Closing data connection. Requested file action successful (for example, file transfer or file abort)" als Antwort, wissen wir, dass der spezifizierte Port offen ist.

Nmap Parameter: −b

Fragmented Packets: Bei diesem "Verfahren" wird die Fragmentierung eines IP−Pakets durch TCP "ausgenutzt". Normalerweise kommt es zur Fragmentierung, wenn die Datagramme größer als die verarbeitbare Größe sind, diese Größenbeschränkung wird MTU (Maximum Transmission Unit) genannt. Die fragmentierten Datagramme werden am Ende des Knotens wieder zusammengesetzt. Dieses Verhalten kann aber auch ausgenutzt werden. Nicht jede IDS/Firewall... kommt mit der Fragmentierung klar, d.h. bei fragmentierten Paketen kommt es schon mal zu Fehlern. Anstatt nun unser Paket ganz normal zu verschicken, unterteilen wir es (in Fragmente). Diese beinhalten dann die "gewöhnlichen" Daten wie Source IP, Destination IP, Source Port... Nun kann es sein, dass die angesprungene Firewall/IDS Probleme bei der Zusammensetzung der Fragmente hat. Diese Probleme können sich auf verschiedenste Art und Weise äussern, entweder führt es zu einem Absturz des ganzen Systems, oder das Paket ist durch. Unser Paket könnte möglicherweise durchkommen, da die Zusammensetzung fehlerhaft war und das Paket fälschlicherweise als "harmlos" spezifiert wurde. Manchmal werden auch nicht alle Fragmente richtig kontrolliert, d.h. es wird nur ein bestimmtes Fragment kontrolliert, so dass unser Paket wiederum durchkommen würde. Durch diese Technik kann man also unauffälliger scannen, da der Verkehr als "harmlos" bewertet werden könnte und somit gar nicht erst ein Alarm entsteht. Andererseits beruht diese Theorie darauf, dass die IDS (Firewall) Probleme bei der Bearbeitung und der Zusammensetzung der Fragmente hat.

Reverse Ident Scanning:
Erst einmal ein Ausschnitt aus RFC1413 − dem RFC zum Identifcation Protocol: "The Identification Protocol (a.k.a., "ident", a.k.a., "the Ident Protocol") provides a means to determine the identity of a user of a particular TCP connection. Given a TCP port number pair, it returns a character string which identifies the owner of that connection on the server's system." Beim Reverse Ident Scanning nutzt man identd also aus, um den Eigentümer des laufenden Prozesses zu "erfragen". Diese Technik dient vor allem dem Auffinden von Daemons die als root laufen, um später genau diese Daemons angreifen zu können. Dumb Scanning: Voraussetzung zum Dumb Scanning ist der sogenannte



Hierbei gilt, dass der Dumb Host möglichst wenig Traffic haben sollte, warum, wird am Ende sicherlich deutlicher. Nur wie wird jetzt der Dumb Host genutzt, bzw. warum brauchen wir überhaupt einen? Ok, diese Frage führt zur eigentlichen Attacke und somit auch der Erklärung, warum ein Dumb Host notwendig ist. Damit man später herausfinden kann, ob bei TARGET ein Port offen oder geschlossen ist, muss man das IP ID−Feld des Dumb Hosts untersuchen. Dafür wird dem Dumb Host erst einmal ein Paket zugesendet (Echo Request), anhand seines Replys kann man das ID−Feld, bzw. den Wert, den das ID−Feld hat, auslesen. Anschließend kann man TARGET ein Paket schicken, wobei die Source Address die vom Dumb Host ist. Die Antwort von TARGET erhält daraufhin unser Dumb Host. Sollte er SYN/ACK von TARGET erhalten, so bedeutet das, dass der Port offen ist. Als Antwort wird unser Dumb Host dann ein Paket zurückschicken, in dem RST gesetzt ist. Sollte der Dumb Host allerdings ein RST/ACK von der Target Machine erhalten, schickt er keine Antwort mehr an TARGET. Um nun herausfinden zu können, welche Antwort der Dumb Host vom TARGET bekam, schicken wir dem Dumb Host nochmals ein Ping. Wenn der Port des Targets offen war und somit ein RST zurückgesendet wurde, wird auch das IP ID−Feld des Dumb Hosts inkrementiert, bei geschlossenem Port passiert nichts. Durch Auslesen des neuen IP ID−Wertes des Dumb Host kann man also erkennen, ob die Ports offen oder geschlossen waren. Nun ist hoffentlich auch klarer, warum wir einen Dumb Host verwenden, also einen Host mit möglichst wenig Traffic. Wenn zuviel Traffic auf dem Host herrscht, wäre es schwieriger zu spezifizieren, welche Ports offen sind (bei TARGET), die Wahrscheinlichkeit, dass man sich vertun würde, wäre bei größerem Traffic natürlich dadurch auch höher.

Fingerprint OS Detection:
Die Fingerprint OS Detection hat zum Ziel, das (auf dem Server...) verwendete Betriebssystem zu erkennen.

Die meisten neueren Scanner liefern hier nicht nur z.B. "Linux" oder "Solaris" als Antwort, sondern eine genauere Spezifizierung der Versionen (des jeweiligen Betriebssystems). Hierfür wird ein "Fingerabdruck" (Profil) des Betriebssystems erstellt. Heutzutage kann man eigentlich den Bannern von Telnet, Ftp (siehe oben)... nicht mehr wirklich trauen, da man diese auch ändern/manipulieren kann. Wenn der Angreifer die genaue Version des anzugreifenden Rechners erfährt, kann er darauf aufbauend entsprechende Skripts, Exploits, ... heraussuchen und mit seinem Angriff fortfahren. Diese Technik nutzt aus, dass sich die Betriebssysteme in (den manchmal kleinsten) Dingen unterscheiden (welch Erkenntnis ;). Da es zahlreiche Dokumente zu dieser Technik gibt, werde ich mich hier kurzfassen und nur die wichtigsten Test−Möglickeiten kurz erläutern:

− FIN Test: Wenn ein Rechner ein FIN−Paket an einem offenen Port empfängt, sollte er eigentlich nicht antworten (gemäß RFC793), allerdings gibt es dort auch Ausnahmen, z.B. bei MS Windows, BSDI, CISCO, MPS, ... die ein RESET als Antwort schicken.

− ACK−Sequenznummern: Beim Senden eines FIN|PSH|URG an einen geschlossenen TCP−Port wird meistens die Sequenznummer des ACK−Pakets auf die eigene Sequenznummer gesetzt. Doch auch hier muss Windows mal wieder eine Ausnahme bilden ;) Windows gibt nämlich die eigene Sequenznummer + 1 zurück...

− BOGUS Flag Test: Wird ein undefiniertes TCP Flag im TCP Header verwendet (in einem SYN−Paket), übernehmen Linux Rechner < 2.0.35 diese Flageinstellungen. Andere OS resetten beim Erhalten eines SYN+BOGUS−Pakets...

− TCP Initial Window: Die meisten Betriebssysteme verwenden (nahezu) konstante Fenstergrössen (der Reply−Pakete). Z.B. AIX liefert 0x3F25, MS NT5, OpenBSD und FreeBSD verwenden 0x402E...

− Don't Fragment Bit Test: Einige Betriebssysteme unterscheiden sich darin, dass (oder ob) sie dieses Bit in einigen Paketen setzen oder nicht. Dadurch lässt sich hierdurch zusätzlich differenzieren, welches OS vorliegt...

−TCP Options Test: Die Grundlage dieses Tests ist recht einfach erklärt: Man sendet eine Anfrage an den entsprechenden Rechner, setzt diverse Optionen und guckt in seiner Antwort, ob dort auch die Optionen gesetzt sind. Die in der Antwort enthaltenen Optionen werden unterstützt... Je nach Betriebssystem (und Version) werden bestimmte Optionen grundsätzlich nicht unterstützt, manche schon. Dadurch lässt sich das verwendete Betriebssystem zusätzlich (exakter) bestimmen.

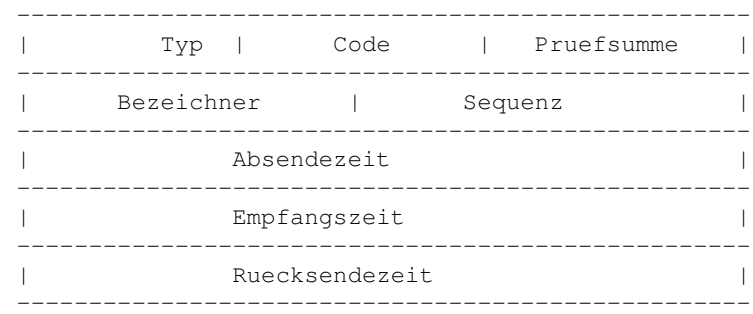
Nmap Parameter : −O

Insgesamt gibt es noch viele nicht besprochene Tests, doch gibt es im Netz genügend Texte über Fingerprint OS Detection. Fyodor (NMAP) hat ein Paper dazu geschrieben, schaut es euch mal an. Dieser kleine Teil sollte lediglich eine kleine Übersicht über weitverbreitete Scan−Techniken geben. Für denjenigen, der sich mit Protocol Anomaly Detection beschäftigt, waren sicherlich einige Ansätze dabei (bestimmte Flagkombinationen, die man als "anormal" betrachten muss...).

Other ICMP Related Stuff
Neben den bereits erwähnten Echo−Reply / Request, stellt ICMP noch weitere Meldungstypen zur Verfügung, durch deren Verwendung man durchaus noch weitere Informationen zum Netzwerk sammeln kann (sogenannte NON − ECHO − REQUESTS):

ICMP Time Stamp Request / Reply (RFC792):
Eigentlicher Sinn eines Time Stamp Request (Typ 13) ist es, die Uhrzeiteinstellungen eines Remote System zu ermitteln. Empfängt der Remote−Rechner einen Time Stamp Request, so sendet er einen Time Stamp

Reply (Typ 14) zurück . Erst einmal zum Aufbau eines Time Stamp (entsprechend der RFC):



Bevor ich auf den Nutzen eines Time Stamp Request für einen Angreifer komme, erst einmal grundsätzlich etwas zum Time Stamp. Für uns sind eigentlich nur die letzten drei "Felder" von Interesse. Hier aber erstmal die entsprechende Stelle im RFC:

```
The Originate Timestamp is the time the sender last touched the
message before sending it, the Receive Timestamp is the time the
echoer first touched it on receipt, and the Transmit Timestamp is
the time the echoer last touched the message on sending it."
```

Entsprechend dem RFC ist der zurückgegebene Time Stamp die Anzahl der Millisekunden, die seit Mitternacht UT (GMT) vergangen ist.

Welchen Nutzen hat aber nun ein Time Stamp Request / Reply für uns?

Erhält man einen Time Stamp Reply zurück, so wüsste man schon mal, dass der Host erreichbar ist, zum anderen kann man anhand der Absende−/Empfangs− und Rücksendezeit Rückschlüsse auf die Belastung des Netzwerkes ziehen (Differenz von Rücksendezeit und Empfangszeit ist entscheidend, natürlich abhängig von den verwendeten Kabeln, Karten...).

Abschliessend noch ein Tcpdump−Trace:

```
11:38:37.898253 Diablo > Diablo: icmp: time stamp query id 53763 seq 64548
(ttl 254, id 13170, len 40)
                4500 0028 3372 0000 fe01 8b60 7f00 0001
                7f00 0001 0d00 61fb d203 fc24 0211 c0ca
                0000 0000 0000 0000

11:38:37.898253 Diablo > Diablo: icmp: time stamp reply id 53763 seq 64548 :
org 0x211c0ca recv 0x211c0ca xmit 0x211c0ca (DF) (ttl 255, id 0, len 40)
                4500 0028 0000 4000 ff01 7dd2 7f00 0001
                7f00 0001 0e00 db43 d203 fc24 0211 c0ca
                0211 c0ca 0211 c0ca
```

ICMP Information Request / Reply (RFC792): "This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on. "

Ein Information Request (Typ 15) hat also den Sinn und Zweck, die Netznummer des Rechners zu ermitteln, der den Request gesendet hat.

```
The address of the source in a information request message will be
the destination of the information reply message.  To form a
information reply message, the source and destination addresses
are simply reversed, ..."
```

Bei einem Information Reply (Typ 16) nimmt man also die Source IP des Information Request als Destination IP des Reply (oder vereinfacht ausgedrückt: Man sendet den Reply an denjenigen, der die Informationen angefordert hat). Als Source IP des Reply nimmt man sich die Destination IP des Request ...

Normalerweise ist es allerdings so, dass der Sender eines Information Request als Destination Address 0 einsetzt (bedeutet dann soviel wie "dieses Netzwerk"). Es besteht allerdings auch die Möglichkeit, sowohl Destination als auch Source IP (beim Absenden des Request) auf 0 zu setzen, in diesem Fall würde der Information Reply sowohl im Source, als auch im Destination Address – Feld die Netznummer des Rechners zugesendet bekommen. Ist das Source Address – Feld beim Request ungleich 0, wird die Netznummer des Rechners nur im Source IP – Feld des Reply zurückgesendet.

```
-----------------------------------------------------------------
|         Typ  |      Code      |   Pruefsumme           |
-----------------------------------------------------------------
|      Bezeichner      |       Sequenz            |
-----------------------------------------------------------------
```

Es scheint, als könnte man nur innerhalb des Netzwerks einen Information Request versenden (s.o.), doch dies muss nicht zwangsläufig so sein. Einige Betriebssysteme antworten auch auf einen Information Request, bei dem die Destination IP nicht im selben Netzwerk liegt. In einem solchen Information Reply würden wir dann die IP des Hosts erhalten (und nicht die Netznummer).

Zum Abschluss wiederum ein kurzer Tcpdump–Trace:

```
11:42:35.608253 Diablo > Diablo: icmp: information request (ttl 255,
 id 13170, len 28)
                         4500 001c 3372 0000 ff01 8a6c 7f00 0001
                         7f00 0001 0f00 1afc d603 0000
11:42:36.608253 Diablo > Diablo: icmp: information request (ttl 255,
 id 13170, len 28)
                         4500 001c 3372 0000 ff01 8a6c 7f00 0001
                         7f00 0001 0f00 19fc d603 0100
```

ICMP Address Mask Request / Reply (RFC 950): Der Address Mask Request (Typ 17) ist in einem anderen RFC beschrieben worden, schaut für nähere Infos also im RFC 950 und nicht im RFC 792 nach. Sinn und Zweck eines Address Mask Request ist es, die Subnetmask des verbundenen Netzes zu ermitteln. Wenn ein Gateway einen solchen Request empfängt, sollte er die entsprechenden Informationen an den entsprechenden Knoten zurücksenden (Address Mask Reply – Typ 18)

```
-------------------------------------------------------------
| Typ  |      Code      |   Pruefsumme                      |
-------------------------------------------------------------
| Bezeichner   |       Sequenz                              |
-------------------------------------------------------------
|               Address Mask                                |
-------------------------------------------------------------
```

Ihr könnt somit nicht nur Hosts im Netzwerk entdecken (die online sind), ihr könntet anhand weiterer Tests auch mehr über die Netzwerk−Konfiguration erfahren ....

Tcpdump−Trace:

```
11:45:26.678253 Diablo > Diablo: icmp: address mask request (ttl 254, id
 13170, len 32)
               4500 0020 3372 0000 fe01 8b68 7f00 0001
               7f00 0001 1100 edd7 dc03 2524 0000 0000
```

Ich hoffe, dieser Abschnitt hat euch gezeigt, dass man weitaus mehr Möglichkeiten hat, Informationen über ein Netzwerk zu erhalten, und dass man nicht immer auf das "normale" ping zurückgreifen muss... In den entsprechenden RFC sind meistens auch Hinweise enthalten, die beschreiben, was man beachten sollte, wenn man die verschiedenen Requests "unterstützen" möchte. Entwickler von ("richtigen") IDS müssen solche Dinge natürlich auch beachten, wenn sie unterstützt werden sollen, muss darauf geachtet werden, dass dies richtig geschieht (s. RFC). Doch auch hier ist noch nicht Schluss, wie ihr im nächsten Abschnitt nachlesen könnt...

Even more information about the Target: Der Einsatz von Paketfiltern (oder allgemeiner Firewalls) ist heutzutage sicherlich nichts seltenes, bzw. besonderes mehr. Auch der Einsatz von sogenannten Firewall Modulen in IDS ist mittlerweile häufiger vorzufinden. Oft hat ein Angreifer gar nicht die Möglichkeiten, einige der besprochenen Scans anzuwenden, da sie geblockt, gefiltert, ... werden. Ziel dieses kleinen Abschnittes ist es, Möglichkeiten aufzuführen, mit denen ihr einige Filter−Rules / Firewall−Rules des Target erarbeiten könnt.

Das Prinzip hinter dieser Idee ist recht einfach, man versucht ICMP−Fehlermeldungen hervorzurufen, bzw. "illegale" Pakete zu versenden, anhand derer man schon die ersten Aussagen über das Ruleset machen kann.

```
If the gateway or host processing a datagram finds a problem with
the header parameters such that it cannot complete processing the
datagram it must discard the datagram.  One potential source of
such a problem is with incorrect arguments in an option.  The
gateway or host may also notify the source host via the parameter
problem message.  This message is only sent if the error caused
the datagram to be discarded."
```

Dieser Abschnitt stammt aus dem RFC 792 und ist ein Teil der Beschreibung des sogenannten Parameter Problem (Typ 12). Wie man diesem Abschnitt entnehmen kann, kann ein Grund für die Meldung "Parameter Problem", ein falscher IP Header sein, d.h. wenn wir einem Rechner ein Paket mit falschem IP Header schicken würden, müssten wir eigentlich diese Fehlermeldung zurückbekommmen. Diese Fehlermeldung hat noch einen weiteren Vorteil, die Unterstützung dieser Fehlermeldung wird nämlich vom RFC 1122 "empfohlen":

```
A host SHOULD generate Parameter Problem messages.  An
incoming Parameter Problem message MUST be passed to the
transport layer, and it MAY be reported to the user.

DISCUSSION:
 The ICMP Parameter Problem message is sent to the
 source host for any problem not specifically covered by
 another ICMP message.  Receipt of a Parameter Problem
 message generally indicates some local or remote
 implementation error."
```

Insgesamt betrachtet, ist dies eine sehr gute Möglichkeit, Hosts im Netzwerk zu erkennen (aus genannten Gründen)...

Ausserdem sei noch auf RFC 1812 verwiesen:

```
 4.3.3.5 Parameter Problem

A router MUST generate a Parameter Problem message for any error not
specifically covered by another ICMP message.  The IP header field or
IP option including the byte indicated by the pointer field MUST be
included unchanged in the IP header returned with this ICMP message.
Section [4.3.2] defines an exception to this requirement.  "
```

Nichtsdestotrotz interpretieren verschiedene Router diese Stelle (und auch weitere) auch unterschiedlich, wodurch nicht wirklich sicher gestellt ist, dass ein Parameter Problem gemeldet wird...

Ein IDS (bzw. Firewalls) sollten dennoch die Felder in dem IP Header überprüfen, da es zwar schon mal dazu kommen kann, dass man ein Paket mit falschem Header erhält, dies sollte aber eigentlich seltener vorkommen. Ein Angreifer, der anhand dieser Methode ein ganzes Netzwerk "scant" (oder sonst einen bestimmten IP Range), weiss schon mal, dass die Firewall / Paketfilter ... diese Fehlermeldung nicht blockiert, filtert..., erhält man hingegen kein Parameter Problem zurück, weiss man schon mal, dass diese Meldung blockiert wird.

Diese Methode zur Feststellung des Ruleset ist allerdings erst der erste Schritt (die Methode stellt vor allem eine Alternative zum "normalen" Ping da). Um eine möglichst exakte Access Control List (ACL) möglicher Filtering / Firewall Software erstellen zu können, müssen wir noch weitere Informationen zur Topologie ... einholen. Um dies zu erreichen, könnte man z.B. die verschiedenen ICMP−Meldungstypen an die einzelnen Hosts schicken (mit falschem IP Header), und darauf warten, wann ein Parameter Problem gemeldet wird. Wenn bei einem Host "X" ein Parameter Problem gemeldet wird, heisst das für uns schon mal, dass der Host diesen ICMP−Meldungstyp nicht filtert (und das der entsprechende Host erreichbar ist). Desweiteren bedeutet das für uns, dass der/die fehlerhaften Einträge im IP Header nicht überprüft werden... Sollten wir kein Parameter Problem erhalten, so können wir auch daraus verschiedenste Schlüsse ziehen. Zum einen wäre es möglich, dass ein Filter den ICMP−Meldungstyp filtert/blockiert, desweiteren wäre es durchaus denkbar, dass der Router ... den Header überprüft und diese Anormalität nicht zulässt ... etc ... Wie ihr seht, kann man aus beiden Ergebnissen Rückschlüsse auf mögliche Filterregeln ziehen, letztendlich erhält man also schon mal eine grobe Vorstellung von dem, was erlaubt ist und was nicht. Weitere Möglichkeiten könnten darin bestehen, die verwendeten Protokolle (TCP, UDP, ...) zu variieren, so dass man weitere Regeln herausfinden könnte. Es bestünde z.B. die Möglichkeit, dass ein bestimmtes Protokoll geblockt/gefiltert wird oder ein bestimmter Port (des Hosts) oder die bereits besprochenen Gründe, könnten der Grund dafür sein, dass kein Parameter Problem zurückgemeldet wird.

Ich denke, die Abhandlung zum Parameter Problem war jetzt ausreichend, so dass wir uns weiteren Fehlermeldungen zuwenden können.

Der folgende Abschnitt ist wiederum dem RFC 1812 entnommen und beschreibt, was ein Destination Unreachable − Fehler ist und was der Grund für diesen Fehler sein kann:

```
The ICMP Destination Unreachable message is sent by a router in
response to a packet which it cannot forward because the destination
(or next hop) is unreachable or a service is unavailable.  Examples
of such cases include a message addressed to a host which is not
there and therefore does not respond to ARP requests, and messages
addressed to network prefixes for which the router has no valid
route.
```

```
A router MUST be able to generate ICMP Destination Unreachable
messages and SHOULD choose a response code that most closely matches
the reason the message is being generated.

0 = Network Unreachable - generated by a router if a forwarding path
     (route) to the destination network is not available;

1 = Host Unreachable - generated by a router if a forwarding path
     (route) to the destination host on a directly connected network
     is not available (does not respond to ARP);

2 = Protocol Unreachable - generated if the transport protocol
     designated in a datagram is not supported in the transport layer
     of the final destination;

3 = Port Unreachable - generated if the designated transport protocol
     (e.g., UDP) is unable to demultiplex the datagram in the
     transport layer of the final destination but has no protocol
     mechanism to inform the sender;

4 = Fragmentation Needed and DF Set - generated if a router needs to
     fragment a datagram but cannot since the DF flag is set;

5 = Source Route Failed - generated if a router cannot forward a
     packet to the next hop in a source route option;

6 = Destination Network Unknown - This code SHOULD NOT be generated
     since it would imply on the part of the router that the
     destination network does not exist (net unreachable code 0
     SHOULD be used in place of code 6);

7 = Destination Host Unknown - generated only when a router can
     determine (from link layer advice) that the destination host
     does not exist;

11 = Network Unreachable For Type Of Service - generated by a router
     if a forwarding path (route) to the destination network with the
     requested or default TOS is not available;

12 = Host Unreachable For Type Of Service - generated if a router
     cannot forward a packet because its route(s) to the destination
     do not match either the TOS requested in the datagram or the
     default TOS (0).         "
```

Wenn wir nun z.B. probieren, an irgendeinen Port ein Paket zu senden, das ein Protokoll verwendet, das gar nicht existiert, sollte eigentlich Destination Unreachable mit Code – Wert 2 (Protocol Unreachable) gemeldet werden. Um bei diesem Beispiel zu bleiben, müsst ihr erst einmal wissen, welche Protokolle "zulässig" sind, dies könnt ihr rausfinden, indem ihr eure /etc/protocols öffnet. Nach der Installation auf einem meiner Rechner, sah die /etc/protocols z.B. so aus:

```
 -------------- /etc/protocols ----------------
# /etc/protocols:
# $Id: protocols,v 1.2 2001/01/29 17:29:30 notting Exp $
#
# Internet (IP) protocols
#
#       from: @(#)protocols     5.1 (Berkeley) 4/17/89
#
# Updated for NetBSD based on RFC 1340, Assigned Numbers (July 1992).
#
# See also http://www.isi.edu/in-notes/iana/assignments/protocol-numbers

ip      0       IP              # internet protocol, pseudo protocol number
```

```
#hopopt 0       HOPOPT          # hop-by-hop options for ipv6
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group management protocol
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
cbt     7       CBT             # CBT, Tony Ballardie
egp     8       EGP             # exterior gateway protocol
igp     9       IGP             # any private interior gateway
                                # (Cisco: for IGRP)
bbn-rcc 10      BBN-RCC-MON     # BBN RCC Monitoring
nvp     11      NVP-II          # Network Voice Protocol
pup     12      PUP             # PARC universal packet protocol
argus   13      ARGUS           # ARGUS
emcon   14      EMCON           # EMCON
xnet    15      XNET            # Cross Net Debugger
chaos   16      CHAOS           # Chaos
udp     17      UDP             # user datagram protocol
mux     18      MUX             # Multiplexing protocol
dcn     19      DCN-MEAS        # DCN Measurement Subsystems
hmp     20      HMP             # host monitoring protocol
prm     21      PRM             # packet radio measurement protocol
xns-idp 22      XNS-IDP         # Xerox NS IDP
trunk-1 23      TRUNK-1         # Trunk-1
trunk-2 24      TRUNK-2         # Trunk-2
leaf-1  25      LEAF-1          # Leaf-1
leaf-2  26      LEAF-2          # Leaf-2
rdp     27      RDP             # "reliable datagram" protocol
irtp    28      IRTP            # Internet Reliable Transaction Protocol
iso-tp4 29      ISO-TP4         # ISO Transport Protocol Class 4
netblt  30      NETBLT          # Bulk Data Transfer Protocol
mfe-nsp 31      MFE-NSP         # MFE Network Services Protocol
merit-inp       32    MERIT-INP       # MERIT Internodal Protocol
sep     33      SEP             # Sequential Exchange Protocol
3pc     34      3PC             # Third Party Connect Protocol
idpr    35      IDPR            # Inter-Domain Policy Routing Protocol
xtp     36      XTP             # Xpress Tranfer Protocol
ddp     37      DDP             # Datagram Delivery Protocol
idpr-cmtp       38    IDPR-CMTP       # IDPR Control Message Transport Proto
tp++    39      TP++            # TP++ Transport Protocol
il      40      IL              # IL Transport Protocol
ipv6    41      IPv6            # IPv6
sdrp    42      SDRP            # Source Demand Routing Protocol
ipv6-route      43    IPv6-Route      # Routing Header for IPv6
ipv6-frag       44    IPv6-Frag       # Fragment Header for IPv6
idrp    45      IDRP            # Inter-Domain Routing Protocol
rsvp    46      RSVP            # Resource ReSerVation Protocol
gre     47      GRE             # Generic Routing Encapsulation
mhrp    48      MHRP            # Mobile Host Routing Protocol
bna     49      BNA             # BNA
ipv6-crypt      50    IPv6-Crypt      # Encryption Header for IPv6
ipv6-auth       51    IPv6-Auth       # Authentication Header for IPv6
i-nlsp  52      I-NLSP          # Integrated Net Layer Security TUBA
swipe   53      SWIPE           # IP with Encryption
narp    54      NARP            # NBMA Address Resolution Protocol
mobile  55      MOBILE          # IP Mobility
tlsp    56      TLSP            # Transport Layer Security Protocol
skip    57      SKIP            # SKIP
ipv6-icmp       58    IPv6-ICMP       # ICMP for IPv6
ipv6-nonxt      59    IPv6-NoNxt      # No Next Header for IPv6
ipv6-opts       60    IPv6-Opts       # Destination Options for IPv6
#       61                      # any host internal protocol
cftp    62      CFTP            # CFTP
```

```
#       63                      # any local network
sat-expak       64      SAT-EXPAK       # SATNET and Backroom EXPAK
kryptolan       65      KRYPTOLAN       # Kryptolan
rvd     66      RVD             # MIT Remote Virtual Disk Protocol
ippc    67      IPPC            # Internet Pluribus Packet Core
#       68                      # any distributed file system
sat-mon 69      SAT-MON         # SATNET Monitoring
visa    70      VISA            # VISA Protocol
ipcv    71      IPCV            # Internet Packet Core Utility
cpnx    72      CPNX            # Computer Protocol Network Executive
cphb    73      CPHB            # Computer Protocol Heart Beat
wsn     74      WSN             # Wang Span Network
pvp     75      PVP             # Packet Video Protocol
br-sat-mon      76      BR-SAT-MON      # Backroom SATNET Monitoring
sun-nd  77      SUN-ND          # SUN ND PROTOCOL-Temporary
wb-mon  78      WB-MON          # WIDEBAND Monitoring
wb-expak        79      WB-EXPAK        # WIDEBAND EXPAK
iso-ip  80      ISO-IP          # ISO Internet Protocol
vmtp    81      VMTP            # Versatile Message Transport
secure-vmtp     82      SECURE-VMTP     # SECURE-VMTP
vines   83      VINES           # VINES
ttp     84      TTP             # TTP
nsfnet-igp      85      NSFNET-IGP      # NSFNET-IGP
dgp     86      DGP             # Dissimilar Gateway Protocol
tcf     87      TCF             # TCF
eigrp   88      EIGRP           # Enhanced Interi
```

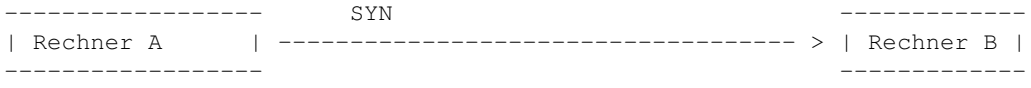
Was, wenn der Host aber kein Protocol Unreachable zurückgibt (obwohl man ein Protokoll verwendet hat, das es eigentlich nicht gibt)? Dies kann nun zwei Ursachen haben, zum einen könnte es sein, dass ihr eine AIX, HP−UX oder Digital Unix Maschine erwischt habt oder aber das Ruleset des Rechners erlaubt den Zugriff auf diese Ports nicht. Stellt also erst mal sicher, was für einen Rechner ihr scant (u.a. mit OS Fingerprint Detection), ansonsten könnt ihr davon ausgehen, dass eben gefiltert/geblockt wird.

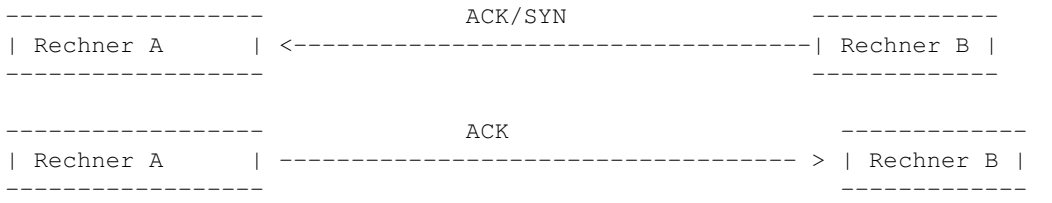
Anmerkung: Die Erkennung einer solchen Attacke ist recht einfach (und gehört übrigens ins Gebiet der Protocol Anomaly Detection). Anomaly Detection wird im nächsten Abschnitt (Analysemöglichkeiten) besprochen, für jetzt reicht es zu wissen, dass der Verkehr nach "Anomalitäten" durchsucht wird, die Verwendung eines nicht vorhandenen Protokolls gehört zu diesen "Anormalitäten".

− Denial of Service (DoS)
DoS (Denial of Service) − Attacken haben meist zum Ziel, den Zielserver lahmzulegen, so dass dieser erst einmal eine Zeit lang nicht erreichbar ist. Hier gibt es viele Techniken, durch die man die Ressourcen des Zielservers völlig "verbrauchen" kann. Im folgenden stelle ich euch die gebräuchlichsten Techniken vor:

ICMP Flooding:
Beim ICMP Flooding wird ICMP "ausgenutzt". Erhält ein Rechner ein ICMP echo request, wird er normalerweise versuchen, darauf mit einem ICMP Echo Reply zu antworten. Dieser Sachverhalt wird beim ICMP Flooding ausgenutzt: Schickt man dem Opfer zu viele Echo Requests, werden dessen Ressourcen sehr lange damit beschäftigt sein, diese Requests zu beantworten (in Form von Echo Replies). Da zusätzlich die IP des Angreifers meist noch gespoofed ist, bekommt nicht er die Replies, sondern jemand anders.

SYN Flooding:
Um SYN Flooding zu verstehen, hier nochmal kurz der "normale" Three−Way−Handshake:





Rechner A schickt also Rechner B ein SYN, um zu sagen, dass er eine Verbindung haben will, B antwortet mit ACK/SYN und wartet nun auf das abschließende ACK, womit die Verbindung vollständig wäre. Doch was ist, wenn das letzte ACK gar nicht erst gesendet wird? Wenn B sein SYN/ACK zurückschickt, wartet er (wie bereits gesagt) schließlich noch auf das ACK von A, solange wird diese noch nicht vollständige Verbindung in die sogenannte Connection Queue von "B" aufgenommen. Sollte der Verbindungsaufbau abgeschlossen sein (also A hat B noch ACK geschickt), wird sie wieder aus der Connection Queue entfernt. Da meist noch die IP−Adresse gespoofed ist, erhält B niemals ein ACK zurück (sollte zumindest so sein). Man kann also einfach die Connection Queue "füllen", jeder weitere Verbindungsversuch würde scheitern, da der Rechner keine weiteren Verbindungen aufnehmen kann.

UDP Flooding:
Bei dieser Flooding−Attacke wird der Zielserver... mit UDP−Paketen überflutet. Schickt man ein UDP−Paket an einen Port auf dem Zielserver, wird erst überprüft, welcher Dienst für diese "Anfrage" zuständig ist. Nun wählt man hier meist willkürliche/zufällige Ports, an die man die Pakete schickt, um die Wahrscheinlichkeit zu erhöhen, dass ein "Port Unreachable" von ICMP gemeldet wird. Als Folge eines solchen Floods, leidet die Performance des Netzwerksegments (oft) erheblich.

Land:
Später werdet ihr einen Filter sehen, der erkennt, ob eine Land−Attacke vorliegt und Gegenmaßnahmen einleiten kann, vorerst aber mal zur eigentlichen Attacke. Bei der Land−Attacke sind sowohl Absender−, als auch Empfänger−IP gleich. Durch Senden eines z.B. SYN−Pakets (mit gleicher Source / Destination IP) an einen offenen Port, wird bei dem Opfer eine Race Condition ausgelöst, die dazu führt, dass das gesamte System lahmgelegt wird. Hier ein Trace einer Land−Attacke:

```
23/06/02  23:12:48  194.157.1.1  80  -> 194.157.1.1
23/06/02  23:14:57  194.157.1.1  31337  ->     194.157.1.1
```

Wie ihr hier nochmals sehen könnt, sind Source IP und Destination IP gleich. Damit Svoern seine Tcpdump−Traces bekommt, hier noch ein Tcpdump−Trace ;) :

```
12:35:26.916369 192.168.38.110.135>192.168.38.110.135: udp 46[tos0x3,ECT,CE]
        4503 004a 96ac 0000 4011 15c7 c0a8 266e
       c0a8 266e 0087 0087 0036 8433 6920 616d
       206c 616d 6520 646f 7320 6b69 6420 6275
       7420 6920 7265
12:35:26.916566 192.168.38.110.135>192.168.38.110.135: udp 46[tos0x3,ECT,CE]
       4503 004a 2923 0000 4011 8350 c0a8 266e
       c0a8 266e 0087 0087 0036 8433 6920 616d
       206c 616d 6520 646f 7320 6b69 6420 6275
       7420 6920 7265
12:35:26.916682 192.168.38.110.135>192.168.38.110.135:udp 46[tos0x3,ECT,CE]
       4503 004a 50a0 0000 4011 5bd3 c0a8 266e
       c0a8 266e 0087 0087 0036 8433 6920 616d
       206c 616d 6520 646f 7320 6b69 6420 6275
       7420 6920 7265
```

Die Attacke, die ihr hier sehen könnt, wird übrigens auch Snork genannt.

Teardrop:
Hier wird die Möglichkeit der Fragmentierung der IP−Pakete genutzt. Wie ihr schon bei der Beschreibung der Scans lesen konntet, kommt es zur Fragmentierung, wenn die Datagramme größer als die verarbeitbare Größe sind, diese Größenbeschränkung wird MTU (Maximum Transmission Unit) genannt. Bei dieser Attacke überlappen sich diese Fragmente, als Folge dieser Überlappung haben (hatten) viele OS Probleme und es kam meistens zum Systemabsturz.

```
10:13:32.104203 10.10.10.10.53>192.168.1.3.53: udp 28(frag 242:36@0+)(ttl64)
       4500 0038 00f2 2000 4011 8404 0a0a 0a0a
       c0a8 0103 0035 0035 0024 0000 0000 0000
       0000 0000 0000 0000 0000 0000 0000 0000
        0000 0000 0000

10:13:32.104272 10.10.10.10 >192.168.1.3: udp 28(frag 242:4@24)(ttl 64)
       4500 0018 00f2 0003 4011 a421 0a0a 0a0a
       c0a8 0103 0035 0035 0024 0000 0000 0000
       0000 0000 0000 0000 0000 0000 0000
```

Ping of Death:
Auch hier spielt die Fragmentierung der IP−Pakete wiederum eine Rolle. Hier will ich allerdings etwas (nur etwas ;) genauer auf die Fragmentierung eingehen. Wie bereits gesagt, bedeutet Fragmentierung, dass die Größe der Datagramme "reduziert" wird, wobei die einzelnen Fragmente natürlich nicht größer als die MTU sein dürfen. Bei der Fragmentierung muss man allerdings noch beachten, dass man nicht beliebig (nach Lust und Laune) fragmentieren darf, bzw. gewisse Felder müssen in jedem Fragment enthalten sein. So muss z.B. jedes Fragment den IP−Protokollkopf enthalten, damit die richtige Route gewählt wird. Damit die Router die Fragmente wieder zu einem Datagramm zusammensetzen können, enthält jedes Fragment eine 16−Bit Kennung (des ursprünglichen "großen" Datagramms). Anhand dieser 16−Bit Kennung ist es also später möglich, die einzelnen Fragmente wieder dem richtigen Datagramm zuzuordnen. Außerdem gibt es noch einen Fragment−Offset, der angibt, an welcher Position das Fragment im ursprünglichen Datagramm stand. Die Position wird allerdings in 8−Oktett−Einheiten bestimmt (da die Position in 8−Oktett−Einheiten gemessen wird). Zusätzlich wird durch das 'More−Bit' gezeigt, ob weitere Fragmente dieses Datagramms folgen oder nicht, steht es auf 1 so folgen noch weitere, steht es auf 0, so war es das letzte Fragment des ursprünglichen Datagramms. Nun aber zum eigentlichen Ping−of−Death−Angriff. Beim Ping−of−Death wird nun dem letzten Fragment ein Offset gegeben für den gilt: Offset + Fragmentgröße > 65535 Bytes . Hierdurch ist es möglich, interne 16−Bit−Variablen zu überfluten, was z.B. einen Systemabsturz als Folge hätte.

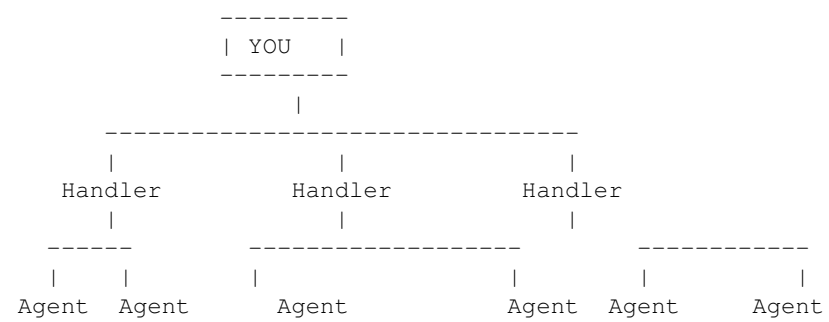
```
17:43:58.431 pinger > target: icmp echo request (frag 4321: 380@0+)
17:43:58.431 pinger > target: (frag 4321: 380@2656+)
17:43:58.431 pinger > target: (frag 4321: 380@3040+)
17:43:58.431 pinger > target: (frag 4321: 380@3416+)
```

Smurf:
Bei dieser Attacke wird zur Broadcastadresse ein Ping geschickt, besser gesagt, viele Pings werden geschickt. An die Broadcastadresse gesandte Pakete werden an alle Rechner des Netzwerks gerichtet und auch von allen "ausgewertet". Wenn man nun viele Pings (ICMP Echo Requests) an die Broadcastadresse schickt (z.B. 10000 pro Sekunde), und man hat ein relativ großes Netzwerk mit 1000 Rechnern, so bedeutet das, dass bei dem Opfer 10000 * 1000 ICMP Echo Replies pro Sekunde eintreffen, also 10 000 000 ICMP Echo Replies.

```
09:28:28.666073 179.135.168.43>256.256.30.255: icmp: echo request (DF)
       4500 001c c014 4000 1e01 6172 b387 a82b
       c0a8 1eff 0800 f7ff 0000 0000 0000 0000
       0000 0000 0000 0000 0000 0000 0000
09:28:28.696073 68.90.226.250>256.256.30.255: icmp: echo request (DF)
       4500 001c c015 4000 1e01 95cf 445a e2fa
       c0a8 1eff 0800 f7ff 0000 0000 3136 3803
       3133 3503 3137 3907 696e 2d61 6464
```

```
09:28:28.726073 138.98.10.247>256.256.30.255:  icmp: echo request (DF)
       4500 001c c016 4000 1e01 27ca 8a62 0af7
       c0a8 1eff 0800 f7ff 0000 0000 0332 3236
       3938 0331 3638 0769 6e2d 6164 6472
09:28:28.756073 130.113.202.100 > 256.256.30.255: icmp: echo request (DF)
       4500 001c c017 4000 1e01 704c 8271ca64
       c0a8 1eff 0800 f7ff 0000 0000 0231 3002
       3938 0331 3338 0769 6e2d 6164 6472
       ...
```

Seit längerer Zeit gibt es auch noch DDoS−Attacken (Distributed Denial of Service). Wie der Name schon sagt, handelt es sich hier um verteilte/vernetzte DoS−Attacken. Der Angreifer (Client) sucht sich andere Rechner/Netzwerke, die man leicht exploiten kann, diese ersten infizierten Rechner sind dann die sogenannten Handler. Von den Handlern aus werden weitere Rechner / Netzwerke infiziert, diese infizierten Rechner werden dann Agenten genannt, d.h. bildlich dargestellt:



Später können dann die Angriffe von den Agenten...aus, ausgeführt werden.

Das war der erste Teil dieser Serie. Es geht weiter in der nächsten Ausgabe von LinuxFocus.

2005−01−11, generated by lfparser_pdf version 2.51