Hauptseminar

„Praktischer Einsatz von IDS - Network Intrusion Detection System“

von

Thomas Volkert
Matrikelnummer 29053
Studiengang Informatik

Betreuer:
Dipl.-Inf. Thorsten Strufe
Fakultät für Informatik und Automatisierung
Fachgebiet Telematik/Rechnernetz

Gliederung:

1. ***Einleitung***

   1.1 Problemstellung
   1.2 Reaktionen

2. ***Intrusion Detection System***

   2.1 Aufbau und Funktion
   2.2 Probleme im Einsatz

3. ***Network Intrusion Detection System***

   3.1 Vorteile eines NIDS
   3.2 Beispielimplementierungen

4. ***Snort in der Praxis***

   4.1 Eigenschaften von snort
   4.2 Der Preprocessor
   4.3 Snort – Rules
   4.4 Snort – Alarmausgaben
   4.5 Konfiguration

5. ***Ausblick***

6. ***Zusammenfassung***

7. ***Literatur***

Kapitel 1 - *Einleitung*

1.1 Problemstellung

Das Ausgangsproblem entsteht bei den zahlreichen Rechnern, die nach aussen über Netze zugänglich sind. Am stärksten betroffen sind Rechner, die direkt an das Internet angebunden sind. Sie stehen nahezu ständig unter Beobachtung und werden in Abhängigkeit des Interesses an ihnen kontinuierlich angegriffen. Diese Angriffe sind oftmals nicht zielorientiert und somit unberechenbar. Ein Angreifer fängt dabei stets mit diversen Scans an. Dabei versucht er zu erfahren, welche Dienste durch den Zielrechner angeboten werden (z.B. mit SYN-Scans[1]).
Weiterhin sind für spätere Zwecke die Versionen der Dienste wichtig. All das kann der Angreifer ohne größeren Aufwand schon mit einfachen Mitteln herausfinden. So zeigt das nachfolgende Beispiel eine Antwort eines Servers auf ein simples „Telnet“ auf Port 80.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>501 Method Not Implemented</TITLE>
</HEAD><BODY>
<H1>Method Not Implemented</H1>
test to /index.html not supported.<P>
Invalid method in request test<P>
<HR>
<ADDRESS>Apache/1.3.28 Server at www.prakinf.de Port 80</ADDRESS>
</BODY></HTML>
```

Dies legt die Versionsnummer „1.3.28“ des überprüften Apache-Dienstes offen.
Der Angriff geht nun weiter über existierende Exploitskripte[2] entsprechend der vorher ermittelnden Diensteversion (OpenSSH, dhcpd) oder das Ausnutzen von Hintertüren (Trojaner), um schlußendlich die Kontrolle über den Rechner zu erlangen. Die Möglichkeiten zur Gefährdung bzw. Übernahme von Rechnern sind nahezu unbegrenzt und die Wege des Angriffs sind vollkommen unvorhersehbar. Oftmals sind nur wenige Auffälligkeiten erkennbar.

1.2 Reaktionen

Natürlich wäre eine erste mögliche Reaktion, die Dienste gegen Angriffe völlig sicher zu machen. Jedoch ist das eine rein illusorische Idealvorstellung. Eine wirkliche erste Maßnahme ist, so wenig Dienste wie möglich anzubieten. Weiterhin könnten Paketfilter dazu dienen, den Zugriff auf die einzelnen Dienste einzuschränken und somit bestimmte Quellen für Angriffe auszusperren. Jedoch würden oftmals Alarmierungen bei Beginn des Angriffs reichen, um Schaden zu begrenzen oder in Ausnahmefällen sogar abzuwenden. Dafür wäre eine Beobachtungsmöglichkeit erforderlich. Es wird also eine Sensorik benötigt, die automatisch Alarm schlägt, sobald verdächtige Aktivitäten geschehen. Sie sollte eine erste Einschätzung der Gefahr geben und die beobachteten Aktivitäten an den Administrator weiterleiten. An diesen Stellen kommen IDS-Systeme ins Spiel. IDS steht hierbei für „intrusion detection system“, was schon einen ersten Hinweis vermittelt, worum es sich dabei handelt. Es ist ein System bezeichnet, welches genannte Forderungen erfüllt. Es benachrichtigt den Administrator über ermittelte Auffälligkeiten im System[3].

---

[1] Literatur [1]
[2] Exploitskripte dienen dazu, Schwachstellen im Programmcode auszunutzen, um Fehler im System zu erzeugen.
[3] Literatur [2]

Kapitel 2 - *Intrusion Detection System*

2.1 Aufbau und Funktion

Um ein IDS effektiv einzusetzen, ist ständige Pflege[4] erforderlich. Ansonsten ist das System zu veraltet, um neue Angriffsmethoden zu erkennen. Aufgrund der zugrundeliegenden Sensorik unterscheidet man hostbasierte und netzwerkbasierte IDS-Systeme. Die ersteren dienen dazu, einzelne Netzknoten zu überwachen. Hierbei greift die Sensorik nur innerhalb des Betriebssystems auf lokale Daten zu, um eine Einschätzung der lokalen Aktivitäten geben zu können. Unter netzbasiert versteht man andererseits Systeme, deren Sensorik die „vorbeifliessenden" Daten am Netzwerkinterface auswerten, um ein ganzes Netz zu überwachen.
Im Allgemeinen ist die Arbeit eines IDS in folgende 4 Stufen unterteilt: Sensorik (Informationen sammeln), Analyse (Informationen aufbereiten), Klassifizierung (vorliegenden Daten einschätzen) und Reaktion (Gegenmaßnahmen treffen). Es beginnt alles bei der Sensorik. Im Falle eines hostbasierten IDS werden hierfür bestimmte Systemaufrufe wie zum Beispiel „exec", „malloc" oder „open" überwacht. Weitere Sensorikteile prüfen kontinuierlich Einträge in den Dateien des Systemloggers. Die netzwerkbasierten IDS-Systeme überwachen widerum zum Beispiel Interfaces von Routern, Gateways oder Switchen. Unter anderem kommt hier das SNMP-Protokoll zum Einsatz, um Netzwerkgeräte zu überwachen. Bei Switchen werden eventuell vorhandene Mirrorports zur Spiegelung und damit Überwachung des gesamten Netzverkehrs genutzt.
Als nächsten Schritt sollte das IDS die ermittelten Daten analysieren. Hierbei werden Abstraktionen der Daten vorgenommen, die unter anderem komplexes Vorgehen wie etwa ein Zugriff auf Samba-Freigaben schlussfolgern lassen. Dieser Schritt ist durchaus Resourcen verschlingend. Ausgeklügelte Filter verhindern an dieser Stelle eine Überflutung mit Informationen. Danach sollte das IDS-System in der Lage sein, nach festgelegten Regeln die Gefahr einzuschätzen und abzuwägen, ob eine Reaktion sinnvoll ist. Diese Regeln können Signaturen bekannter Angriffe oder aber Heuristikverfahren sein. Letztere Verfahren machen es möglich mit neuen unbekannten Angriffen umzugehen. Eine ausgelöste Reaktion auf einen Angriff kann zum Beispiel eine Alarmierung (Mail, SNMP-Traps, Pager), eine Aufzeichnung aller Abläufe, eine Abschaltung des Netzes, eine Anpassung der Paketfilter oder sogar eine Abänderung der Netzdaten sein. Die genauen Abläufe sind abhängig von der jeweiligen Implementierung[5].

2.2 Probleme im Einsatz

Die IDS scheinen auf den ersten Blick reine Wunderwaffen zu sein. Allerdings trügt der Schein etwas. Sie haben doch einige Probleme zu bewältigen. Ein erster Hinweis steht im Kapitel 2.1. Hier wird erwähnt, dass die Filterung und die Zusammenfassung von Informationen nötig sind, um die Datenmengen der Sensorik zu bewältigen. Nur so kann man mehrfachen Alarmen derselben Erignisse entgegen wirken. Es gibt 2 Fehlerzustände beim Einsatz von IDS, die dafür sorgen, dass der Nutzen von IDS bisher umstritten blieb. Zum einen sind das „false-psotive". Damit sind Fehlalarme gemeint, die aufgrund falscher Schlußfolgerungen signalisiert wurden. Grund dafür können falsche Signaturen sein oder aber

[4] Es sollten neue Signaturen eingespielt oder aber Softwareupdates gemacht werden.
[5] Eine kleine Übersicht der Implementierungen findet sich in Kapitel 3.2 wieder.

die generische Heuristik führte zu falschen Aussagen. Zum anderen können sogenannte „false-negative“ auftreten. In diesen Fällen sind als gefährlich einzustufende Aktivitäten als harmlos eingestuft wurden. Dies kann zu schweren Schäden am System trotz aktiviertem IDS-System führen.

Kapitel 3 - *Network Intrusion Detection System*

3.1 Vorteile eines NIDS

Wie schon in Kapitel 2.1 erwähnt, dient ein NIDS-System[6] zur Überwachung eines ganzen Netzes. Ein NIDS wird möglichst abseits der Kommunikation aufgebaut. Es verhält sich durchweg passiv und dient ausschließlich der Überwachung, so dass es nicht mittelbar im Fokus des Angriffes steht. Oftmals ist es so in das Netz integriert, dass es unsichtbar für den Angreifer ist. Es befindet sich auf einem seperaten Netzknoten und beeinflusst somit nicht das Produktivsystem, welches ansonsten Leistungseinbußen verzeichnen müßte. Angeschlossen ist das System oftmals über Mirror-Kanäle, so dass es den gesamten Netzverkehr mitloggen und auswerten kann. All dies ermöglicht es, Angriffe, die über das Netz geschehen, aufzudecken und Alarm zu melden. Es dient sozusagen ein Netzknoten zur Sicherheit beliebig[7] vieler Netzknoten. Eine Übersicht einiger vorhandener NIDS-Implementierungen schließt sich im nachfolgenden Kapitel an.

3.2 Beispielimplementierungen

Snort:

- Homepage: http://www.snort.org/

Snort_inline:

- Homepage: http://snort-inline.sourceforge.net

BlackICE:

- Homepage: http://blackice.iss.net/product_pc_protection.php

Proventia Enterprise Protection :

- Homepage : http://www.iss.net/products_services/enterprise_protection

Tripwire :

- Homepage : http://www.tripwire.org/

AIDE:

- Homepage: http://www.cs.tut.fi/~rammer/aide.html

[6] Literatur [3]

[7] In Abhängigkeit des Netzverkehrs, da nur eine begrenzte Menge an Daten bewältigt werden kann.

Kapitel 4 - *Snort in der Praxis*

4.1 Eigenschaften von Snort

Das NIDS-System „sort“ ist eines der bekanntesten IDS-Systeme mit einem grossen Entwickler-Team dahinter. Das System ist unter Linux sowie unter Windows einsetzbar und steht zum freien Download im Internet bereit. Es analysiert in Echtzeit den Datenverkehr mit anderen Netzknoten und kann dabei Pakete auf dem Bildschirm ausgeben, Pakete für spätere Zwecke loggen oder aber als NIDS fungieren[8], somit Pakete nach bestimmten Eigenschaften durchsuchen. Aus den untersuchten Rohdaten kann es mit Hilfe von Vergleichssignaturen[9] bekannter Angriffsmuster entsprechend laufende Scans (wie etwa „buffer overflow“-Scans, „os finger print“-Scans) und Angriffe (wie etwa „DoS“-Angriffe) erkennen. Dies wird dem Administrator gemeldet[10], der widerum durch die erfolgte Benachrichtigung die Möglichkeit hat, entsprechende Maßnahmen zu ergreifen.

4.2 Der Preprocessor

Zusätzlich zu den Signaturen gibt es in neueren Versionen von snort die Möglichkeit das IDS-System durch „preprocessor“-Code zu erweitern. Dabei handelt es sich um Codefragmente, die nach dem Paketdekoder (gewinnt die Rohdaten) und vor dem Angriffsdetektor (schlußfolgert Angriffe aufgrund von vorliegenden Vergleichssignaturen) ausgeführt werden. Pakete werden mit ihrer Hilfe analysiert und eventuell sogar verändert bevor sie mit Hilfe der snort-Regeln (siehe nächstes Kapitel) mit den vorhandenen Signaturen verglichen werden. Für nähere Details sei hier auf die Onlinedokumentation[11] verwiesen.

4.3 Snort - Rules

Die im vorherigen Kapitel angesprochenen Signaturen sind Bestandteil der Regeln (Englisch: „rules“). Diese Regeln definieren aufgrund welcher Pakete bzw. Datenverkehrs welche Alarmzustände ausgelöst werden. Sie unterteilen sich in Regelkopf und Regeloptionen. Jede Regel beginnt mit dem obligatorischen Regelkopf. Daran schließen sich die Regeloptionen als Klammerausdruck an. Ein Beispiel ist im folgenden aufgeführt:

```
alert tcp 141.24.49.55 any -> 141.24.32.142 111 (content: „|00 01 86 a5|“; msg: „external
mountd access“;)
```

Das erste Schlüsselwort “alert” gibt die Aktion an, die ausgeführt werden soll, wenn diese Regel zutrifft. Als mögliche Aktionen stehen zur Verfügung:

- ➔ „alert“ - generiert Alarm und logt das Paket
- ➔ „log“ - logt das Paket
- ➔ „pass“ - ignoriert das Paket

[8] Dazu gibt es 3 Modi, in denen Snort arbeiten kann. In dieser Arbeit ist einzig der Modus „NIDS“ interessant.
[9] Siehe Kapitel 4.2 dazu.
[10] Siehe Kapitel 4.3 dazu.
[11] Literatur [4]

- ➔ „activate“ - generiert Alaram und aktiviert eine dynam. Regel
- ➔ „dynamic“ - bleibt inaktiv bis zur Aktivierung durch „activate“-Regel und agiert dann als Logregel

In der in Kapitel 4.5 beschriebenen Konfiguration wird nur Gebrauch der Aktion „alert“ gemacht.
Das zweite Schlüsselwort gibt das Protokoll an, dem das Paket zugehörig sein muss, damit die Regel zutrifft. Man kann hierbei zwischen TCP, UDP, ICMP und IP wählen. Weitere Protokolle sind für zukünftige Versionen von snort von den Entwicklern geplant.
Anschließend zur Protokollauswahl wird die Quelle des Paketes festgelegt. Dies wird mit der IP (kann auch eine Kombination aus IP und CIDR-Maske sein) und dem Port festgelegt. Darauffolgend wird die Richtung des Paketes festgelegt. Hier sind 3 Fälle möglich: „->“, „<-“ (Quelle und Ziel sind vertauscht), „<>“(Quelle und Ziel können gegeneinander vertauscht sein). Danach wird das Ziel festgelegt. Dies wird analog zur Quelle mit IP und Port angegeben.
Als letztes schließt sich ein Klammerausdruck an, der die näheren Paketeigenschaften[12] (hier wird zum Beispiel nach dem Inhalt `„|00 01 86 a5|“` gefiltert), eventuelle Meta-Informationen (in diesem Beispiel wird der zu loggende Text als `„external mountd access“` festgelegt) oder Trigger beschreibt. Triggeraktionen werden ausgelöst, sobald die Regel zutrifft. Alle Optionen innerhalb der Klammer werden durch „;“ voneinander getrennt. Für weitere Details zu möglichen Optionen sei auch hier auf die Onlinedokumentation verwiesen.

4.4 Snort – Alarmausgaben

Für mehr Flexibilität sorgt die Veränderlichkeit der Senken für eventuelle Alarmmeldungen. Hierbei ist es möglich zwischen 9 verschiedenen Senkentypen zu wählen:

Syslog

Hier wird das Standardsystem für Systemlogging verwendet. So dass Alarmmeldungen unter „/var/log/“ in den entsprechenden Dateien gespeichert werden. Ohne nutzerdefinierte Konfiguration schreibt snort alle Meldungen in die Datei „/var/log/snort“.

Fast

Diese Methode dient dazu, nur verkürzte Meldungen ohne detailierte Paketinformationen in eine Datei zu schreiben.

Full

Bei der Ausgabevariante „full“ verhält sich snort wie bei „syslog“. Jedoch werden zusätzlich vollständige Paketinformationen in die Ausgabedatei geschrieben. Somit stellt dies die ausführlichste und dadurch auch aufwendigste Variante dar.

Unixsocket

Hierbei werden alle Alarmmeldungen über Sockets[13] verschickt. Diese Methode gilt bisher als experimentelle Ausgabemöglichkeit.

[12] führt zur Signatur des Angriffes
[13] Literatur [5]

Tcpdump

Mit Hilfe dieser Methode erzeugt man tcpdump[14]-kompatible Ausgaben in einer Datei, die man zu einem späteren Zeitpunkt leicht weiterverarbeiten kann.

Database

Die Daten werden hier an eine SQL-Datenbank gesendet.

Csv

Hier werden die Daten im csv-Format in eine Datei geschrieben. Diese Datei kann aufgrund ihres Formates einer leichten Integration in eine Datenbank dienen.

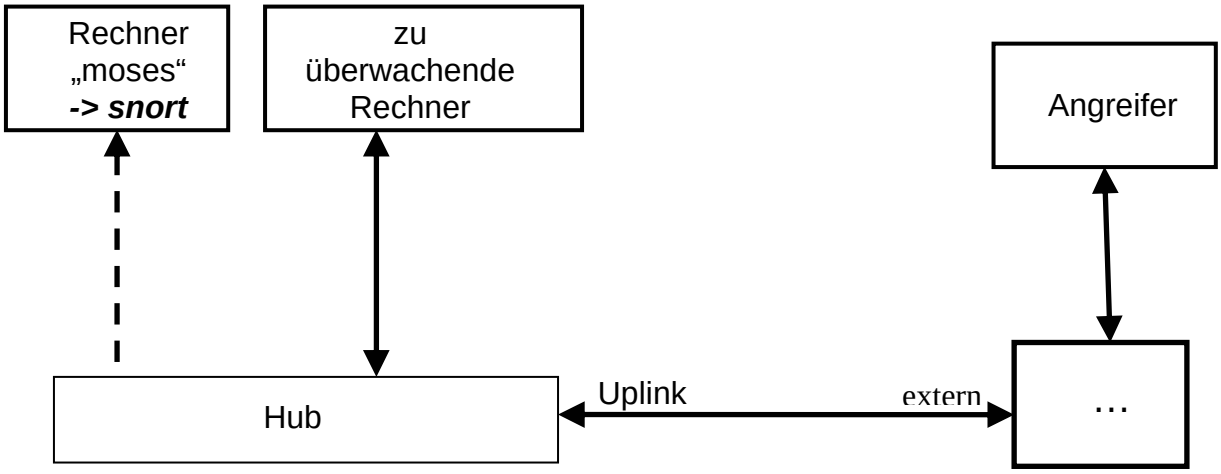
Unified

Es werden 2 Dateien mit den Daten im Binärformat gefüllt. Zum einen wird eine Datei dazu verwendet, Alarme und die damit verbundenen höheren Paketinformationen, wie zum Beispiel IP und Port, zu speichern. Auf der anderen Seite wird eine Datei mit den Details zu den einzelnen Paketen geschrieben. Dieses Verfahren soll als sehr schnelle Ausgabemethode dienen und Anwendungen die Möglichkeit geben, die Ausgaben schnell zu verarbeiten und keine größeren Logausgaben nach den wichtigen Informationen parsen zu müssen.

Null

Hier werden keine Logeinträge geschrieben. Jedoch wird Alarm ausgelöst, sobald eine Regel mit dieser Ausgabemethode zutrifft.

4.5 Konfiguration



Hardware „moses“:

- Prozessor: P3 / 600 Mhz
- Speicher: 256 MB
- Netzanbindung: 100 Mbit/s

[14] Literatur [6]

Der Rechner „moses“ ist im Labor als NIDS-Knoten eingerichtet wurden. Er bekommt über den Laborhub Netz. Als Basissystem wurde ein Gentoo Linux installiert. Neben diversen Abgrenzungen (X-Server sowie ähnliche Pakete wurden weggelassen) der installierten Softwarepakete, wurde eine spezielle Partitionierung vorgenommen. Dabei bekam der Ordner „/var“ eine seperate Partition, um Überläufe wichtiger Partitionen zu verhindern. Weiterhin wurde für „/tmp“ eine eigene Partition eingerichtet, die mit dem Attribut „noexec“ in der Datei „/etc/fstab“ eingetragen wurde. Dies geschah, um schadhaften Skripten die Ausführung im temporären Ordner zu verweigern.
Snort benötigt in der aktuellen Konfiguration etwa 10 MB Plattenplatz. Seine Konfigurationsdateien liegen, wie unter Linux üblich, unterhalb von „/etc/snort/“. Zur ersten Inbetriebnahme von „moses“ als NIDS mit Hilfe von snort war es nötig, einige rudimentäre Anpassungen in der Datei „/etc/snort/snort.conf“ vorzunehmen. Vor allem war es nötig, die Variable „HOME_NET“ zu setzen. Hiernach ist es bereits möglich, snort über sein vorhandenes Initskript[15] zu starten.

Kapitel 5 – *Ausblick*

Das aktuell installierte System ist bisher nur zum Aufspüren von Angriffen ausgerüstet. Der Schutz des NIDS-Rechners selbst fand bisher kaum Beachtung. Ein Paketfilter wäre empfehlenswert. Desweiteren wäre Software wie „portsentry“[16] ein guter Ansatz, um Scans zu erschweren und falsche Fingerprints beim Scan des NIDS-Systems zu erzwingen. Zuletzt sind ständige Updates des Systems nötig, die, wie unter Gentoo gewohnt, sehr leicht realisierbar sind.

Kapitel 6 – *Zusammenfassung*

Wie in Kapitel 5 angedeutet, kann das System in seinem aktuellen Zustand dem Aufspüren von Angriffen dienen. Allerdings ist die Konfiguration noch nicht optimal gelöst. Diverse Verbesserungen, wie etwa eine Mailbenachrichtigung (Mailflut möglich!) in Notfällen, wären wünschenswert. Die übermäßige Produktion von Logeinträgen stellt bisher noch ein nicht zu unterschätzendes Problem dar. Dies muss bei der weiteren Arbeit am System unter Kontrolle gebracht werden.
Die Erfahrungen während den Tests mit dem System zeigten, dass snort zuverlässig Auffälligkeiten bemerkt und auch auswerten kann. Die Software stellte mit einer Speicherlast von etwa 30 MB und einer Prozessorlast von etwa 15 % wenige Anforderungen an die Hardware. Jedoch stellte sich bei Lasttests heraus, dass der Rechner in der aktuellen Hardwarekonfiguration nicht zuverlässig arbeitet. Dies sollte behoben werden.

[15] Literatur [7]
[16] Literatur [8]

Kapitel 7 - *Literatur*

[1] http://www.linuxfocus.org/Deutsch/July2001/article170.shtml#170lfindex1

[2] http://www.sans.org/resources/idfaq/

[3] http://www.snort.org/docs/snort_manual/

[4] http://www.robertgraham.com/pubs/network-intrusion-detection.html

[5] http://www.manualy.sk/sock-faq/unix-socket-faq.html

[6] http://www.tcpdump.org/

[7] http://www.gentoo.de/doc/de/handbook/handbook-x86.xml?part=2&chap=5

[8] http://linux.cudeso.be/linuxdoc/portsentry.php