Studienarbeit

Host-orientiertes Netzwerk-Audit für Linux

*Konzeptionierung und Implementierung eines Host-orientierten Netzwerk-Audits auf der Basis von **LOSA***

Oliver Stecklina <ost@stecklina.net>

9. Oktober 2001

Betreuer: Dipl.-Inf. Birk Richter

Dipl.-Inf. Thomas Holz

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die zunehmende globale Vernetzung gibt dem Anwender die Möglichkeit alle Informationen, die er benötigt, aus allen Teilen der Welt in seine geschäftliche oder private Arbeitsumgebung einzubeziehen. Für ein Unternehmen ist es heutzutage unumgänglich, eine eigene Internetpräsenz zu besitzen. Hat es diese nicht, bedeutet dies meist das Auslassen eines gewaltigen Marktpotenzials. Jedoch birgt die globale Preisgabe von Informationen auch die Gefahr des Missbrauchs und des Datendiebstahls von unautorisierten Personen. Meistens wird diese Gefahr nur im Zusammenhang mit eCommerce bzw. sogenannten Online-Shops gesehen. Doch gerade diese Annahme ist nicht korrekt und führt nicht selten dazu, dass wichtige Unternehmensdaten gestohlen oder missbraucht werden. Bereits eine nur zeitweise aktivierte Online-Verbindung oder die Präsentation des Unternehmens auf einem Internet-Server öffnet fremden Personen einen ausreichend großen Zugang zu wichtigen Unternehmensdaten.

Informationstechnische (IT-) Systeme sind aus den unterschiedlichsten Gründen als inhärent unsicher zu betrachten. Dabei sind die Ursachen hierfür nicht nur in den Systemen selbst in Form von konzeptionellen Schwachstellen und Implementierungsfehlern enthalten, sondern sind oftmals auch durch Konfigurationsfehler zu einem späteren Zeitpunkt in das System eingebracht worden. Besonders kleinere Unternehmen ohne einen speziellen Systemadministrator betreiben nicht selten Internet-Server bzw. Firmennetzwerke, die ohne jeglichen Schutz dem Internet geöffnet werden. Hinzu kommt die wachsende Komplexität von IT-Systemen, die es einem Administrator nahezu unmöglich macht, Schwachstellen oder Konfigurationsfehler vor einem Angreifer zu finden. Wenn man nicht verhindern kann, dass eine fremde oder, was oftmals viel wahrscheinlicher ist, eine dem System bekannte Person unautorisiert Daten liest, vernichtet oder verändert, möchte man wenigsten wissen:

„**Wer** diese Person war, **Wie** sie es und **Wann** sie es von **Wo** aus ausgeführt hat und **Welche** Ressourcen betroffen war.“

Hierzu ist eine Protokollierung und Analyse der auf dem System ausgeführten Aktionssequenzen notwendig. In IT-Systemen wird diese elementare Sicherheitsfunktion als Audit bezeichnet.

Der Audit-Prozess in einem sicheren System beinhaltet die Aufzeichnung, die Untersuchung und die Überprüfung einzelner oder aller sicherheitsrelevanten Aktivitäten auf dem System. Die gewonnenen Informationen können zur Erkennung und zur Abschreckung von Penetrationen in ein IT-System und zur Identifikation von Missbrauch auf dem System genutzt werden [1]. Nahezu alle etablierten Betriebssysteme (BS) bieten ein sogenanntes Betriebssystem-Audit (BS-Audit). Dies ist oftmals eine sicherheitstechnische Erweiterung des BS und nicht integraler Bestandteil des Systems, d.h. das BS ist nicht vom Audit abhängig und kann ohne dieses betrieben werden.

Bei den erzeugten Audit-Daten muss man zwischen Host- (Rechner-) und Netz-basierten Informationen unterscheiden. Host-basierte Audit-Daten geben Auskunft über Aktionen, die auf dem Rechner initiiert und ausgeführt wurden. Die Funktionen zur Protokollierung sind oftmals in die Systemrufe des Betriebssystems integriert. Sie überwachen jedoch keine Aktionen, die über das Netzwerk initiiert wurden. Erst wenn ein Systemruf, bspw. `accept(...)` oder `connect(...)`, ausgeführt wird, ist die Aktion in den Audit-Daten sichtbar. Jedoch sind Aktionen, die vor dem Aufruf

dieser Systemfunktionen innerhalb des Netzwerk-Stacks[1] ausgeführt wurden mitunter nicht weniger sicherheitsrelevant. Hierzu benötigt man die Protokollierung von Netzwerkaktivitäten, die sogenannte Netz-basierte Audit-Daten erzeugen.

Um eine Protokollierung von Netzaktivitäten unabhängig vom verwendeten Protokoll zu gewährleisten, wurde in [2] das Konzept des Host-orientierten Netz-Audit (HoNA) vorgestellt. Hierbei handelt es sich um eine Audit-gestützte Netzüberwachung, die in den Protokoll-Stack des Betriebssystems integriert wurde. Das Konzept wurde bereits für das Betriebssystem Solaris 2.5 von Sun Microsystems und Microsoft Windows NT/2000 [5] implementiert. Allerdings weicht die Implementierung für Windows NT/2000 in wesentlichen Punkten vom ursprünglichen Konzept ab, so dass man diese Arbeit nicht direkt als eine Implementierung des HoNA-Konzeptes bezeichnen kann. Die hier vorgestellte Implementierung für das Betriebssystem LINUX orientiert sich am ursprünglichen Konzept, wobei folgende Punkte als integrale Bestandteile angesehen werden:

- Integration in den Betriebssystemkern,
- Protokollierung von sicherheitsrelevanten Informationen und die
- Anbindung an ein bestehendes Betriebsystem-Audit.

Integration in den Betriebssystemkern

Die Quellen des Betriebssystems LINUX sind frei zugänglich und können von jedem Nutzer im Internet unter [3] heruntergeladen und entsprechend den eigenen Anforderungen verändert werden. Dies bietet die Möglichkeit, die für das HoNA-Konzept notwendigen Funktionen in den Betriebssystemkern zu integrieren und damit einen wesentlichen Leistungs- und Sicherheitsvorteil gegenüber applikativen Netzwerkmonitoren zu erlangen. Hinzu kommt, dass mit der zunehmenden Verwendung von Protokollen zur Verschlüsselung des Netzwerkverkehrs und zur Authentifizierung der Kommunikationspartner (*Protokoll Sicherheit – engl. Protocol Security*) nur mit einem in den Netzwerk-Stack integriertem Audit eine vollständige Überwachung des Netzwerkverkehrs gewährleistet werden kann. Nutzerapplikationen können die Daten dieser Protokolle nicht auswerten, da sie hierzu die aktuell verwendeten Schlüssel benötigen. Diese Schlüssel sollten und werden bei den aktuellen Implementierungen in einem geschütztem Bereich des Betriebssystemkerns abgelegt.

Protokollierung von sicherheitsrelevanten Informationen

Neben dem Aufzeichnungsumfang, welcher den Informationsgehalt eines protokollierten Ereignisses (*Audit-Event*) beschreibt, stellt die Möglichkeit zur Selektion der Audit-Events ein wesentliches Kriterium für die Nutzbarkeit der gewonnenen Informationen dar. Sicherlich ist es ohne weiteres möglich, alle Aktionen auf einem System zu protokollieren. Die gewonnenen Informationen haben jedoch einen relativ geringen Wert, da eine Auswertung der hierbei anfallenden Datenmenge nahezu unmöglich wird. Ein Audit-System, welches bereits bei der Generierung der Audit-Daten entscheidet, welche Informationen relevant sind, bietet nicht nur einen Geschwindigkeits- und Ressourcenvorteil, sondern ermöglicht erst eine praktikable Auswertung der protokollierten Daten.

Einbindung in ein bestehendes Betriebsystem-Audit

Da die Generierung der Audit-Daten im Betriebssystemkern erfolgt, muss eine externe Komponente zur Speicherung der Audit-Daten herangezogen werden. Hierzu bietet sich die Nutzung eines bestehenden Betriebssystem-Audits an. Dieses bietet meist ein Puffer-Management für die temporäre Speicherung von Audit-Events im Kern und ein Trail-Management für die permanente Speicherung

[1] Als Netzwerk-Stack, auch Protokoll-Stack, wird die Implementierung der Netzwerkschichten-Architektur bezeichnet.

der Daten auf einem Datenträger. Für Linux existieren verschiedene Konzepte für ein Betriebssystem-Audit, jedoch sind bisher nur wenige Konzepte vollständig implementiert. Das Konzept des *LINUX Operation System Audit – LOSA* entstand im Rahmen einer Praktikumsaufgabe und wird hier als Grundlage für die HoNA-Implementierung verwendet. [6]

Die Arbeit ist inhaltlich wie folgt gegliedert: Kapitel 2 befasst sich mit der Beschreibung von Netzwerk-Architekturen und Netzwerk-Protokollen hinsichtlich ihrer sicherheitstechnischen Probleme und Schwachstellen. Anschließend werden die bisherigen Möglichkeiten zur Protokollierung von Netzwerk-Aktivitäten vorgestellt, klassifiziert und bewertet. Darüber hinaus erfolgt eine Vorstellung der vorhandenen Betriebssystem-Audit-Konzepte für LINUX. Auf das Konzept von LOSA wird genauer eingegangen, da dieses als Grundlage der hier vorgestellten Implementierung diente. Das Kapitel 3 erläutert die Umsetzung des HoNA-Konzeptes im Allgemeinen und anschließend für Linux. Bei der Umsetzung für LINUX wird insbesondere auf den möglichen Aufzeichnungsumfang und die Konfigurierbarkeit des Systems eingegangen. Im Rahmen der Beschreibung des Aufzeichnungsumfangs erfolgt die Vorstellung der protokollierten Audit-Events und des Inhaltes eines Audit-Records. Das vierte Kapitel befasst sich mit der Implementierung von HoNA für LINUX. Hierbei wird das Puffer-Management, die Integration in den Netzwerk-Stack und die Architektur zur Speicherung der Selektionsregeln vorgestellt. Im Kapitel 5 erfolgt eine Bewertung der Leistungsfähigkeit der Implementierung, dabei wird insbesondere auf mögliche Leistungsverluste, die durch das Audit-Modul erzeugt werden, eingegangen. Zusätzlich beinhaltet dieser Abschnitt einen Vergleich des HoNA-Konzeptes mit den im Abschnitt 2.2 beschriebenen Netzwerk-Monitoren bzw. Intrusion Detection Systemen (IDS). Das Kapitel 6 fasst die wesentlichen Erkenntnisse, die bei der Umsetzung des Konzeptes gewonnen wurden, zusammen und gibt einen Ausblick auf mögliche Erweiterungen und offengebliebene Schwachstellen.

Kapitel 2

Protokollierung und Analyse von Netzwerkaktivitäten

Um das Konzept von HoNA und das Grundprinzip von Netzwerkmonitoren besser verstehen zu können, erfolgt zu Beginn dieses Abschnittes eine Erläuterung der Netzwerk-Protokolle. Anschließend werden verschiedene Protokollierungsmöglichkeiten unter LINUX vorgestellt. Hierbei werden diese zunächst klassifiziert und anschließend anhand von Beispielen im Einzelnen beschrieben. Im letzten Abschnitt dieses Kapitels werden die Grenzen und die Lücken der beschriebenen Konzepte aufgezeigt. Damit sollen die Vorteile des HoNA-Konzeptes, welches im anschließenden Kapitel erläutert wird, besser verdeutlicht werden.

In der UNIX-Welt und im Internet dominieren die unter der Bezeichnung TCP/IP zusammengefassten Netzwerk-Protokolle. LINUX ist ein UNIX-ähnliches Netzwerk-Betriebssystem, weshalb das Hauptaugenmerk des Netzwerk-Stacks auf der TCP/IP-Implementierung liegt. Andere etablierte Netzwerk-Betriebssysteme, wie die Windows-Produktfamilie von Microsoft oder NetWare von Novell, verwenden teilweise eigene Protokolle zum Datenaustausch. Mit der zunehmenden Verbreitung des Internets hat sich auch bei diesen Produkten die Verwendung eines TCP/IP-Stacks durchgesetzt, so dass heutzutage im LAN/WAN-Bereich nahezu der gesamte Netzwerkverkehr mittels der TCP/IP-Protokollfamilie realisiert wird.

Die darunter liegende Kommunikation wird im Wesentlichen mittels Ethernet realisiert. Im privatem Bereich oder in kleineren Unternehmen wird darüber hinaus oftmals PPP (Point to Point Protokoll) für die Verbindung zum Internet-Provider verwendet. Eines der Entwicklungsziele von LINUX war der problemlose Einsatz in heterogenen Netzwerken oder in Netzwerk-Umgebungen, deren Kommunikation nicht auf Ethernet und TCP/IP basiert. Als Ergebnis dieser Entwicklung unterstützt LINUX heutzutage nahezu alle Netzwerk-Protokolle, die veröffentlichten bzw. etablierten IT-Systemen eingesetzt werden.

Da eine Untersuchung aller von LINUX unterstützten Kommunikationsprotokolle hinsichtlich ihrer Schwachstellen und die Integration all dieser Protokolle in HoNA zu umfangreich und für den Einsatz im LAN-Bereich unnötig sein würde, beschränkt sich diese Arbeit auf die Protokolle der TCP/IP-Familie und auf das Ethernet-Protokoll.

2.1 Audit-relevante Informationen der TCP/IP-Protokollfamilie

Die TCP/IP-Architektur ist das Ergebnis der Entwicklung eines Netzwerkes namens ARPANET. Das ARPANET war ein Forschungsnetz, das vom US-Verteidigungsministerium (*DoD – Department of Defense*) gefördert wurde. Es verband zunächst die Rechner verschiedener Universitäten über Mietleitungen. Später wurde die Verwendung von Funk- und Satellitenverbindungen hinzugefügt, dies wiederum führte zu Schwierigkeiten bei der Verwendung der vorhandenen Protokolle. Aus diesem Grund musste eine neue Architektur geschaffen werden, welche die nahtlose Verbindung von verschiedenen Netzen ermöglicht. Diese Architektur wurde später nach den zwei Hauptprotokollen TCP/IP benannt. [7]

Eines der wichtigsten Design-Ziele der TCP/IP-Protokollfamilie war die Fehlertoleranz gegenüber Ausfällen einzelner Netzwerkknoten. Solange die Endpunkte der Verbindung funktionierten, sollte die Übertragung intakt bleiben. Neben diesem Designziel war das Erreichen einer möglichst hohen

Datenübertragung ein wichtiger Schwerpunkt bei der Entwicklung der Protokolle. Dies und die geringe Verbreitung von Computern, führte zu einer wesentlichen Vernachlässigung der Entwicklung im Bereich der „*Security*“. Das Thema der Sicherheit, insbesondere der Security, wurde vor allen Dingen im letzten Jahrzehnt immer mehr zu einem Entwicklungsschwerpunkt. Im Standard der neuen Generation des Internet Protokolls IPv6 ist sie ein wesentlicher Bestandteil. Allerdings ist die Verbreitung von IPv6 derzeit noch sehr gering und die Umsetzung in einigen Punkten, insbesondere bei der Implementierung der Protokollsicherheit, noch nicht abgeschlossen.

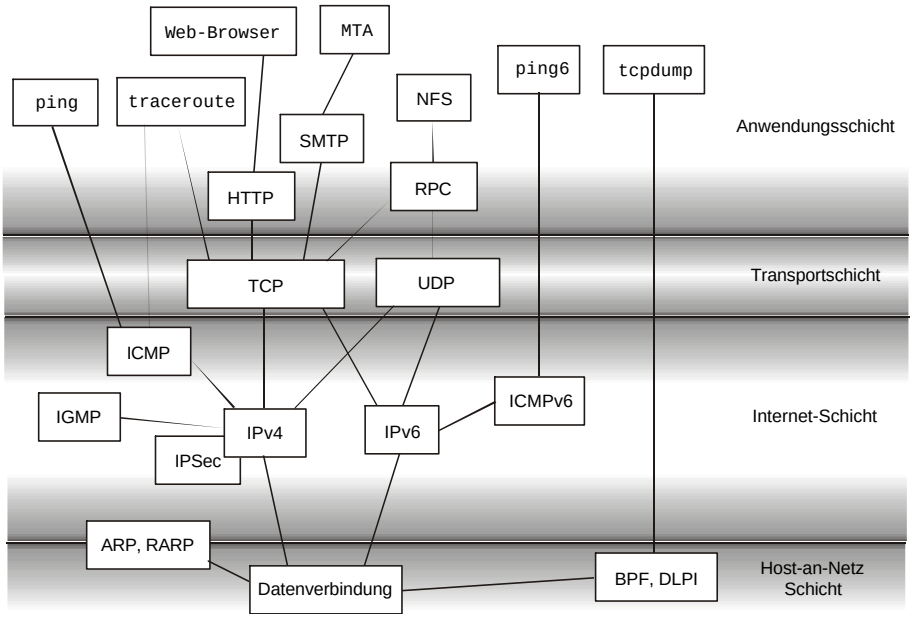

Abbildung 1 : Ein Überblick über die TCP/IP-Protokolle

Für alle in Abbildung 1 dargestellten Protokolle, Schnittstellen und Anwendungen existieren Implementierungen für das BS LINUX. Die Protokollschichten 2 (Host-an-Netz Schicht) bis 4 (Transportschicht) sind innerhalb des BS-Kerns implementiert. Schwächen innerhalb des Designs und der Implementierung dieser Protokolle sind eine direkte Bedrohung für das BS selbst und werden aus diesem Grund häufig für Angriffe auf ein IT-System ausgenutzt.

Neben den Protokollen der Schichten 1 bis 3 des TCP/IP-Referenzmodells beinhaltet die Abbildung 1 zusätzlich einige Protokolle der Anwendungsschicht. Diese Protokolle werden im Rahmen der hier vorliegenden Arbeit nicht untersucht, da deren Verarbeitung in den jeweiligen Anwendungen und nicht im BS-Kern erfolgt. Schwächen dieser Protokolle bedrohen meist nur die Anwendung selbst und nicht direkt die Funktionen des Betriebssystems. Darüber hinaus besteht innerhalb der BS-Kerns keine Notwendigkeit, diese Protokolle auszuwerten.

Die Abkürzungen BPF und DLPI stehen für *BSD Paket Filter* und *Data Link Provider Interface*. Beides sind Schnittstellen, die einen direkten Zugang zur Schicht 2 (Host-an-Netz Schicht) bereitstellen. Sie werden hauptsächlich von Programmen wie `tcpdump`, `snoop` (Solaris) oder `ethereal` zum „Loggen“ bzw. „Sniffen“ des Paketstroms verwendet.

Die *Internet Protocol Security* (*IPSec*) ist eine Erweiterung des Internet Protokolls der Version 4. In der Version 6 sind, wie bereits zu Beginn des Abschnittes beschrieben, die Protokollelemente zur Sicherung der Datenübertragung gegen Mitlesen und Manipulieren bereits im Standard enthalten. Allerdings existiert bisher keine Implementierung für LINUX, so dass für die Version 6 die Mechanismen der Protokollsicherheit nicht Bestandteil der hier durchgeführten Untersuchungen sein können. Für die Version 4 existiert die Open-Source-Implementierung FreeS/WAN [17]. Sie beinhaltet alle notwendigen Mechanismen zur Sicherung von IPv4 und wird im Rahmen der hier vorliegenden Arbeit mituntersucht.

Jede Protokollschicht muss für die Erbringung der an sie gestellten Anforderungen Kontroll- bzw. Steuerdaten zwischen den Kommunikationspartnern austauschen. Diese Kontroll- bzw. Steuerdaten werden im Kopf (*Header*) des Paketes parallel zu den eigentlichen Nutzdaten übertragen. Das Format des Headers ist für jedes Protokoll verschieden und unabhängig von anderen Protokoll-Headern. Wie in Abbildung 2 dargestellt, werden die Pakete eines Protokolls in entsprechende Pakete der darunter liegenden Schicht eingepackt (*Encapsulation*).

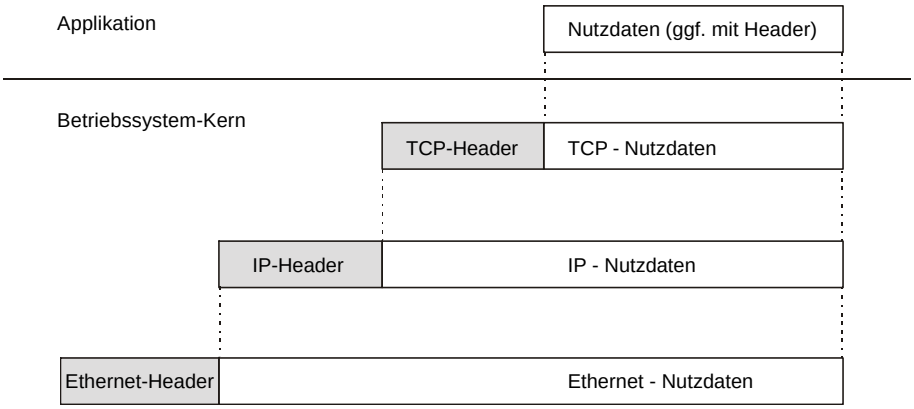

Abbildung 2 : Paket-Encapsulation

Abbildung 2 stellt die Encapsulation eines TCP-Paketes dar. Für Pakete anderer Protokolle ist die Vorgehensweise äquivalent. In der IP-Schicht wird das TCP-Paket (mit TCP-Header) als Nutzdatum betrachtet. Für die Verarbeitung des Paketes auf der IP-Schicht wird lediglich die ID des darberliegenden Protokolls benötigt. Ebenso ist die Erbringung des Dienstes der darunter liegenden Schicht für die IP-Schicht völlig transparent. Im Beispiel aus Abbildung 2 werden die Daten mittels Ethernet übertragen, ebenso gut könnte die Übertragung mittels des Point-to-Point-Protocols (PPP) erfolgen, der IP-Header bliebe dabei unverändert.

Im Folgenden werden die Protokolle der TCP/IP-Familie im Einzelnen vorgestellt. Eine genaue Beschreibung der Protokolle ist in [7] zu finden. In diesem Abschnitt wird nur insoweit auf die Protokolle eingegangen, wie es für eine sicherheitskritische Betrachtung notwendig ist.

ARP (Address Resolution Protocol)

Das *Address Resolution Protocol* dient der Zuordnung von logischen IP-Adressen zu Hardware-Adressen. Heutzutage sind die meisten Rechner mit Ethernet-Karten ausgestattet. Jede dieser Karten erhält vom Hersteller eine eindeutige 48Bit Ethernet-Adresse. Im Folgenden wird lediglich auf Ethernet-Adressen eingegangen, jedoch sind die Verfahren und Mechanismen für andere Netzwerkkarten, bspw. Token Ring ähnlich.

Zur Übertragung von Netzwerk-Paketen können keine IP-Adressen verwendet werden, da die Hardware der Sicherungsschicht (für TCP/IP Host-an-Netz Schicht) diese nicht auswerten kann. Aus diesem Grund muss für eine IP-Adresse zunächst die zugehörige Ethernet-Adresse ermittelt werden, dies erfolgt durch das Senden eines lokalen Broadcasts[2], welcher die IP-Adresse des gesuchten Rechners enthält. Jeder Rechner des Netzwerkes wertet das Paket aus, jedoch nur der Rechner mit der übermittelten IP-Adresse sendet ein ARP-Reply an den anfragenden Rechner zurück. Der anfragende Rechner kann dann die Hardware-Adresse aus dem Ethernet-Header des Paketes oder aus den Nutzdaten (*Payload*) entnehmen.

[2] Ein Broadcast ist ein spezielles Netzwerkpakete, das an alle Rechner eines Netzwerks gerichtet ist.

Das Problem beim ARP ist die „nicht Eindeutigkeit“ der Hardware-Adresse. Zwar verkauft jeder Hersteller nur Netzwerkkarten mit einer eindeutigen Ethernet-Adresse, jedoch bieten moderne Karten die Möglichkeit die Adresse zu einem späteren Zeitpunkt beliebig zu ändern. Damit ist es für einen Angreifer möglich die Adresse seiner Netzwerkkarte zu ändern und damit den Datenverkehr unbemerkt auf seinen Rechner umzuleiten. Damit die Daten zum Angreifer gelangen, muss dieser zuvor das ursprüngliche Zielsystem außer Betrieb setzen oder eine kürzere Signallaufzeit zum echten Ziel besitzen. Allerdings ist ein sogenannten ARP-Spoofing auf ein lokales Netzwerk begrenzt, da ein ARP-Reply nie über Netzwerkgrenzen hinaus übertragen wird. [3], [10]

IPv4 (Internet Protocol, Version 4)

IP ist der verbindungslose, ungesicherte Transportdienst des Internets. Der IPv4-Header enthält neben der Quell- und der Zieladresse, (hierbei handelt es sich jeweils um eine 32bit-Zahl) eine Vielzahl von zusätzlichen Feldern, die den Transport des Paketes beeinflussen.

In einer friedlichen Netzwerkumgebung besitzt jeder Rechner eine oder mehrere eindeutige IP-Adressen, die mit den übrigen Daten von den Funktionen des Betriebssystem-Kerns in den IP-Header eingetragen werden. In Betriebssystemen, die über eine Zugriffskontrolle verfügen, ist es nur für Nutzer mit speziellen Rechten möglich, die Daten des IP-Headers zu beeinflussen, so dass die Daten der Protokoll-Header teilweise als vertrauenswürdig angesehen werden können. Dies kann jedoch nur in den seltensten Fällen sichergestellt werden. Zum einen existieren Betriebssysteme, die über gar keine Zugriffskontrolle verfügen (z.B. MS-Dos) und zum anderen ist es nicht unüblich, dass in einem Netzwerk Rechner von den jeweiligen Personen verwaltet werden, die mit diesen Systemen arbeiten. Diese Personen besitzen fast immer die notwendigen Rechte, um die Netzwerkdaten zu manipulieren. Aus diesem Grund müssen alle Informationen aus dem IP-Header als unzuverlässig betrachtet werden. Dies gilt vor allem für die Quelladresse, jegliche Längen- und Offset-Angaben und insbesondere Routing-Informationen.

In der Vergangenheit und auch noch heute ist eine Zugriffskontrolle über das Netzwerk mittels der IP-Adresse stark verbreitet. Dabei wird einem fremden Rechner Zugriff auf Ressourcen gewährt, wenn dessen IP-Adresse mit Einträgen entsprechender Dateien auf dem Zielrechner übereinstimmt. Hierbei handelt es sich jedoch nur um eine Identifizierung und nicht um eine Authentifizierung, so dass diese Dienste besonders anfällig gegenüber sogenannten IP-Spoofing-Attacken sind. Wie beim ARP-Spoofing gaukelt der Angreifer dem Zielrechner eine gefälschte Quelladresse vor, um unautorisiert Zugriff auf verschiedene Ressourcen zu erlangen. Die Schwierigkeit des Angriffs liegt auch hier im Abfangen der Antwortpakete, hierzu können gefälschte Routing-Einträge verwendet werden. Dies ist zum einen beim dynamischen Routing mittels falscher Source-Routing-Informationen oder dem Manipulieren von DNS (*Domain Name Service*) Anfragen möglich [3], [10]. Letzteres wird auch als DNS-Spoofing bezeichnet [10].

Die Manipulation des IP-Headers ist außer bei Spoofing-Attacken insbesondere bei *Denial of Service* - Angriffen (DoS) hilfreich. Einer der bekanntesten Exploits[3] dieser Klasse ist `teardrop`. Hierbei wird ein Fehler in einer alten Implementierung der Funktion `ip_fragment()` ausgenutzt. IP-Pakete, die größer als die *Maximum Transport Unit* (MTU) des darunter liegenden Protokolls sind, müssen zuvor auf dem Quellrechner fragmentiert werden. Die einzelnen Fragmente werden auf dem Zielrechner mittels der Funktion `ip_glue()` wieder zusammengesetzt. Ist das Paket größer als 64 kByte wird es vom Zielrechner verworfen. Übermittelt der Quellrechner jedoch eine Paketgröße kleiner Null, versucht die Funktion `ip_glue()` dieses Paket zu verarbeiten. Dabei kopiert die Funktion eine große Menge von Daten, was zum Absturz des Rechners führt. [10]

[3] Als ein Exploit (exploit = ausbeuten) wird ein Programm bezeichnet, welches eine Schwachstelle des Systems ausnutzt, um dem Anwender unautorisiert zusätzliche Rechte innerhalb des Systems zu gewähren.

IPv6 (Internet Protocol, Version 6)

Das Internet Protokoll der Version 6 (auch als *IPnG – Internet Protocol next Generation* bezeichnet) ist ebenfalls ein verbindungsloses Transportprotokoll zur Übertragung von einzelnen Datenpaketen über ein Netzwerk. Jedoch unterscheidet sich die Umsetzung dieser Anforderung wesentlich von der Version 4. Der Basis-Header ist auf ein notwendiges Minimum reduziert. Dies soll die Bearbeitungszeit auf den Zwischensystemen reduzieren. Alle zusätzlich notwendigen Informationen müssen in zusätzliche Header (*Extention-Header*) verpackt werden. Dabei ist die Anzahl und die Reihenfolge dieser Header nicht immer zwingend vorgegeben und kann in den jeweiligen Einsatzfällen an die eigenen Anforderungen angepasst. Dies gilt insbesondere beim Einsatz des Verschlüsselungs- bzw. des Authentifizierungsheaders, deren Position entscheidend dafür ist, welche Informationen des Paketes verschlüsselt bzw. signiert werden.

Eine wesentliche Verbesserung zur Sicherung der Datenübertragung gegenüber Mitlesen oder Manipulation soll durch die Verwendung des Verschlüsselungs-Headers erreicht werden. Hiermit lässt sich die Authentizität des Absenders verifizieren und jede Manipulation auf dem Transportweg erkennen. Auf Grund der Tatsache, dass die Reihenfolge der Extention-Header frei wählbar ist, lässt sich mittels Verschlüsselung deren Inhalt und der Inhalt der Nutzdaten gegenüber potenziellen Angreifern verbergen.

Allerdings ist die Verbreitung des Internet Protokolls der Version 6 derzeit auf einzelne Rechner und Subnetze beschränkt. Demzufolge sind die Erfahrungen, die mit dem neuen Protokoll und insbesondere mit dessen Implementierung gesammelt wurden, noch sehr gering. Eine Vielzahl der möglichen Angriffe auf IT-Systeme basiert auf einer fehlerhaften Implementierung oder Konfiguration der Komponenten. Welche Angriffe auf IPv6-Rechner möglich sind, wird sich mit einer größeren Verbreitung des Protokolls zeigen.

Sollten die neuen Mechanismen zur Authentifizierung des Kommunikationspartners nicht genutzt werden, gelten bei einer Host-basierten Netzwerkzugriffskontrolle die selben Sicherheitsrisiken wie für IPv4. Sicherlich muss das Abfangen der Antwortpakete auf anderen Wegen erfolgen, jedoch ist davon auszugehen, dass dies auch bei IPv6 möglich ist. Im Rahmen der hier vorliegenden Arbeit werden nur Ereignisse protokolliert, die sich direkt aus der IPv6-Implementierung von LINUX ergeben. Eine genaue Untersuchung des Quelltextes hinsichtlich Schwachstellen oder Fehlern, die für DoS- oder ähnliche Angriffe ausgenutzt werden können, ist nicht Bestandteil dieser Arbeit.

ICMP (Internet Control Message Protocol)

Das *Internet Control Message Protocol* dient der Übertragung von Fehlermeldungen, Steuerinformationen und Statusmeldungen. Der Betrieb des Internets ist eng an dieses Protokoll gebunden. Passiert etwas Unerwartetes, wird dieses Ereignis mittels des ICMP verbreitet und muss von den betroffenen Rechnern ausgewertet werden. Darüber hinaus kann ICMP für Testzwecke verwendet werden. ICMP stellt kein eigenes Transportprotokoll dar, die Übertragung der Pakete erfolgt mittels des Internet Protokolls.

Das Protokoll basiert im Wesentlichen auf Frage- und Antwort-Paketen. Die Fragen und Antworten werden durch die ICMP-Nachrichtentypen vorgegeben. Insgesamt gibt es etwa ein Dutzend verschiedener Typen von ICMP-Nachrichten. Hinsichtlich einer sicherheitskritischen Betrachtung des Protokolls sind insbesondere die Nachrichtentypen aus Tabelle 1 relevant.

Obwohl nicht jeder ICMP-Nachrichtentyp als sicherheitskritisch anzusehen ist, sollte das Auftreten von ICMP-Nachrichten in einem lokalen Netzwerk sehr eingeschränkt sein. Dies ist vor allen Dingen immer dann der Fall, wenn mit statischen Routen und fest definierten Netzwerkstrukturen gearbeitet wird. In solchen Netzwerken sind ICMP-Nachrichten immer beachtenswert, da sie als nicht typisch bzw. als nicht notwendig anzusehen sind.

Nachrichtentyp	Beschreibung
Destination Unreachable	Das Paket kann nicht zugestellt werden. Dies deutet darauf hin, dass auf dem Zielsystem eine Störung aufgetreten ist.
Source quench	Der Quellrechner dieser ICMP-Nachricht wird mit Paketen „überflutet“. Hierbei handelt es sich um eine mögliche Reaktion eines Rechners auf ein ping-Flooding[4].
Redirect	Wird von Routern zum Umleiten von IP-Paketen gesendet, wenn diese überlastet oder nicht verfügbar sind. Dieser Nachrichtentyp kann aber auch verwendet werden, um Datenpakete über einen bestimmten Rechner zu leiten, auf welchen ein Paket-Sniffer oder eine Spoofing-Attack läuft.
Echo Request	Fragt einen Rechner, ob dieser noch erreichbar ist. Echo-Requests können darüber hinaus zum ausspionieren des Netzes verwendet werden.

Tabelle 1 : Die wichtigsten sicherheitskritischen ICMP-Nachrichtentypen

ICMPv6 (Internet Control Message Protocol, Version 6)

Der Übergang von IPv4 zu IPv6 erforderte bei ICMP einige Erweiterungen. Obwohl ein großer Teil der Funktionen von der Version 4 übernommen wurde, waren die Änderungen so gravierend, dass eine neue Versionsnummer für ICMP definiert wurde. Die wesentlichen Änderungen gegenüber ICMP der Version 4 sind:

- die Adressauflösung wurde in ICMP integriert und ersetzt damit ARP,
- zusätzliche Elemente zur dynamischen Ermittelung der MTU wurden hinzugefügt,
- Nachrichtentypen zur dynamischen Konfiguration der IP-Adresse wurden definiert,
- die Steuerung von Multicast-Gruppen wurde in ICMPv6 integriert und ersetzt IGMP der Version 4.

Demnach enthält das ICMP der Version 6 alle sicherheitskritischen Punkte der Protokolle ARP, ICMP und IGMP der Version 4. Insbesondere die Bedenken hinsichtlich des dynamischen Routings bzw. des Source-Routings müssen auch für ICMP der Version 6 beachtet werden.

IGMP (Internet Group Management Protocol)

Die übliche IP-Kommunikation findet zwischen einem Sender und einem Empfänger statt. In manchen Fällen kann es jedoch nützlich sein, ein Paket an mehrere Empfänger gleichzeitig zu schicken, z.B. im Rahmen verteilter Datenbanken, aktueller Börsennotierungen oder digitaler Konferenzen.

Zu diesem Zweck wurde das *Internet Group Management Protocol* entwickelt. Wie ICMP stellt es kein Transportprotokoll dar, sondern nutzt zum Versenden der Pakete das Internet Protokoll. Darüber hinaus ist das IGMP auch sonst dem ICMP sehr ähnlich, es werden ebenfalls verschiedene Nachrichtentypen zur Steuerung verwendet und der Protokollablauf basiert ebenfalls auf Frage- und Antwort-Paketen.

[4] Beim ping-Flooding sendet ein Host ICMP-Pakete ohne Wartezeit zwischen den einzelnen Paketen an einen Zielrechner. Diese Art ping zu verwenden, ist nur zu Testzwecken (Performancetests) üblich und ist in nahezu allen anderen Fällen eine eindeutige IT-Sicherheitsverletzung.

Hinsichtlich einer sicherheitskritischen Betrachtung besitzt das IGMP keine wirkliche Relevanz. Die übertragenen Pakete beinhalten lediglich Steuerinformationen für das Group-Management, so dass der normale IP-Verkehr nicht beeinflusst werden kann.

TCP (Transmission Control Protocol)

Für die Transportschicht haben sich im Internet zwei verschiedene Arten von Protokollen etabliert: ein verbindungsorientiertes (TCP) und ein verbindungsloses (UDP) Protokoll. Das *Transmission Control Protocol* ist ein verbindungsorientierter, gesicherter Übertragungsdienst, der das Internet Protokoll zum Übertragen der Pakete verwendet.

Eine TCP-Verbindung ist durch den Vektor (`localhost`, `localport`, `remotehost`, `remoteport`) gekennzeichnet und wird durch einen Dreiwege-Handshake (Abbildung 3) aufgebaut. `localhost` und `remotehost` werden durch die IP-Adressen der kommunizierenden Systeme beschrieben, `localport` und `remoteport` sind vergleichbar mit der Nebenstellennummer einer Telefonanlage und beschreiben den adressierten bzw. den sendenden Prozess auf dem Ziel- bzw. dem Quellsystem. Die Vergabe von Portnummer erfolgt entweder dynamisch oder zentral. Zentral vergebene Portnummern werden einmal festgelegt und gelten global im gesamten Netz. Bei der dynamischen Vergabe entscheidet die Applikation, auf welchem Port sie ihren Dienst anbietet. Da sich dieser Port ständig ändert, wird ein zusätzlicher Dienst (unter UNIX der `portmapper`) angeboten, welcher für einen gesuchten Service den aktuellen Port kennt. Im Internet hat sich eine hybride Form durchgesetzt. Ein Teil der Portnummern ist für verschiedene Dienste fest vorgegeben, sie werden als *well-unknown* Ports bezeichnet. Sie beinhalten unter anderem die Portnummer 0 bis 1024, die auch als *privileged* Ports werden, da auf ihnen nur mit speziellen Rechten ein Service angeboten werden kann. Alle übrigen Portnummern (*high-numbered* Ports) können dynamisch zugewiesen werden.

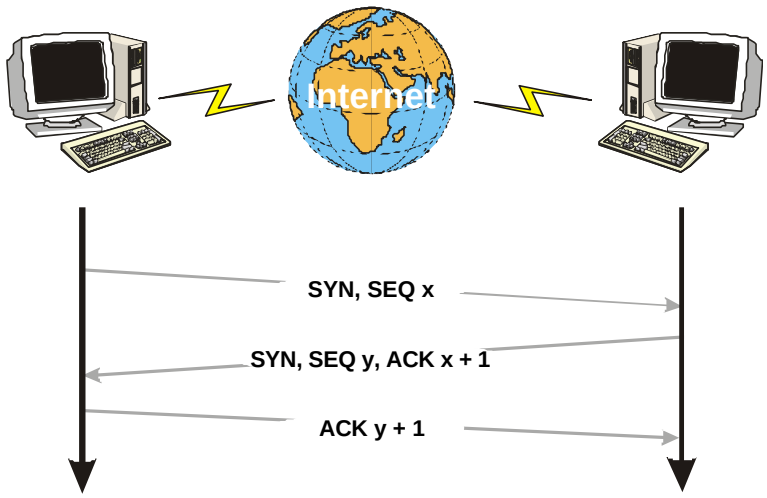

Abbildung 3 : TCP-Verbindungsaufbau

Für die Gewährleistung eines verbindungsorientierten, gesicherten Dienstes ist es notwendig, genaue Informationen über den aktuellen Zustand einer Verbindung zu speichern. Diese können durch Flags im TCP-Header oder durch Timeouts beeinflusst werden. Das Gesamtverhalten einer TCP-Verbindung lässt sich mittels des *Finite-State-Machine*-Modells beschreiben. Eine einfache Erläuterung der Finite-State-Machine ist in [15] und speziell für TCP in [7] zu finden. In jedem Zustand sind bestimmte Ereignisse zulässig. Tritt ein anderes Ereignis ein, ist dieses mit einem Fehler gleichzusetzen.

Ein Großteil der Sicherheitsrisiken wird durch die Verwendung von vertrauenswürdigen Port-Nummern und der Steuerung des TCP-State-Machine mittels Flags im TCP-Header verursacht. Eine kurze Übersicht der potentiellen Risiken und der mögliche Angriffe ist im Anhang A zu finden.

UDP (User Datagram Protocol)

Das *User Datagram Protocol* stellt den verbindungslosen, ungesicherten Dienst der Transportschicht dar. Wie beim TCP werden die Verbindungen mittels IP-Adresse und Portnummern spezifiziert, so dass ein Rechner mit einer IP-Adresse mehrere UDP-Verbindungen unterscheiden kann.

Beim UDP werden keine Sequenznummer und keine Flags verwendet, zudem ist die Angabe des Quellports optional. Demzufolge ist es für einen potenziellen Angreifer viel leichter möglich, mit einer gefälschten IP-Adresse zu kommunizieren. Authentifizierungsmechanismen für UDP-Verbindungen müssen in der Anwendung oder durch ein anderes Protokoll erbracht werden.

Wie beim TCP ist es auch mittels des UDP möglich, Zielports zu analysieren. Obwohl UDP-Pakete nicht beantwortet werden müssen, schicken viele Implementierungen ein `ICMP_PORT_UNREACH`-Fehler an den Sender zurück, wenn man ein Paket an einen geschlossenen UDP-Port sendet. Damit kann man wenigstens herausfinden, ob ein Port geschlossen ist. Ob ein Port offen ist, lässt sich damit nicht sagen, da sowohl das ICMP-, als auch UDP-Paket verloren gegangen sein kann, oder die UDP-Implementierung des Zielrechners möglicherweise keine ICMP-Fehlermeldung zurückgeschickt hat.

IPSec (Internet Protocol Security)

In der neuen Version des Internet Protokolls wurden bereits in der Design-Phase Komponenten zur Sicherung der Daten vorgesehen. Im Hinblick auf die neuen Anforderungen, die bzgl. der Sicherheit an neue Internet Protokolle gestellt werden, ist eine Erweiterung von IPv4 dringend notwendig. Da zudem eine vollständige Ablösung des „alten“ Protokolls noch einige Zeit dauern wird, ergab sich die Notwendigkeit eine Erweiterung für dieses Protokoll zu schaffen, so dass eine sichere Übertragung von Paketen auch mittels der bestehenden Architektur möglich ist.

Heutzutage ist für nahezu jedes Betriebssystem eine IPSec-Implementierung für IPv4 verfügbar. Unter LINUX hat sich insbesondere FreeS/WAN [17] etabliert, welches seit kurzem auch bei großen Distributoren wie SuSE oder Mandrake zum Lieferumfang gehört.

Die grundlegende Aufgabe einer Sicherheitserweiterung auf IP-Ebene besteht darin, jedes einzelne Datenpaket vor Verfälschung zu schützen (Authentizität | Integrität) und | oder zu verschlüsseln (Vertraulichkeit). Hierzu wendet IPSec im Wesentlichen zwei Mechanismen an: Den *Authentication Header* (AH) und den *Encapsulated Security Payload* (ESP).[3]

AH (Authentication Header)

Der AH enthält eine kryptographische Prüfsumme, die mittels einer sogenannte Hash-Funktion gebildet wird. Als Eingabewerte für die Funktion wird der Payload des Paketes und Teile des Headers verwendet. Die einzusetzende Hash-Funktion ist hierbei nicht genauer spezifiziert, als Minimum wird SHA vorausgesetzt. Die Hash-Funktion selbst wird mit einem symetrischen Schlüssel parametrisiert und dann bspw. HMAC-SHA genannt.

Mittels des AH können sowohl die Nutzdaten als auch die einbezogenen Teile des Headers vor Verfälschung während des Transports gesichert werden, so dass ein Großteil der IP-Spoofing Angriffe, wie sie zuvor beschrieben wurden, unmöglich werden sollten. Einige Teile des Headers müssen ausgelassen werden, da sie während des Transports von Zwischenstationen geändert werden müssen und damit die Prüfsumme zerstören würden, bspw. das *time to live* –Feld.

ESP (Encapsulated Security Payload)

ESP dient der Verschlüsselung der Nutzdaten eines IP-Paketes. Als Verschlüsselungsverfahren wird mindestens DES-CBC gefordert. Es existieren auch Implementierungen in denen andere, kryptographisch stärkere Verfahren wie 3DES oder IDEA angewendet werden. Alle verwendeten Verfahren beruhen auf symmetrischen Schlüsseln. Die Verwendung von asymmetrischen Verfahren wie RSA ist auf Grund der schlechten Performance der Algorithmen nicht möglich.

Verschlüsselt wird immer der Payload des Paketes, wobei dieser vom verwendeten Modus abhängig ist. Als Modi sind der *Transport Mode* und der *Tunnel Mode* vorgesehen. Beim *Transport Mode* werden lediglich die Nutzdaten des IP-Paketes verschlüsselt. Die Adressen des ursprünglichen Headers bleiben weiterhin sichtbar. Im Gegensatz dazu wird beim *Tunnel Mode* das gesamte Datenpaket verschlüsselt und mit einem neuen Header versehen. Die Quell- und die Ziel-Adresse des neuen Paketes sind die Adressen der Tunnelendpunkte. Sie müssen nicht identisch mit der ursprünglichen Quell- und Zieladresse des Paketes sein.

Security Associations

Sowohl der AH- als auch der ESP-Header beinhaltet keine Informationen über das anzuwendende Verschlüsselungsverfahren bzw. welcher Schlüssel momentan zu verwenden ist. Zur Verarbeitung des Paketes sind diese Informationen jedoch zwingend erforderlich. Sie werden durch den SPI (*Security Parameter Index*), der innerhalb der Header vermerkt ist, adressiert. Die eigentliche Datenstruktur wird lokal auf den Tunnelendpunkten, die im Folgenden auch als IPSec-Gateways bezeichnet werden, gehalten und als *Security Association* (SA) bezeichnet.

Eine SA gilt immer unidirektional, ist auf den Empfänger bezogen und enthält die IP-Adressen der Verbindungsendpunkte, den SPI, das Protokoll (AH oder ESP) und den Modus. Für eine typische bidirektionale Verbindung müssen demnach immer zwei SAs existieren. Werden die verwendeten Verfahren, bspw. AH und ESP kombiniert, muss dies in separaten SAs gespeichert werden. Somit ist für eine bidirektionale, verschlüsselte Kommunikation mitunter ein „SA-Bündel“ notwendig.

Eine ausführliche Beschreibung der Verschlüsselungsverfahren, Header-Formate und Protokollabläufe ist in [3] zufinden. Für die Beschreibung der Mechanismen zur Protokollierung von IPSec sind diese nicht zwingend erforderlich und werden aus diesem Grund hier ausgelassen.

Im Wesentlichen lässt sich zusammenfassen, dass IPSec zusätzliche Header-Formate und Datenstrukturen zur Realisierung von sicheren Internetverbindungen verwendet. All diese Erweiterungen sollen die Schwächen von IP bezüglich der Sicherheit ausgleichen oder zumindest auf ein Mindestmaß reduzieren, so dass Attacken, wie sie in der Vergangenheit möglich waren, verhindert werden können. Dies ist auch zum Großteil gelungen, jedoch hat man sich diesen Vorteil mit einem großen Maß an Performanceeinbußen erkaufen müssen und ist damit um so mehr offen gegenüber DoS-Attacken. Insbesondere DoS spielt bei IPSec eine große Rolle. Durch die Verwendung von ineinander verschachtelten Paketen ist ein IPSec-Gateway gezwungen einen großen Aufwand an Rechenleistung „vorzustrecken“, bis er letztendlich in der Lage ist, die Korrektheit des Paketes zu überprüfen.

2.2 Aktuelle Protokollierungs- und Analysemöglichkeiten unter Linux

Nachdem im vorangegangenen Abschnitt die jeweiligen Schwachstellen der einzelnen Netzwerkprotokolle vorgestellten wurden, befasst sich dieser Abschnitt mit deren Protokollierungs- und Analysemöglichkeiten. Zunächst erfolgt eine Klassifizierung von Netzwerkmonitoren, dabei wird zu jeder Klasse mindestens ein Beispiel angeführt, welches im anschließenden Abschnitt erläutert wird. Nach der Beschreibung der einzelnen Netzwerkmonitore wird kurz auf deren Schwächen eingegangen.

2.2.1 Klassifizierung von Protokollierungs- und Analysewerkzeugen

Die Erbringung von Diensten mit Hilfe von Computernetzwerken erfolgt im Wesentlichen nach dem Client-Server-Prinzip. Hierzu tauschen Client und Server neben den eigentlichen Nutzdaten, Protokoll-spezifische Informationen aus, die die Grundlage des Netzwerk-Audits bilden. Die Aufzeichnung der Daten erfolgt durch Netzwerkmonitore parallel zur Inanspruchnahme des Dienstes. Dabei können Netzwerkmonitore bzgl. der softwaretechnischen Einbettung in der gegebenen Kommunikationsarchitektur und ihrer Positionierung innerhalb der zu überwachenden Netzwerkdomäne klassifiziert werden [2].

Die softwaretechnische Einbettung bestimmt wesentlich den Inhalt der protokollierten Daten. Wie in Abbildung 4 dargestellt, unterscheidet man zwischen einer applikativen und einer BS-Kern-integrierten Implementierung. Während man bei der Integration in den BS-Kern, die vorhandenen Analysefunktionen verwenden und Zustände von Protokollen (bspw. TCP, IPSec) mit einbeziehen kann, muss dies beim applikativen Ansatz für jedes Paket separat durch die Applikation erfolgen. Insbesondere Protokolle, bei denen der Inhalt der Pakete (IPv6, IPSec) verschlüsselt wird, können nur mittels eines BS-Kern-integrierten Ansatzes effizient protokollierten werden.

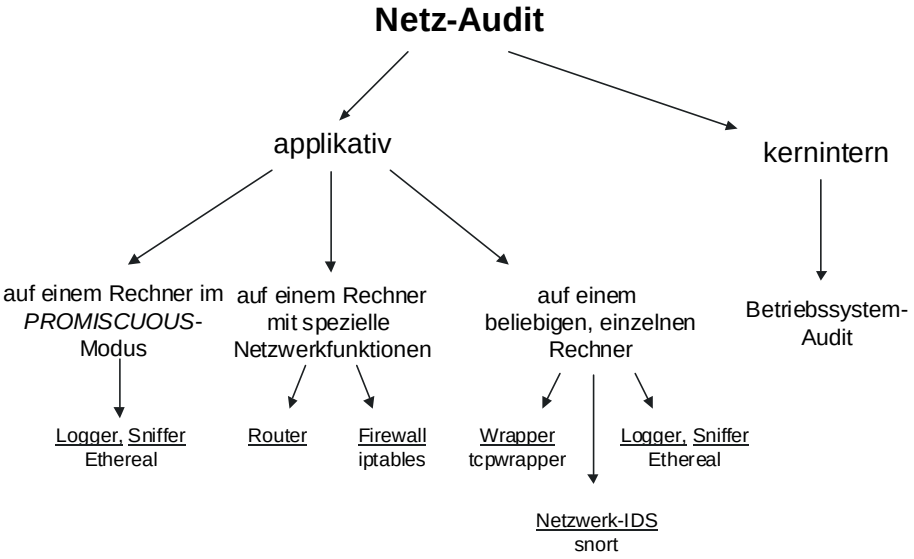

Abbildung 4 : Netzwerk-Monitore im Überblick

Die Positionierung des Netzwerk-Monitors innerhalb des Netzwerkes ist entscheidend für dessen Sichtfeld. Mit dem zunehmenden Einsatz der Switch-Technologie wird diese jedoch immer mehr eingeschränkt, so dass ein Rechner, selbst wenn er im *Promiscuous*-Modus[5] betrieben wird, nur noch die Datenpakete sieht, die für ihn bestimmt sind. Dieses Problem kann mittels eines Switches umgangen werden, der die Einrichtung eines sogenannten Span-Ports unterstützt. Dieser leitet eine Kopie aller eingehenden Pakete an den Span-Port weiter. Ein Netzwerk-Monitor hinter diesem Port kann somit den gesamten Datenverkehr des Switches überwachen. Darüber hinaus haben Rechner mit speziellen Funktionen innerhalb des Netzwerkes wie bspw. Router oder Firewalls Zugriff auf alle Pakete, die zwischen den Netzwerken, die sie miteinander verbinden, ausgetauscht werden.

[5] Per Voreinstellung antwortet ein Rechner nur auf Pakete, die an ihn gerichtet sind. Es ist allerdings möglich, einen Rechner, besser gesagt dessen Netzwerk-Interface, in einen spezielle Modus zu setzen, so dass er alle Pakete verarbeitet, die für ihn sichtbar sind. Dieser Modus wird Promiscuous-Modus genannt.

2.2.1.1 Logger, Sniffer und Capturer

Für die Protokollierung des Paketstroms auf den Schichten zwei bis vier existieren sogenannte *Logger*, *Sniffer* bzw. *Capturer*. Hierbei handelt es sich um Applikationen, die die Pakete direkt aus dem Protokoll-Stack des BS abgreifen. Mit Hilfe von Filtermechanismen sind diese Programme in der Lage den Paketstrom nach bestimmten Protokollen und manchmal auch Paketinhalten zu durchsuchen. Je nach Implementierung können die gewonnenen Informationen verschieden dargestellt bzw. gespeichert werden.

Durch die Verwendung des *Promiscuous*-Modus der Netzwerkkarte sind diese Programme meist in der Lage, den Paketstrom eines ganzen Netzsegmentes zu protokollieren. Hierzu muss das System, auf dem ein Sniffer betrieben wird, in der Lage sein, auf alle Pakete des Segmentes zugreifen zu können. Bei der Verwendung von Bus- (BNC) oder Hub-Technologien ist dies bereits durch die Architektur gegeben. Wird für die Koppelung der einzelnen Netzknoten ein Switch eingesetzt, muss der Sniffer an einem Span-Port des Switches betrieben werden, um alle Rechner überwachen zu können.

2.2.1.2 Firewall

Nahezu alle Firewalls sind optional in der Lage die Protokolldaten des von ihm verarbeiteten Netzwerkverkehrs aufzuzeichnen. Je nach Art des Firewalls (Proxy oder Paketfilter) unterscheidet sich die Qualität und die Menge der protokollierten Daten. Im Gegensatz zu Sniffern, die nur der Protokollierung dienen, sind Firewalls in der Lage den Aufbau einer Verbindung zu unterbrechen bzw. einzelnen Rechnern den Zugriff auf das von ihnen geschützte Netzwerk zu verbieten.

Proxy, Application Level Gateway

Proxies, auch als *Application Level Gateways* bezeichnet, befinden sich meist zwischen Client und Server und laufen als Applikationen im Nutzerspeicher. Bei der Verwendung von Proxies verbindet sich der Client nicht mehr direkt mit dem Server, sondern nur mit dem Proxy und sagt diesem, mit welchem Server er gern kommunizieren möchte. Der Proxy baut daraufhin selbst die Verbindung zu diesem Server auf und leitet die Daten über sich zum Client weiter. Bevor der Proxy die Daten weiterleitet, werden diese meist einer Content-Kontrolle unterzogen. Hierbei werden die Inhalte der Pakete auf der Anwendungsschicht analysiert. Damit lassen sich bspw. einzelne URLs (*Uniform Resource Locator*) sperren, Virentests durchführen oder die Ausführung von Active-X-Component verbieten. [14]

Paketfilter

Paketfilter analysieren normalerweise den Paketstrom auf den Schichten drei und vier. Die Anwendungsschicht wird von ihnen in der Regel nicht untersucht. Einige Paketfilter sind zu dem in der Lage Protokolle (meist Ethernet) der Host-an-Netz Schicht mit einzubeziehen. Moderne Filter führen auf der Transportschicht für das TCP eine State-Full-Inspektion durch. Hierbei wird der tatsächliche Status der Verbindung mit einbezogen.

Für eine effiziente und wirkungsvolle Implementierung eines Paket-Filters ist es sinnvoll diese in den BS-Kern zu integrieren. Dies erspart wesentliche Performanceverluste beim Kopieren der Pakete in den Nutzerspeicher und wieder zurück in den Kernel-Speicher und ermöglicht zusätzlich eine Härtung der Stack-Implementierung gegenüber Angriffen gegen das BS selbst.

2.2.1.3 Wrapper

Wrapper sind Mechanismen, die Netzwerkdiensten, die keine oder eine zu geringe Zugriffskontrolle besitzen, die Möglichkeit geben, diese wesentlich zu erweitern. Hierbei werden die übertragenen Protokolldaten auf den Schichten zwei und drei mit den eigenen Autorisationsdaten verglichen.

Entsprechend des Resultats des Vergleichs wird der Zugriff gewährt oder verweigert. Darüber hinaus verfügen *Wrapper* über Möglichkeiten zur Protokollierung der untersuchten Pakete.

Die Implementierung bzw. die Integration des Wrappers ist abhängig vom Diensterbringer (Server). Entweder werden sie direkt in diesen integriert (z.B. Samba, `ssh`) oder befinden sich in einem sogenannten Super-Server (z.B. `inetd`), der verschiedene Dienste verwaltet.

2.2.1.4 *Intrusion Detection System* (IDS)

Wie aus dem Namen hervorgeht, befasst sich ein IDS mit der Erkennung von IT-Sicherheitsverletzungen. Als IT-Sicherheitsverletzung werden sämtliche Aktionen bezeichnet, die der Sicherheitspolitik des Systems zuwiderlaufen. Für die Erkennung werden vorrangig Werkzeuge eingesetzt, die eine Analyse oder Überwachung des Systems oder des Netzwerkes mit Hilfe von BS-Audit-Daten bzw. Netzwerk-Loggdaten durchführen. [4]

Grundlegend unterscheidet man zwei Analysekonzepte bei der Intrusion Detection:

- die Erkennung von Anomalien, die auf IT-Sicherheitsverletzungen hindeuten und
- das Aufspüren bekannter Angriffe mit Hilfe der Signaturanalyse.

In der Praxis haben sich bisher nur Systeme etabliert, die auf der Signaturanalyse beruhen. Die Anomalieerkennung ist zur Zeit noch ein wesentlicher Bestandteil der Forschung und in den wenigsten kommerziellen IDS integriert.

Anomalie-Erkennung

Der Grundgedanke bei der Anomalie-Erkennung ist das Vorhandensein von festen Verhaltensmustern auf etablierten Systemen. Tritt ein davon abweichendes anormales Verhalten auf, kann dies auf eine mögliche Sicherheitsverletzung hindeuten und entsprechend signalisiert werden. Für die Anomalie-Erkennung ist es demzufolge notwendig, das gesamte Verhalten eines Nutzers bzw. eines Systems effizient beschreiben und mit dem auftretenden Verhalten vergleichen zu können.

Ist dies möglich, könnte theoretisch eine Sicherheitsverletzung erkannt werden, ohne genaue Kenntnisse über diese selbst zu haben. Dies würde ein extrem flexibles und wartungsarmes IDS ermöglichen. Jedoch führt jedes abweichende Kommando eines Nutzers, z.B. wenn dieser einen neuen Browser ausprobiert, mitunter zu einem Alarm. Die Menge der Fehlalarme wäre in einem neuen oder in einem sich stetig verändernden System dermaßen groß, dass der Administrator leicht dazu geneigt wäre Alarme zu ignorieren oder die Empfindlichkeit des Systems zu verringern. Letzteres würde es einem Angreifer ermöglichen mit nur kleinen Änderungen seines Verhaltens einen Angriff unentdeckt durchführen zu können.

Signaturanalyse

Die Signaturanalyse basiert auf der Grundlage bekannter und hypothetischer Angriffsszenarien. Eine Voraussetzung für die Erkennung von Angriffen mit Hilfe der Signaturanalyse ist die Kenntnis über signifikante Verhaltensmuster der einzelnen Angriffe. Dazu gehört z.B. das Initialisieren von bestimmten Variablen in der jeweiligen Systemumgebung.

Der Vorteil der Signaturanalyse liegt in der Erkennungsgenauigkeit. Erst wenn das aufgetretene Verhalten mit einer bekannten Signatur übereinstimmt, kann ein Alarm ausgelöst werden. An dieser Stelle kann man davon ausgehen, dass wirklich eine IT-Sicherheitsverletzung aufgetreten ist. Der Nachteil des Konzeptes liegt in der Abhängigkeit von den Signaturen. Es können nur Angriffe erkannt werden, die mittels einer Signatur beschrieben und in das IDS integriert sind. Neue Angriffe können in

der Regel nicht erkannt werden. Diese müssen erst beschrieben und dem IDS bekannt gegeben werden. Dies führt wiederum zu einem ständigen Wartungsbedarf des IDS.

2.2.1.5 Betriebssystem Audit

Das Betriebsystem-Audit ist meist eine grundlegende Voraussetzung für ein Host-basiertes IDS und stellt im Wesentlichen eine sicherheitstechnische Erweiterung des BS dar. Jedoch wurde es fast nie zu diesem Zweck integriert. Der eigentliche Grund liegt wohl eher in denen vom *Department of Defense* (DoD) *Computer Security Center* aufgestellten *„Security Requirements for Automatic Data Processing (ADP) Systems“*. Darin werden die ADP-Systeme in vier Stufen unterteilt, wobei die Stufe C2 ein Auditing zwingend voraussetzt. Genau diese Stufe C2 gilt in vielen Unternehmen oder bei Behörden als Mindestanforderung für den Einsatz von Software-Produkten. Nahezu jedes etablierte BS verfügt über ein Betriebsystem-Audit-Konzept. Bei Windows NT ist dies z.B. der Ereignismonitor, während es bei Solaris es durch das *Basic Security Modul* (BSM) realisiert wird. Wie gut diese Systeme implementiert sind und wie groß der Aufzeichnungsumfang sein muss, ist in den Anforderungen des DoD nur sehr ungenau beschrieben, so dass sich die einzelnen Konzepte stark unterscheiden.

All diese Konzepte habe eine Gemeinsamkeit: Sie protokollieren fast ausschließlich nur Aktionen, die infolge der Ausführung eines Systemrufes initiiert wurden. Demnach werden Aktionen des Netzwerk-Stacks nur protokolliert, wenn sie durch die Ausführung eines Systemrufs initiiert werden, z.B. `connect()` oder die Fortführung einer solchen Funktion zur Folge haben, z.B. `accept()`. Aktionen, die innerhalb des Stacks infolge eines empfangenen Netzwerkpaketes ohne die Bindung an einen Prozess ausgeführt werden, sind von der Protokollierung ausgeschlossen.

2.2.2 Praxisbeispiele für Linux

Während bei den etablierten Betriebssystemen wie Windows NT, Solaris oder AIX zunächst die Hersteller selbst und später Drittanbieter den Großteil der verfügbaren Software bereitstellten, wurde die Entwicklung von Linux und dessen Software von Anfang an von einer großen Anzahl freiwilliger Entwickler vorangetrieben. In letzter Zeit erkennen die Hersteller großer Software-Produkte die Bedeutung von Linux und beginnen mit der Portierung ihrer Produktpalette. Dies gilt auch für Produkte, die sich mit der Sicherheit von Systemen bzw. Netzwerken beschäftigen. Obwohl Linux zunächst nur auf dem Markt der Server-Betriebssysteme einen nicht unwesentlichen Marktanteil besaß, liegt dieser heute bei 30 Prozent und ist noch ansteigend. Das zögerliche Engagement der großen Softwarehersteller und die Größe des Markanteils von LINUX hat dazu geführt, dass die frei verfügbaren Programme den professionellen Produkten nicht immer unterlegen waren. Insbesondere was die Stabilität und Response-Zeit bei auftretenden Sicherheitslöchern betrifft, sind diese Programme manchmal weitaus besser und werden nicht selten bevorzugt eingesetzt.

Die in diesem Abschnitt vorgestellten Programme sind alle frei verfügbar und kostenlos. Darüber hinaus gibt es heute für LINUX eine nicht unwesentliche Anzahl an professionellen Software-Produkten, deren Leistungsfähigkeit den im Folgenden beschriebenen Programmen, weit überlegen sein kann. Im Wesentlichen gehören diese Produkte zur Gruppe der Netzwerk Intrusion Detection Systeme. In der hier vorliegenden Analyse soll das Programm `snort` als Beispiel herhalten. Da jedoch von diesen Produkten nur schwer Demo-Programme oder Evaluationskopien zu erhalten sind, beschränkt sich diese Arbeit auf frei verfügbare und kostenlose Produkte.

2.2.2.1 Netzwerk-Sniffer `ethereal`

Das Programm `ethereal` ist eines von vielen Netzwerk-Monitoren für LINUX, welches auf der `libpcap`-Bibliothek basiert. Das Programm sowie die Bibliothek unterliegen der *GNU General Public License* (GPL) und sind somit frei verfügbar. Die Homepage des Projektes ist unter [18] zu finden.

Die `libpcap`-Bibliothek bietet ein portables Framework für die Überwachung des Netzwerkpaketstroms. Die angebotenen Funktionen reichen vom Sammeln statistischer Informationen bis hin zum Debugging von Netzwerkprotokollen. Bedingt durch die verschiedenen Implementierungen der einzelnen BS, bietet jedes einen anderen Zugang zum reinen Paketstrom. Mittels der `libpcap`-Bibliothek wird ein System-unabhängiges API angeboten.

Im Gegensatz zu den schon etwas älteren Programmen wie bspw. `tcpdump` verfügt `ethereal` über eine graphische Benutzeroberfläche, welche die Bedienung und die Auswertung der enthaltenen Informationen wesentlich vereinfacht. Zusätzlich können die Daten ausgedruckt oder in verschiedenen Dateiformaten abgespeichert werden.

Zum Funktionsumfang des Programms gehört die Unterstützung bzw. Visualisierung der verschiedensten Netzwerkprotokolle der Schichten eins bis vier. So ist es unter anderem möglich, die Header der Anwendungsprotokolle RPC (*Remote Procedure Call*) oder SMB (*Server Message Blocks*) anzuzeigen und auszuwerten. Darüber hinaus kann der Paketstrom mit frei definierbaren Filtern selektiert oder einzelne TCP-Sessions verfolgt und gesondert dargestellt werden.

2.2.2.2 Paketfilter mit Netfilter / `iptables`

Seit der Version 2.0 bietet LINUX offiziell einen Kern-integrierten Firewall, der in den folgenden Versionen 2.2 und 2.4 stetig erweitert wurde. Den derzeitigen Entwicklungsstand stellt die Version 2.4 mit Netfilter und `iptables` dar. Bei der Netfilter-Architektur handelt es sich um ein in den LINUX Kernel integrierten Paketfilter. Für die Administration des Filters wird das Programm `iptables` verwendet.

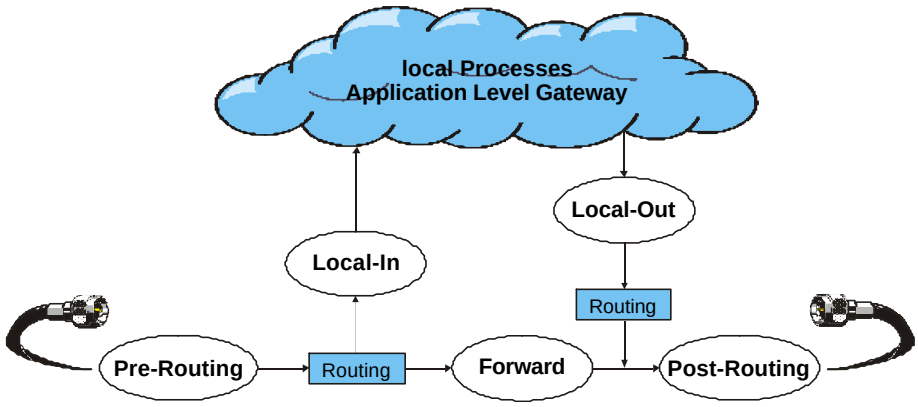

Abbildung 5 : IP-Reisediagramm mit Netfilter

Netfilter besteht im Wesentlichen aus einer Reihe von Hooks innerhalb des Netzwerk-Stacks. Zur Zeit können die Pakete der Protokolle IPv4, IPv6 und DECnet gefiltert werden. Ein idealisiertes IP-Reisediagramm ist in Abbildung 5 dargestellt.

Reise eines IP-Paketes

Alle empfangenen Pakete passieren zunächst den `Pre-Routing` Hook. Hier erfolgt die *Destination Network Address Translation* (*Destination NAT*), dabei wird die Zieladresse des Paketes durch die wirkliche IP-Adresse des Ziels ersetzt. Anschließend wird der Routing-Code des Netzwerk-Stacks aufgerufen. An dieser Stelle wird entschieden, wie das Paket weiterzuverarbeiten ist. Ist das Paket an den Host selbst gerichtet, wird es dem `Local-In` Hook übergeben. Soll das Paket weitergeleitet werden, muss es dem `Forward` Hook übergeben werden.

Im `Local-In` Hook befinden sich die Filter und Logging-Regeln des Firewalls für empfangende Pakete. Entscheiden diese Regeln, dass das Paket nicht weiterverarbeitet werden darf, wird es an dieser Stelle verworfen.

Der `Forward` Hook enthält die Regeln für weiterzuleitende Pakete. Jedes Paket, welches den Firewall passiert (dies werden die meisten Pakete sein), muss durch den `Forward` Hook geleitet werden[6]. Damit entscheiden die Regeln des `Forward` Hooks über die Behandlung von Paketen eines ganzen Netzwerkes. Anschließend werden die Pakete dem `Post-Routing` Hook übergeben.

Pakete, die lokal auf dem Rechner generiert wurden, passieren zunächst den `Local-Out` Hook. Wie in Abbildung 5 dargestellt, wird anschließend der Routing-Code aufgerufen. Dies ist nicht ganz korrekt, da das Paket vor dem `Local-Out` Hook bereits einmal geroutet wurde. Da es mittels Netfilter-Regeln möglich ist, die Quelladresse des Paketes zu verändern, muss das Paket anschließend noch ein weiteres Mal geroutet werden. Das Austauschen der Quelladresse wird auch als *Source NAT* bezeichnet[7].

Abschließend passiert ein Paket immer den `Post-Routing` Hook. Für lokal generierte Pakete werden hier in der Regel keine Operationen ausgeführt. Für Pakete, die über den `Forward` Hook weitergeleitet wurden, erfolgt im `Post-Routing` Hook Source NAT, für lokal generierte Pakete erfolgte dies bereits im `Local-Out` Hook.

Logging-Möglichkeit mittels Netfilter

Jede Netfilter-Regeln besteht aus der Hook-Angabe, in dem sie ausgeführt werden soll, einem Pattern und der Zielangabe. Das Ziel entscheidet, wie mit dem Paket weiterverfahren wird. Das Paket kann still und heimlich verworfen (`DROP`) werden oder es kann vor dem Verwerfen dem Sender eine ICMP-Meldung, dass das Ziel nicht erreichbar ist, zurückgeschickt werden (`REJECT`). Außerdem kann das Paket mit dem Ziel `ACCEPT` akzeptiert werden. All diese Ziele werden ohne eine Meldung aufgerufen und ausgeführt.

Mittels des Ziels `QUEUE` kann das Paket einem Nutzerprozess übergeben werden. Dieser könnte das Paket, ähnlich wie `ethereal` analysieren und entsprechende Protokolldaten erstellen. Das Paket wird mittels der Regel `QUEUE` aus dem Stack entfernt und muss mittels einer `RETURN`-Regel in einen Hook eingefügt werden. Dieses Ziel dient eigentlich der leichten Integration von Filter-Modulen, die nicht in den Kernel eingebunden werden sollen.

Eine wesentlich leichtere Möglichkeit zum Protokollieren der Pakete wird mittels des Ziels `LOG` angeboten. Hiermit können direkt Kernel-Meldungen an den Syslog-Dämon übergeben werden. Beim Zutreffen einer solchen `Log`-Regel wird der IP-Header in einer lesbaren Form ausgegeben. Um die einzelnen Regeln in den Log-Dateien unterscheiden zu können, kann jeder Regel eine Beschreibung mitgegeben und das Log-Level des Syslog-Dämon (`debug`, `info`, `notice`, `warning`, `err`, `crit`, `alert` oder `emerg`) vorgegeben werden. Anders als bei den Zielen `QUEUE`, `DROP`, `REJECT` und `ACCEPT` werden die Pakete, wenn sie auf eine Log-Regeln treffen, anschließend durch den Hook

---

[6] Es sei denn, auf dem Firewall wird zusätzlich ein Application Level Gateway eingerichtet. In diesem Fall passieren nur die wenigsten Pakete den `Forward` Hook. Sie werden direkt an den Proxy gesendet und anschließend von ihm weitergeleitet werden. Da der Proxy auf dem Firewall ein Nutzerprozess ist, werden die Pakete durch den Local-In und den Local-Out Hook geleitet.

[7] Es gibt einen Spezialfall von Source NAT, der Masquerading genannt wird. Es sollte nur für dynamisch zugeordnete IP-Adressen verwendet werden und ersetzt automatisch die Quelladresse des Paketes durch die Adresse des Interfaces, an der das Paket ausgeht.

weiterverarbeitet. So kann das Übereinstimmen einer Filter-Regeln leicht durch das Kopieren dieser und das Ersetzen des Ziels protokolliert werden.

2.2.2.3 Netzwerk-IDS `snort`

Bei `snort` [16] handelt es sich um ein sogenanntes Netzwerk – Intrusion Detection System, welches in der Lage ist, den Netzwerkverkehr online zu analysieren und aufzuzeichnen. Das Programm kann sowohl zur Protokollierung des Paketstroms als auch zur Suche nach bestimmten Inhalten in den Paketen des Netzwerkverkehrs eingesetzt werden. Dabei können eine Vielzahl von Angriffen und Scans, wie Buffer Overflows, Stealth Scans, CGI-Attacken, SMB-Sondierung oder bspw. aktives Fingerprinting erkannt werden. Die gewonnenen Informationen können dem Syslog-Dämon übergeben, in eine Datei geschrieben oder über *smbclient* dem Administrator mitgeteilt werden. [8]

Funktionsweise

`snort` basiert auf der Signaturanalyse. Die Signaturen werden mittels Regeln beschrieben, die der Analyseeinheit in Form einer Konfigurationsdatei übergeben wird.

Wie `ethereal` verwendet `snort` die `libpcap` Bibliothek zum Auslesen der Netzwerk-Pakete. Die Funktionsweise beider Programme ist diesbezüglich im Wesentlichen identisch, die Pakete werden mittels der Bibliotheken dem Netzwerk-Stack entnommen und einem User-Level Prozess übergeben. Dieser ist für die Protokollierung bzw. Analyse der Pakete verantwortlich. Während `ethereal` einzig zur Protokollierung der Daten verwendet werden kann, ist `snort` ebenso in der Lage die Pakete hinsichtlich der ihm bekannten Regelbasis zu untersuchen und entsprechende Aktionen einzuleiten.

Regelbasis

Die Regelbasis besteht aus einzelnen Regeln, die in eine fest vorgegebene Konfigurationsdateie eingetragen werden müssen. Die Konfigurationsdatei kann wiederum include-Anweisungen enthalten, so dass der Administrator nicht an diese eine Datei gebunden ist und die Regelbasis strukturieren kann. Jede Regel setzt sich aus 6 Komponenten zusammen:

- der *Rule Action*: sie beschreibt, wie das Programm zu verfahren hat, wenn die Regel mit einem untersuchten Paket übereinstimmt,
- dem Protokoll: auf welches sich die Regel bezieht. Derzeit werden die Protokolle TCP, UDP und ICMP unterstützt,
- den IP-Adressen: sie beschreiben die Quell- und die Zieladresse des Paketes. Da `snort` den Datenverkehr eines gesamten Netzsegmentes überwachen kann, ist die Zieladresse zwingend erforderlich,
- den Port-Nummern: sie spezifizieren den Quell- und den Zielport des Paketes. Da ICMP keine Ports unterstützt, werden die Angaben bei diesem Protokoll ignoriert,
- dem Richtungsoperator: er legt die genaue Interpretation der Quell- und Zieladresse fest. Sie können sowohl undirektional als auch bidirektional verwendet werden und
- der *Activate / Dynamic Rule*: sie ermöglicht die Beschreibung zusätzlicher Kontextbedingungen für das empfangende Paket und die Angabe zusätzlicher Erläuterung, die bei der Protokollierung mit aufgenommen werden sollen.

Mit diesen Beschreibungsmöglichkeiten lassen sich komplexe Regeln zusammenbauen, die für bestimmte Pakete sogar unterscheiden können, von welchen BS das Paket generiert wurde. Da das

Erstellen der Regel nicht trivial ist, werden unter [16] vorgefertigte Signaturen angeboten, die nur noch den eigenen Anforderungen angepasst werden müssen.

Zusätzlich zu den Regeln können *Präprozessoren* definiert werden, die vor der Signaturanalyse angewendet werden. Diese erkennen spezielle Fehler oder Angriffversuche in einzelnen Paketen bzw. Paketströmen. Hierzu gehört das Erfassen von fragmentierten Paketen, deren Fragmentgröße kleiner als 128 Byte ist oder das Erkennen von Port-Scans.

Logging- und Response-Möglichkeiten

Für die Ausgabe der gewonnenen Informationen stellt `snort` eine Reihe von Ausgabe-Modulen zur Verfügung. So können die Daten dem Syslog-Dämon übergeben, in eine Datei gespeichert, an eine Windows-Workstation als WinPopup (*smbclient*) gesendet oder in eine Datenbank eingetragen werden. Zudem kann ein Abbild des verursachenden Paketes in eine Datei gespeichert oder über UNIX-Domain-Sockets einem anderen Programm übergeben werden. Das Format der Ausgabe kann mittels der Activate Rule den eigenen Anforderungen angepasst werden.

Darüber hinaus werden verschiedene Response-Aktionen angeboten. So kann durch eine Regel an einen oder an beide Verbindungspartner ein RST-Paket gesendet werden, um die Verbindung zu unterbrechen. Außerdem können ICMP-Pakete an den Sender zurückgeschickt werden, womit es unter anderem möglich ist, die Ergebnisse von Port-Scans zu verfälschen bzw. deren Aussagekraft zu verringern.

2.2.2.4 Betriebssystem-Audit Implementierungen

Obwohl LINUX bereits einen festen Platz im Marktsegment der Server-Betriebssysteme besitzt, existierte bisher keine Implementierung eines BS-Audit. Der Audit-Dämon vom H.E.R.T. (*Hacker Emergency Response Team*) [21] stellte lange Zeit die einzige Realisierung eines Audit-Dämon für LINUX dar. Derzeit existieren noch weitere Projekte, die sich mit der Umsetzung eines BS-Audit befassen. Hierzu zählt unter anderem das *BSM for Linux* Projekt [22] und das *Linux Operation System Audit* (LOSA) [6].

Da die meisten Audit-Projekte für LINUX teilweise nicht implementiert sind oder sich noch in einem sehr frühen Entwicklungsstadium befinden, wird lediglich auf das Projekt vom H.E.R.T und LOSA eingegangen.

Audit-Dämon vom H.E.R.T.

Das Projekt wurde ursprünglich für die Kernel-Version 2.0 implementiert. Derzeit sind auch Versionen für die Kernel-Releases 2.2 und 2.4 erhältlich. Am Funktionsumfang des BS-Audit hat sich seit der ersten Version nicht viel geändert. Das gesamte Programmpaket setzt sich aus dem Audit-Dämon und einen Kernel-Patch zusammen.

Audit-Dämon

Bei dem Audit-Dämon handelt es sich um ein Nutzerprogramm, welches Audit-Daten aus dem Kernel-Interface `/proc/audit` ausliest und in eine Datei schreibt. Es ist zudem für die Verwaltung der Audit-Trails und die Administration der Filterregeln verantwortlich.

Wurde der Audit-Dämon einmal gestartet, besteht keine Möglichkeit einer Rekonfiguration. Um das Programm zu beenden, muss diesem mittels kill ein Signal geschickt werden.

Kernel-Patch

Der Kernel-Patch enthält die Implementierung des Interfaces `/proc/audit` und der Stubs für die Audit-Funktionen zur Überwachung der Systemrufe:

- `execve()`, führt ein Programm aus,
- `open()`, dient dem Öffnen und dem Anlegen einer Datei,
- `modint()`, ist für das Laden eines Kernel-Moduls notwendig,
- `setuid()`, verändert die Nutzer-ID des aktuellen Prozesses,
- `listen()`, wird zum Einrichten einer Socket-Warteschlage verwendet,
- `connect()`, baut eine Netzwerkverbindung zu einem entfernten Rechner auf und
- `accept()`, dient dem Annehmen einer Netzwerkverbindung.

Darüber hinaus werden keine Systemrufe überwacht. Außerdem ist es nicht möglich zu unterscheiden, ob der Nutzer den Systemruf erfolgreich oder erfolglos ausgeführt hat. Dies stellt eine wesentliche Einschränkung bei der Interpretation der Audit-Daten dar.

Die Generierung der einzelnen Audit-Records erfolgt mit Hilfe der Systemfunktion `sprintf()`, welche die Übergabe eines Format-Strings und einer variablen Anzahl von Parametern gestattet. Sowohl die Implementierung der Audit-Stubs mittels `sprintf()` als auch die des Interfaces `/proc/audit` sind wenig performant, so dass beim Überwachen zusätzlicher Systemrufe bzw. beim Ansteigen der anfallenden Audit-Daten die Performance des Audit-Moduls die Gesamtleistung des Systems stark beeinträchtigen wird.

Linux Operating System Audit

Beim LOSA handelt es sich um die Implementierung eines BS-Audit, welches sich stark am BSM von Solaris orientiert. So wurden sowohl wesentliche Komponenten als auch das Konzept der Trail-Verwaltung an das BSM angelegt. Eine nicht unbedeutende Einschränkung des BSM ist das Fehlen einer Selektionsmöglichkeit anhand der betroffenen Ressource. Dieses ist bei Windows NT mittels der in das Dateisystem NFTS integrierten ACL (*Audit Access Control List*) realisiert worden. Für LOSA wurde ein ähnliches Konzept implementiert, welches den Mangel des BSM beseitigen soll.

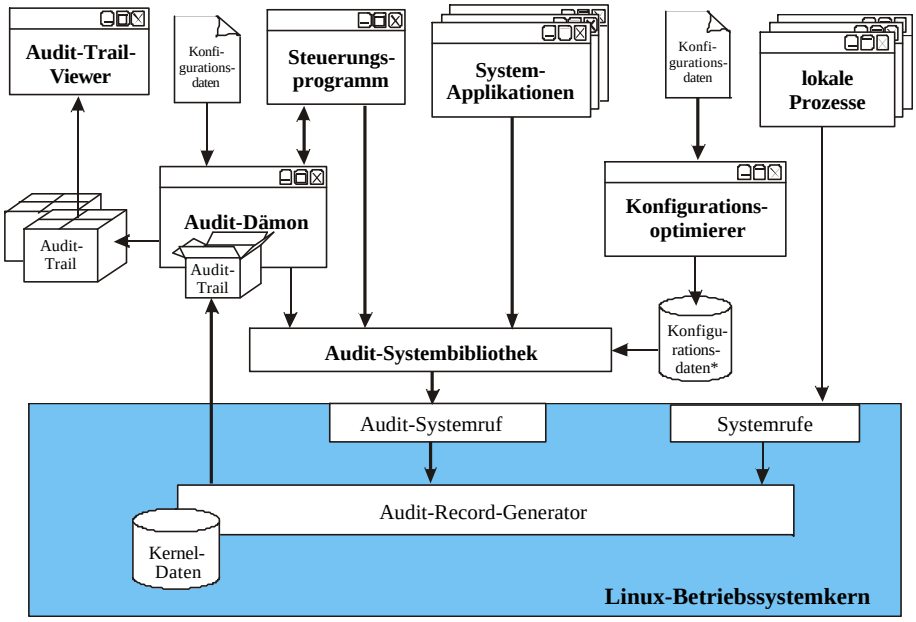

Abbildung 6 : Übersichtsbild der Bestandteile von LOSA

Da LOSA als BS-Audit für die Implementierung von HoNA verwendet wird, erfolgt in diesem Abschnitt eine kurze Vorstellung der wesentlichen Komponenten.

Aufzeichnungsumfang

Insgesamt können mittels LOSA 51 der über 200 Systemrufe überwacht werden. Dazu gehören alle System-verändernden (z.B. `settimeofday()`), Ressourcen-verändernden (z.B. `open()`), Prozess-beeinflussenden Systemrufe und die Funktionen `connect()` und `accept()` zum Aufbau einer Kommunikationsverbindung. Jedes protokollierte Ereignis wird in einem separaten Audit-Record abgelegt, welcher einen Zeitstempel, die eindeutige Identifikation des Initiators, die IP-Adresse des Rechners, von welchem aus die Aktion initiiert wurde, und die eindeutige Beschreibung der betroffenen Ressource enthält.

Bestandteile:

- **Audit-Dämon :** dient der Speicherung, der im Betriebssystem-Kern erzeugten Audit-Daten, und der Verwaltung der Audit-Trails.
- **Audit-Systembibliothek :** beinhaltet die Schnittstellen zu den Audit-Funktionen im Betriebssystem-Kern.
- **Steuerungsprogramm :** ist die allgemeine Managementschnittstelle des Systems.
- **Ereignis-Konfigurationsoptimierer :** übernimmt eine Vor-Optimierung der im Betriebssystem-Kern verwendeten Konfigurationsdaten.
- **Trail-Viewer :** ist ein einfaches Analyse-Tool zum Durchsuchen der Audit-Trails.

Konfigurierung der Aufzeichnungsgranularität

Neben der Beschränkung des Aufzeichnungsumfanges durch die Implementierung selbst, ist zur Laufzeit noch einmal eine Differenzierung der Audit-Events anhand des finalen Ausführungsstatus, der Audit-ID und des Ereignisses möglich. Zudem können sicherheitsunbedenkliche Ressourcen mit Hilfe der Audit-ACL ausgeblendet werden.

Die Konfigurierung des Systems ist jederzeit möglich. Sie erfolgt beim Starten des Systems mit Hilfe von Konfigurationsdateien und zur Laufzeit mittels des Steuerungsprogramms.

Sicherheitskonzept

Da das Audit-System selbst eine sicherheitsrelevante Anwendung darstellt, verfügt LOSA über ein eigenes Sicherheitskonzept. Dazu zählt die Einführung einer eindeutigen Audit-ID und einer Session-ID, eine eigene Verwaltung von Zugriffsrechten mittels Capabilities (Fähigkeiten) und die Verwendung von verschiedenen Betriebsmodi.

2.2.3 Grenzen klassischer Netzwerkmonitore

Bereits in den vorangegangenen Abschnitten wurde kurz auf die Nachteile eingegangen, die sich mit der zu nehmenden Verwendung von neuen Netzwerkprotokollen und –architekturen für klassische Netzwerkmonitore ergeben. Letztendlich lassen sich die Probleme auf die folgenden drei Punkte zusammenfassen:

Einsatz von High-Performance Netzwerk-Architekturen

Noch vor 5 Jahren war der Einsatz von High-Performance Netzwerk-Architekturen[8] auf den Backbone-Bereich[9] beschränkt. Mit dem Aufkommen preiswerter und zum Ethernet kompatibler Hardware in Form des Fast-Ethernets sind Transferraten von 100 MBps in den meisten lokalen Netzwerken nicht mehr selten. Seit den letzten zwei Jahren ist selbst der Einsatz von Gigabit-Ethernet für stark belastete Verbindungen im LAN-Bereich nicht unüblich.

Dies bedeutet heutzutage für einen Netzwerkmonitor, selbst wenn er nur eine Verbindung überwacht, ein Datenaufkommen von mehr als 10 MByte pro Sekunde. Wird der Monitor zusätzlich zu anderen Applikationen auf dem System betrieben, übersteigt dessen Ressourcenbedarf (CPU-Leistung, Speicher- und Festplattenbedarf) oftmals ein vertretbares Maß. Wird der Netzwerk-Monitor auf separaten Systemen betrieben, so müssen diese in der Regel mehr als eine Verbindung überwachen. Hierbei kann das Datenaufkommen 100 MByte pro Sekunde schnell überschreiten, was selbst die heutzutage leistungsfähigsten Workstations nicht immer auswerten können.

Verwendung von Switch-Technologien

Beim Einsatz von Bus-Technologien oder Hubs als Sternkoppler teilen sich alle Rechner die maximale Bandbreite des Mediums. Dies führt zu starken Performanceeinbusen aller am Netzsegment angeschlossenen Systeme, wenn lediglich zwei Rechner oder jeweils Rechnerpaare Daten untereinander austauschen. Beim Einsatz der Switch-Technologie erhält jeder Rechner nur die Pakete, die an ihn gerichtet sind. Damit kann im gesamten Netzwerk theoretisch ein Vielfaches der maximalen Bandbreite einer einzigen Verbindung genutzt werden. Ein 16 Port-Switch hätte bspw. einen maximalen Leistungsdurchsatz von 1,6 GBit pro Sekunde. In der Praxis wird dieser Datendurchsatz fast nie erreicht, jedoch liegt der Gesamtdurchsatz meist über den sonst möglichen 100 MBit.

Damit ergeben sich für klassische Netzwerk-Monitore zwei Probleme: Zum einen besitzt ein zentraler Monitor nicht mehr den Zugriff auf alle Pakete des Netzsegments. Die Lösung dieses Problems mit Hilfe von Span-Port führt wiederum dazu, dass die auszuwertende Bandbreite um ein Vielfaches erhöht wird. Bei einem Switch mit 16 Ports wären dies theoretisch bis 1,6 GBit pro Sekunde. Da solche Bandbreiten nicht mehr mittels Ethernet übertragen werden können, muss der Monitor auf einen Teil der Daten verzichten oder der Gesamtdurchsatz des Netzwerkes begrenzt werden.

Verschlüsselung des Paketstroms

Ein nicht unwesentlicher Teil der Angriffe auf IT-Systeme basiert auf den unzureichenden Authentifizierungsmöglichkeiten der Kommunikationspartner bzw. auf der uneingeschränkten Sichtbarkeit der Paketinhalte. Um dies zu unterbinden, kam es zur Einführung von Verschlüsselungs- und kryptographischen Authentifizierungsverfahren.

Während die ersten beiden Probleme klassischer Monitore durch die Erhöhung der Leistung des Systems, auf dem der Monitor betrieben wird, theoretisch gelöst werden könnten, stellt die Verschlüsselung des Paketstroms ein unüberwindbares Problem dar. Um den Inhalt der Pakete auswerten zu können, müsste dieser die Daten entschlüsseln können. Die hierzu notwendigen Schlüssel sind allerdings nur den Endpunkten des Tunnels bekannt, so dass ein zentraler Monitor keinen Zugriff auf sie hat und aus sicherheitstechnischen Gründen auch nicht haben sollte. Hinzu kommt, dass die Schlüssel (bei IPSec) in gesicherten Bereichen der BS-Kerns gehalten werden, auf

---

[8] Hierzu können alle Netzwerkarchitekturen mit einer Übertragungsrate von mehr als 100 MBps (*Mega Bits pro Sekunde*) gezählt werden. Die bekanntesten und verbreitetsten Architekturen sind Fast-Ethernet, FDDI (*Fiber Distributed Data Interface*) oder ATM (*Asynchronous Transfer Mode*).

[9] Als Backbone wird die Architektur zur Verbindung lokaler Netzwerke bezeichnet.

den Nutzerprozesse keinen Zugriff haben, wodurch selbst ein lokaler Monitor nicht in Besitz der Schlüssel kommen kann.

Werden die Schlüssel, allen Sicherheitsrichtlinien zum Trotz, an den Monitor exportiert, muss dieser die Daten immer noch entschlüsseln. Beim Einsatz eines zentralen Monitors bedeutet dies, dass er die Pakete vieler Verbindungen gleichzeitig dekodieren muss und bei einem lokalen Monitor bleibt immer noch eine Doppelbelastung des Systems, da die Daten vom Kern und vom Monitor entschlüsselt werden müssen. Selbst die leistungsstärksten Rechner bewältigen heutzutage kaum mehr als 50 bis 70 MBit pro Sekunde, so dass nicht einmal eine 100 MBit Verbindung für die Kommunikation voll ausgenutzt werden kann. Beim Einsatz von Netzwerk-Monitoren würde sich die Bandbreite noch weiter reduzieren.

Kapitel 3

Konzept eines Host-orientierten Netz-Audit für Linux

Nachdem in den vorangegangenen Abschnitten die Protokollierungsmöglichkeiten unter Linux und deren Grenzen beschrieben wurden, erfolgt in diesem Abschnitt die Vorstellung des HoNA-Konzeptes. Hierzu wird zunächst auf das Konzept bezüglich Platzierung und softwaretechnischer Realisierung im Allgemeinen eingegangen. Anschließend erfolgt die Beschreibung der im Rahmen dieser Arbeit entstandenen Integration in das Betriebssystem Linux im Hinblick auf deren Aufzeichnungsumfang, Filtermöglichkeiten und zur Verfügung gestellten Analyseprogramme.

3.1 Dezentraler Einsatz von Netzwerk-Monitoren

Im Abschnitt 2.2.1 wurde die Bedeutung, die der Platzierung eines Netzwerk-Monitors zukommt, erläutert. Hierbei wurde deutlich, dass durch die Verwendung von Switch-Technologien und dem zunehmenden Einsatz von High-Performance-Kommunikation die Realisierung einer Protokollierung mittels eines zentralen Netzwerk-Monitors immer schwieriger wird und in größeren Netzwerken schon heute nicht mehr möglich ist. Wenn die Stärkung des einzelnen Netzwerk-Monitors nicht mehr ausreichend ist oder dessen Sichtbarkeit auf einzelne Bereiche des Netzwerkes beschränkt ist, kann eine aussagekräftige und flächendeckende Überwachung nur durch eine dezentrale Lösung erreicht werden. Eine solche Lösung basiert auf folgender Prämisse:

> *Netzangriffe sind gegen bestimmte Rechner (bzw. netzspezifische Geräte) gerichtet. Infolge dessen sollten netzbasierte Audit-Daten, die ein Nachvollziehen dieser Angriffe ermöglichen, auf den betroffenen Ressourcen generiert werden.[2]*

Beim Konzept des Host-orientierten Netzwerk-Audit wird diese Prämisse mittels des Einsatzes von Kern-integrierten Netzwerk-Monitoren auf den Zielsystemen realisiert. Das heißt beim HoNA werden die Audit-Daten dezentral auf den jeweiligen Rechnern generiert. Hierbei können alle Rechner eines Netzwerkes oder die für eine Überwachung relevant erscheinenden Systeme mit einbezogen werden. Um einen Gesamtüberblick über die Aktivitäten, die in einem Netzwerk ablaufen, zu bekommen, können die Daten auf einen zentralen, mit unter speziell gesicherten Log-Server übertragen werden.

Als Beispiel betrachte man ein Netzwerk, wie es in Abbildung 7 dargestellt ist. Neben einem Firewall, einem WWW-Server und einem Datenbank-Server werden verschiedene Client-Stationen verwendet. Durch den Einsatz eines Switches für die Kommunikation zwischen den Servern könnte ein einzelner Netzwerk-Monitor (bspw. der abgebildete Log-Server) nur durch die Verwendung eines Span-Ports auf dem Switch den gesamten Datenverkehr protokollieren. Dies hätte allerdings zur Folge, dass der WWW-Server die Menge seiner Datenbankfragen auf die Bandbreite des Span-Ports beschränken müsste und in diesem Fall wäre die Protokollierung des angrenzenden lokalen Netzwerkes noch nicht mit eingeschlossen.

Verwendet man stattdessen einen dezentralen Ansatz, wäre es im Beispiel lediglich notwendig, Netzwerk-Monitore auf dem Firewall, dem WWW-Server, dem Datenbank-Server und einzelnen Arbeitsstationen einzurichten. Diese Netzwerk-Monitore protokollieren nur den Datenverkehr, der für sie selbst bestimmt ist. Damit muss der Durchsatz der eigenen Datenleitung nicht an die Bandbreite der Anbindung des zentralen Netzwerk-Monitors angepasst werden. Hinzu kommt, dass unwichtige

Stationen wie z.B. Thin-Clients von der Überwachung ausgeschlossen werden können, ohne dass zusätzliche Filterregeln angelegt werden müssen.

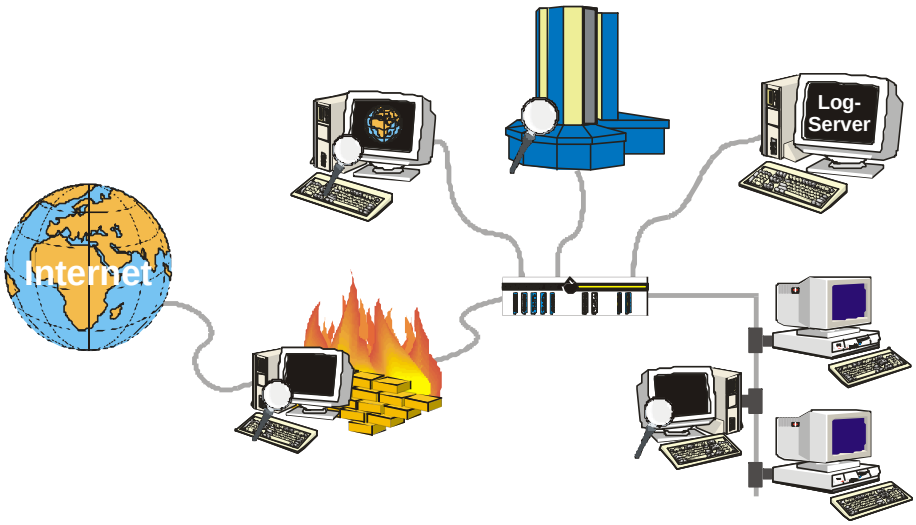

Abbildung 7 : Protokollierung der Netzwerk-Aktivitäten einzelner Rechner

Darüber hinaus ermöglicht der Einsatz dezentraler Netzwerk-Monitore eine feinere Granularität der Filterregeln auf den einzelnen Stationen, ohne die Überwachung auf den anderen Rechner zu beeinflussen.

3.2 Realisierung eines Kern-integrierten Netzwerk-Monitors für LINUX

Der Netzwerk-Stack des Betriebssystems übernimmt die Verarbeitung von Netzwerk-Paketen bis zur Transportschicht. Hierbei werden die Pakete entsprechend des Inhaltes ihrer Header analysiert und gefiltert. Dies entspricht im Wesentlichen der Funktionalität, die ein Netzwerk-Monitor erbringen müsste, um eine Überwachung des Systems zu gewährleisten. Durch das Einbringen von zusätzlichen Funktionen in den Netzwerk-Stack kann eine Protokollierung bereits an dieser Stelle erfolgen. Hierzu werden zwei Forderungen an das Betriebssystem gestellt:

- die Verfügbarkeit des Quelle-Codes und
- das Vorhandensein eines Protokollierungsmechanismus.

Die Quellen des Betriebssystems LINUX sind seit dessen Veröffentlichung frei zugänglich und können unter [3] heruntergeladen werden. Hinzu kommt, dass die Quellen ohne Lizenzgebühren von jedem verändert werden können und ausreichend Dokumentationen im Internet oder in der Fachliteratur (bspw. [11] und [13]) verfügbar sind.

Für die Protokollierung von Daten innerhalb der BS-Kerns steht unter LINUX der `printk`-Mechanismus zur Verfügung. Die Funktion ist nahezu identisch mit der Bibliotheksfunktion `printf(...)` zur formatierten Ausgabe von Textmeldungen. Die Daten werden mittels des Syslog-Dämons in einer frei wählbaren Datei abgelegt. Auf Standardsystemen ist dies `/var/log/messages`. Darüber hinaus können die im Abschnitt 2.2.2.4 erläuterten Betriebssystem-Audit-Konzepte verwendet werden.

Durch die Verwendung eines Kern-integrierten Netz-Audit kann ein Teil der im Abschnitt 2.2.3 beschriebenen Probleme überwunden werden. Im Wesentlichen ergeben sich folgende Vorteile gegenüber eine applikativen Lösung:

- Eine Überwachung von verschlüsselten Verbindungen wird möglich, da der Netzwerk-Stack die Pakete entschlüsseln muss, um die enthaltenen Informationen auswerten zu können.

- Der Overhead des Netzwerk-Monitors beeinträchtigt nur das Zielsystem und ist unabhängig vom Datenverkehr anderer Systeme.

- Die Implementierung des Netzwerk-Monitors ist unabhängig vom verwendeten Netzwerkzugriffsverfahren.

- Ein Kern-integrierter Netzwerk-Monitor ist in der Lage die Bandbreite der Netzwerkanbindung des Zielsystems zu regulieren. Damit kann eine vollständige Überwachung des Paketstroms gewährleistet werden.[10]

- Die Protokollierung der Netzwerkdaten ist durch die Mechanismen des Betriebssystems geschützt und kann vor potentiellen Angreifern verborgen werden.

- Die Sichtweite des Gesamtsystems kann durch den Einsatz zusätzlicher Monitore auf bisher nicht überwachten Systemen leicht erweitert werden.

- Der Ausfall eines Monitors beeinträchtigt nur die Überwachung eines einzelnen Rechners.

Durch die Beschränkung des Monitors auf die Analysefunktionen des BS-Kern können die Protokolle der Anwendungsschicht nicht überwacht werden. Hierzu müssten zusätzliche Funktionen des Netzwerk-Monitors die Nutzdaten der Pakete im BS-Kern analysieren. Da ein solcher Ansatz zu Ressourcenintensiv wäre und eine doppelte Belastung für das System darstellen würde, ist eine Erweiterung der Server-Applikation hinsichtlich eines applikativen Audits vorzuziehen.

3.2.1 Struktureller Aufbau des Host-orientierten Netz-Audit

Der strukturelle Aufbau eines Netzwerk-Monitors lässt sich in die funktionalen Schichten: Paketentnahme, Protokollanalyse | Filter, Datenaufzeichnung und Reportgenerierung einteilen. Bei der Realisierung eines applikativen Monitors müssen alle diese Schichten innerhalb der Module des Monitors implementiert werden. Beim Konzept von HoNA können einzelne Module durch bestehende Systemkomponenten des Protokoll-Stacks ersetzt werden. Ein wesentlicher Vorteil bei der Nutzung von bereits vorhandenen Systemkomponenten ist der erzielte Performancegewinn. Allerdings können nicht alle Komponenten ersetzt werden, da die Funktionalität des Netzwerk-Stacks im Wesentlichen unverändert bleiben sollte.

Paketentnahme und Protokollanalyse

Durch die Integration des Monitors in den Netzwerk-Stack entfällt die Notwendigkeit eines Entnahme-Moduls. Die Pakete passieren in jedem Fall die Funktionen des Netzwerk-Stacks und können an dieser Stelle analysiert werden.

Die Protokollanalyse erfolgt bereits in den Funktionen des Stacks. Hierbei werden die wichtigsten Informationen der Header, die aktuelle Konfiguration und der Zustand des Systems überprüft. Enthält ein Paket fehlerhafte oder nicht Police-konforme Daten, wird dieses verworfen. Beim Einsatz von

[10] Eine vollständige Überwachung muss die Auswertung jedes Paketes gewährleisten, welches das Zielsystem passiert. Dies kann, wenn dem Netzwerk-Monitor keine Auswertungskapazitäten mehr zur Verfügung stehen, durch das Ablehnen (Verwerfen) von Paketen erreicht werden.

HoNA wird an dieser Stelle zusätzlich ein Audit-Event generiert. Insbesondere bei der Verwendung von IPSec ist diese Integration und Ausnutzung der vorhandenen Funktionen zwingend erforderlich.

Um die Funktionalität des Systems nicht zu beeinflussen, werden keine zusätzlichen Überprüfungen eingefügt. Beim Bekannt werden eines neuen Fehlers kann dieser zwar auch trotz des Einsatzes von HoNA ausgenutzt werden, jedoch ist die Funktion des BS-Audits die Protokollierung von aufgetretenen Ereignissen und nicht die Korrektur von Sicherheitslöchern. Zudem würden zusätzliche Überprüfungen und Analysefunktionen die Performance des Systems beeinträchtigen, was gerade mittels HoNA im Gegensatz zu den klassischen Netzwerk-Monitoren verhindert werden soll.

Filter

Durch die Beschränkung der Audit-Events auf wesentliche Ereignisse bei der Analyse der Netzwerkpakete, wird der Protokollierungsumfang bereits bedeutend reduziert. Jedoch kann es notwendig sein, einzelne Events von der Protokollierung auszuschließen, da sie in der aktuellen Umgebung keine Sicherheitsverletzung darstellen oder bspw. bei der Rekonfigurierung des Netzwerkes zu häufig auftreten.

Für solche Fälle sind dynamisch konfigurierbare Filter notwendig. Diese erfordern Funktionalität, die gegenüber der reinen Protokollierung einen zusätzlichen Overhead erzeugen. Aus diesem Grund ist es sinnvoll, den Einsatz von Filtern als Option zu implementieren. Der Umfang und der damit verbundene Performanceverlust kann dann vom Administrator vor Ort eingeschätzt und entsprechend reguliert werden.

Datenaufzeichnung

Da die Generierung der Audit-Daten innerhalb des BS-Kerns erfolgt, muss eine zusätzliche Instanz vorhanden sein, die Daten in den Nutzerspeicher transferiert und später in eine Datei ablegt. Hierzu bietet sich die Nutzung eines bestehenden BS-Audit-Konzeptes an.

Beim BS-Audit werden ebenfalls Audit-Records als Resultat aufgetretener Ereignisse innerhalb des BS-Kerns generiert. Diese werden mittels eines sogenannten Audit-Dämons in den Nutzerspeicher transferiert und dort gespeichert. Der Audit-Dämon ist neben der Verwaltung der Audit-Trails für die Administration der gesamten Audit-Funktionalität innerhalb des BS-Kerns verantwortlich. Im Folgenden wird dieser Teil des Audit-Systems als Audit-Modul bezeichnet.

Je nachdem, inwieweit die Implementierung des BS-Audits angepasst werden kann, erfolgt die Integration von HoNA. Nahezu jedes Audit-Konzept bietet eine Schnittstelle für Nutzerapplikationen oder einen Record-Generator innerhalb des BS-Kerns. Ist deren Schnittstelle, insbesondere das angebotene Record-Format, ausreichend für die Integration von HoNA, können diese ohne weiteres genutzt werden. Ist die Definition eines eigenen Formates für die Audit-Records möglich, erlaubt dies mitunter eine zusätzliche Ressourceneinsparung.

Reportgenerierung

Die Reportgenerierung bzw. eine tiefergehende Analyse der Audit-Events ist nicht mehr Bestandteil von HoNA und muss mittels eines externen Programms realisiert werden. An dieser Stelle bietet sich der Einsatz eines IDS an. Arbeitet dieses bereits auf den BS-Audittrails, muss lediglich die Integration der neuen Record-Typen realisiert werden.

3.2.2 Bestandteile von HoNA für LINUX

Im vorangegangenen Abschnitt wurden die notwendigen Komponenten eines Netzwerkmonitors genannt und deren Funktionsumfang bzw. Integration in Bezug auf das HoNA-Konzept beschrieben. In diesem Abschnitt erfolgt eine Beschreibung einer möglichen Realisierung für LINUX.

Als Grundlage für die im Folgenden vorgestellte Infrastruktur diente das im Abschnitt 2.2.2.4 beschriebene BS-Audit LOSA. Abbildung 8 zeigt die Anpassung des Konzeptes an die Anforderung von HoNA. Darüber hinaus wurden einige Komponenten des ursprünglichen Konzeptes teilweise überarbeitet. Eine tiefergehende Beschreibung der notwendigen Erweiterungen erfolgt im Kapitel 4.

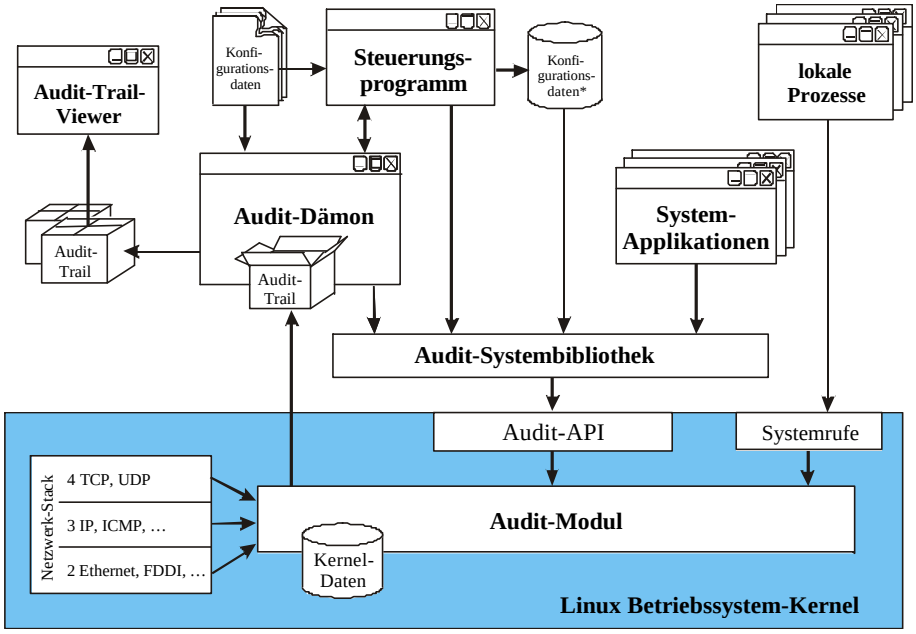

Abbildung 8 : Bestandteile von HoNA für LINUX

Zugriff auf den Netzwerk-Stack

Der Zugriff auf die Paketdaten erfolgt durch den Aufruf von Audit-Funktionen an den entsprechenden Stellen in den Analysefunktionen des Protokoll-Stacks. Da das Gesamtkonzept nicht als Modul ausgelegt, sondern als Option bei der Konfiguration des Kernel konzipiert ist, sind die Funktionsaufrufe immer mit Funktionalität gefüllt und nicht wie beim BSM von Solaris als reine Stubs zu betrachten. Die Implementierung der Funktion befindet innerhalb des Audit-Moduls. Damit soll verhindert werden, dass zu viele Änderungen am Originalcode durchgeführt werden müssen und ein großer, schwer zu wartender Patch entsteht.

Um den Overhead des Audit-Moduls möglichst gering zu erhalten, erfolgt vor dem Aufruf der eigentlichen Audit-Funktion bereits die Ausführung der Filteroperationen. Erst wenn die Filterfunktion entscheidet, dass das Ereignis aufgezeichnet werden soll, wird die zugehörige Audit-Funktion aufgerufen. Die Funktionalität der Filter befindet sich wiederum im Audit-Modul. Eine genaue Erläuterung der angebotenen Filterfunktionen erfolgt im Abschnitt 3.4.

Audit-Modul

Das Audit-Modul enthält den größten Teil des BS-Audit. Hier erfolgt die Aufzeichnung der Audit-Events, die Verwaltung der Audit-Record-Puffer, die Durchsetzung der Filterregeln und die Überprüfung der Sicherheitsrichtlinien.

Ein Großteil der Komponenten wurde aus der LOSA-Implementierung übernommen. Erweiterungen waren am Puffermanagement, den Filterregeln und dem Record-Generator notwendig. Da die gesamte Implementierung des Audit-Konzeptes verfügbar und hinreichend bekannt war, wurden die Komponenten teilweise komplett überarbeitet. Das Ziel war die Schaffung eine Architektur, die eine leichte Integration von neuen Record-Formaten erlaubt.

Audit-API

Das Audit-API ist die allgemeine Schnittstelle zum Audit-Modul und wurde von LOSA übernommen. Erweiterungen bzw. Veränderungen waren an dieser Stelle nicht notwendig.

Da die Anzahl der Systemrufe unter Linux auf 256 beschränkt ist, erfolgt der Aufruf aller Funktionen des Audit-Moduls über einen Systemruf, welcher als Parameter eine Funktions-ID und einen Zeiger auf die Argumente erhält. Um dem Nutzer die Verwendung der Funktionen des Audit-Moduls zu erleichtern, enthält das Audit-API eine mehrstufige Hierarchie von Funktionsaufrufen, die das Auswerten der Funktions-IDs und das Umwandeln der Argumente übernimmt.

Audit-Systembibliothek

Die Audit-Bibliothek ist das Gegenstück zum Audit-API des Kernel-Moduls. Sie enthält für jede Funktion des Audit-Moduls eine Bibliotheksfunktion, die das Verpacken der Argumente und das Setzen der Funktions-ID übernimmt.

Da gerade bei der Integration von neuen Record-Formaten eine Umstellung der Schnittstellen notwendig werden könnte, enthält die Bibliothek eine Versionssicherung. Nur wenn die Bibliothek mit dem Kernel erzeugt wurde, insbesondere mit dessen Header, ist die Verwendung der Funktionen des Audit-Moduls möglich. Damit kann zudem verhindert werden, dass ein Angreifer die Audit-Bibliothek unbemerkt austauschen oder das System mit einem anderen BS-Kern starten kann.

Audit-Dämon

Der Audit-Dämon dient der Administration des Puffermanagements im Audit-Modul und der Verwaltung der Audit-Trails. Für die Integration von HoNA waren nur geringe Änderungen am Audit-Dämon notwendig. Einzig die Administration der HoNA-Puffer musste hinzugefügt werden.

Der Inhalt und das Format der Puffer ist für den Audit-Dämon transparent. Die Übertragung der Daten in den Nutzerspeicher erfolgt durch das Audit-Modul. Der Audit-Dämon muss hierzu nur einen ausreichend großen Speicherbereich zur Verfügung stellen und die Audit-Service-Funktion des Audit-Moduls aufrufen.

Die Konfigurationsdatei des Audit-Dämon enthält die Größe der Puffer und die Konfiguration des dynamischen Zusatzpuffers (siehe Abschnitt 3.3.1.3). Bisher werden die beiden Record-Typen für das Host- und das Netzwerk-Audit unterstützt. Wird ein neuer Record-Typ eingeführt, müssen die Funktionen zum Einlesen der Konfigurationsdatei angepasst werden.

Steuerungsprogramm

Das Steuerungsprogramm ist die allgemeine Management-Komponente des Audit-Systems. In der ursprünglichen Implementierung waren drei Programme für die Administration des Audit-Systems notwendig. Deren Funktionalität wurde in das Steuerungsprogramm integriert, so dass bei allen zukünftigen Erweiterungen nur noch ein Programm gepflegt werden muss.

Da das Steuerungsprogramm für die Übertragung der Filterregel in das Audit-Modul verantwortlich ist, musste die Grammatik der Konfigurationsdateien und damit die Implementierung des Parsers erweitert werden. Eine Beschreibung der Regeln erfolgt im Abschnitt 3.4.2.

Außerdem wurden einige Shell-Kommandos hinzugefügt, die sich beim Arbeiten mit der ursprünglichen Implementierung als nützlich erwiesen haben bzw. die Sicherheit des Systems erhöhen.

System-Applikationen

Für die Aufzeichnung der Protokolle der Anwendungsschicht müssen die entsprechenden Applikationen erweitert werden. Die Audit-Bibliothek und das API des Kernels stellen eine Schnittstelle zur Protokollierung von Audit-Events aus Nutzerprogrammen zur Verfügung. Die im Rahmen dieser Arbeit vorgestellte Implementierung beschränkt sich auf die Integration von Audit-Events des BS-Kerns. Das Hinzufügen von Audit-Events aus Nutzerapplikationen bleibt für zukünftige Arbeiten offen.

Audit-Trail-Viewer

Der Trail-Viewer dient der Darstellung der Trail-Inhalte und ermöglicht eine einfache Suche auf ihnen. Obwohl sich diese Arbeit auf die Implementierung von HoNA beschränkt und dieses eigentlich keine Komponenten zur Analyse der Audit-Daten enthält, wurde der Audit-Trail-Viewer zur Darstellung von HoNA-Events erweitert (siehe Abbildung 9). Die Erweiterung des Trail-Viewers beschränkt sich auf die Darstellung der HoNA-Events, eine Suche ist nur für Events des Host-orientierten Audits möglich.

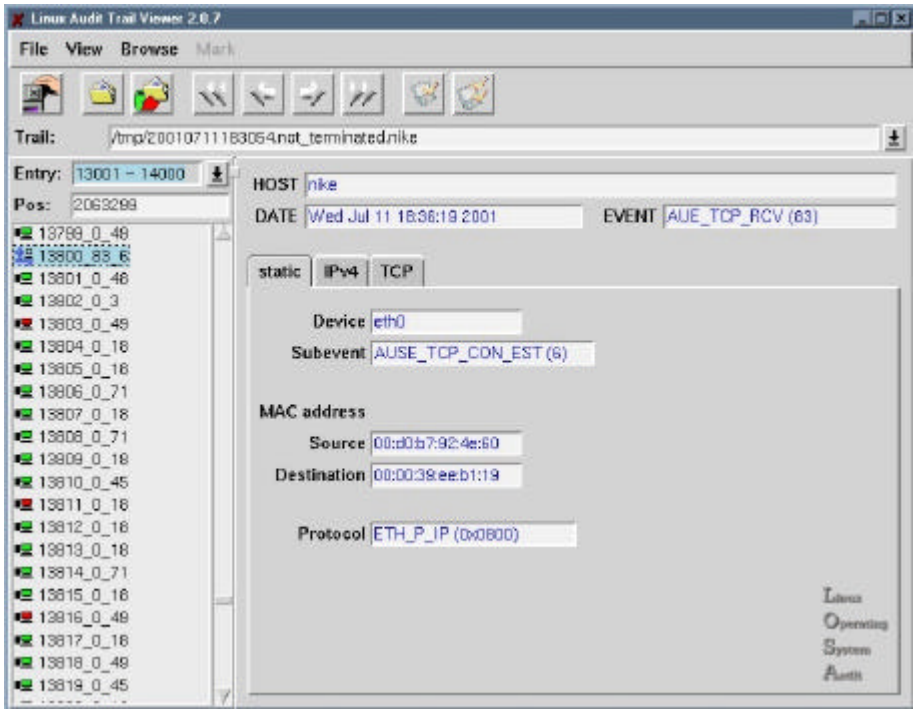

Abbildung 9 : Hauptfenster des Trail-Viewers

Die Implementierung des Trail-Viewers erfolgte mittels Tcl/Tk und dem Zusatzpaket Tix. Zum Einlesen des Audit-Trails werden Funktionen der Audit-Bibliothek verwendet. Dies wiederum erfordert die Erzeugung eines eigenen Tcl/Tk-Interpreters. Die Erzeugung des Tcl/Tk-Interpreters hat neben dem Performancegewinn den Vorteil, dass der Nutzer keinen eigenen Tcl/Tk-Interpreter benötigt, um die Daten der Audit-Trails darstellen zu können.

Abbildung 9 zeigt das Hauptfenster des Trail-Viewers nach dem ein Audit-Trail erfolgreich geöffnet und eine HoNA-Event ausgewählt wurde. Die linke Hälfte des Hauptfensters dient der Navigation im

Trail und beinhaltet den Event-Block[11], die Byte-Position im Trail und eine Auflistung der Events des aktuellen Blocks. Jeder Eintrag des Listfeldes enthält die Nummer des Ereignisses und eine kurze Beschreibung. Für Host-Events sind dies die Audit-ID des Initiators und die Event-ID. Die Farbe des Bildschirmsymbols gibt an, ob die Aktion erfolgreich (grün) oder erfolglos (rot) ausgeführt wurde. Für HoNA-Event wird neben der Event-Nummer, die Major- und die Minor-Event-ID dargestellt. Da es für HoNA-Events keine Unterscheidung anhand des finalen Ausführungsstatus gibt, hat das Symbol des Elementes keine weitere Bedeutung.

Das Format der rechten Seite des Hauptfensters ist abhängig vom ausgewählten Event und stellt dessen Inhalt dar. Im Beispiel wurde ein *TCP-Connection-Established*-Ereignis ausgewählt. Hierbei handelt es sich um ein HoNA-Event, welches Informationen über die Protokoll-Header der Schichten eins bis drei enthält. Die einzelnen Daten der Header werden in den jeweiligen *Notepad*-Blättern dargestellt. Für jedes von HoNA unterstützte Protokoll ist ein separates Notepad-Blatt definiert. Dies wird jeweils abhängig vom Event ein- bzw. ausgeblendet. Da das Ethernet-Protokoll das einzige unterstützte Protokoll der Schicht 1 ist, wurde das Minor-Event und der Gerätename mit in dieses Blatt übernommen.

3.3 Aufzeichnungsumfang

Ein wesentliches Kriterium für die Leistungsfähigkeit eines Audit-Systems ist der Aufzeichnungsumfang. Er wird durch die Anzahl der verschiedenen Audit-Events und den Informationsgehalt eines Audit-Events bestimmt. Man kann jedoch nicht allgemein sagen, um so mehr Daten aufgezeichnet werden, desto besser ist die Leistung des Audit-Systems. Ein BS-Audit besitzt in der Regel einen großen Ressourcenbedarf, der durch geeignete Filter- und Selektionsmöglichkeiten begrenzt werden muss. Um eine spätere Analyse und eine dauerhafte Speicherung der Audit-Daten zu ermöglichen, wird deren Menge schon bei der Protokollierung soweit wie möglich einzuschränken. Um dies zu ermöglichen, war es notwendig eine Analyse des Netzwerk-Stacks von LINUX durchzuführen. Bei dieser Analyse wurden alle hinsichtlich einer sicherheitskritischen Analyse notwendigen erscheinenden Überprüfungen innerhalb des Stacks herausgesucht. Sie bilden in der späteren Implementierung die Audit-Events. Darüber hinaus erfolgte eine Auswahl der notwendigen Protokolldaten, welche den Inhalt des Audit-Records definieren. In diesem Abschnitt wird zunächst der Inhalt eines Audit-Records erläutert, anschließend werden die einzelnen Audit-Events vorgestellt.

3.3.1 Inhalt eines Audit-Records

Wie bei LOSA wird auch für HoNA für jedes protokollierte Audit-Event ein Audit-Record generiert, dessen Format fest vorgegeben ist. Er enthält alle Informationen, die zu einem Event protokolliert und damit zu einem späteren Zeitpunkt aus dem Audit-Trail rekonstruiert werden können.

Jeder Audit-Record für HoNA lässt sich in drei Komponenten unterteilen:

- dem Record-Header,
- den Protokoll-spezifischen Daten und
- dem dynamischen Zusatzpuffer.

Der Inhalt eines Audit-Records ist abhängig vom Event. Während der Record-Header komplett für jedes Event gefüllt wird, können die verbleibenden Komponenten teilweise leer bleiben. So wird ein

[11] Mittels des Event-Blocks werden immer 1.000 Events für das Listenfeld selektiert. Dies ist aus implementierungstechnischen Gründen notwendig, da innerhalb eines Listenfeldes nicht mehr als 1.000 Einträge dargestellt werden können.

Event, welches bereits in der Internet-Schicht generiert wird, keine Daten der Transportschicht im Audit-Record ablegen. Dies hat zwei Gründe: Zum einen kann den Daten der Transportschicht nicht vertraut werden, da sie vom Netzwerk-Stack noch nicht analysiert wurden, und zum anderen sind sie nicht relevant, da sie von den Funktionen des Stacks niemals ausgewertet wurden.

Im Folgenden werden die Inhalte der Record-Komponenten im Einzelnen vorgestellt.

3.3.1.1 Record-Header

Der Record-Header fasst alle Informationen zusammen, die bei jedem Audit-Event protokolliert werden. Die folgenden Bezeichnungen für die Record-Elemente sind der Implementierung entnommen und werden teilweise für Host- und Netzwerk-Events verwendet, so dass die Bezeichnung nicht immer direkt den Inhalt beschreibt.

`date`
: Der Zeitstempel speichert die Systemzeit, wann das Event aufgetreten ist. Die Zeit wird als die Anzahl der verstrichenen Millisekunden seit 0:00 UCT, 1.Januar 1970 abgelegt.

`event`
: Die Major-Event-ID ist ein eindeutiger numerischer Identifier des protokollierten Ereignisses. Eine Beschreibung der Events erfolgt im Abschnitt 3.3.2.

`pid`[12]
: Die Minor-Event-ID ist eine numerische Beschreibung des Ereignisses. Für ein HoNA-Event können bis zu 16 verschiedene Minor-Events definiert werden. Die genaue Bedeutung des Minor-Events hängt vom Major-Event ab.

`hostname`
: Der Rechnername enthält den aktuellen Namen des Rechners, auf dem das Ereignis aufgetreten ist.

`name`
: Der Gerätename beinhaltet den Namen des Gerätes, auf dem das Paket entgegengenommen wurde. Unter LINUX ist der Gerätename nicht eindeutig und gilt immer nur für die aktuelle Konfiguration. Der Name eines Ethernet-Interfaces kann `eth0` bis `eth16` sein. Die genaue Nummer ist abhängig davon, wann das Gerät aktiviert wurde. Dies ist wiederum abhängig von der Startkonfiguration des Systems.

`dev_type`
: Der Gerätetyp ist eine numerische Bezeichnung des Gerätes auf dem das Paket entgegengenommen wurde.

`protocol`
: Der Inhalt des Protokolleintrages ist abhängig vom verwendeten Gerätetyp. Für Gerätetypen, die eine Hardware-Adresse besitzen, bspw. Ethernet oder Tokenring, enthält der Eintrag `protocol` das auf der Host-an-Netz Schicht verwendete Protokoll. Anderenfalls, bspw. bei der Verwendung des Point-to-Point Protokolls, ist in diesem Record-Element das auf der Internet-Schicht verwendete Protokoll hinterlegt.

---

[12] Die Bezeichnung `pid` ist von LOSA übernommen worden und steht für die Prozess-ID. Da diese für Netzwerk-Events nicht benötigt wird, wurde dieser Eintrag für die Sub-Event-ID übernommen.

3.3.1.2 Protokoll-spezifische Daten

Die Protokoll-spezifischen Daten des Audit-Records sind wiederum in die drei aufgezeichneten Protokoll-Schichten aufgeteilt. Für verschiedene Audit-Events kann ist es notwendig, für jede dieser Protokollschichten Daten im Audit-Record abzuspeichern, so dass der Audit-Record Einträge für alle drei Schichten enthalten muss.

Auf jeder Schicht des TCP/IP-Modells werden mehrere Protokolle in die Überwachung mit einbezogen. Die Anzahl der gespeicherten Protokolle ist abhängig vom Audit-Event bzw. in welcher Protokollschicht das Ereignis aufgetreten ist. Es ist nicht möglich, in einem Audit-Record die Daten für zwei verschiedene Protokolle einer Protokollschicht aufzuzeichnen. Demzufolge ist es notwendig, dass der Audit-Record Speicherplatz für die Daten von drei verschiedenen Protokolle bietet. Die Daten der drei Protokollschichten werden in drei Containern zusammengefasst, die entsprechend dem Ereignis zu interpretieren sind.

Im Folgenden werden die einzelnen Container im Einzelnen vorgestellt.

Host-an-Netz Schicht Container

Auf der Host-an-Netz Schicht wird bisher nur das Ethernet-Protokoll unterstützt, für alle weiteren Protokolle existieren bisher keine Implementierungen. Die Daten des Ethernet-Protokolls werden in Form des Ethernet-Headers abgelegt. Dieser wird direkt in den Audit-Record kopiert und später in diesem Format im Audit-Trail abgelegt.

Für die Implementierung weiterer Protokolle muss die Struktur des Audit-Records erweitert werden oder die Daten des Protokolls ohne eine genaue Beschreibung der Struktur im Audit-Record abgelegt werden. Hierzu enthält jeder Container für die einzelnen Protokollschichten ein `raw`-Element, dessen Größe fest vorgegeben ist. Für die Host-an-Netz Schicht wird die Größe des `raw`-Elementes durch den Ethernet-Header auf 16 Bytes festgelegt.

Internet Schicht Container

Auf der Internet Schicht können die Header der Protokolle IP, ARP und IPv6 abgelegt werden. Wie für das Ethernet-Protokoll enthält der Audit-Record die Strukturen der Protokoll-Header, so dass diese direkt in den Record kopiert werden können. Für das Internet Protokoll der Version 6 ist im Audit-Record nur Speicherplatz für den Basis-Header vorgesehen. Die Daten der übrigen Header werden im dynamischen Puffer abgelegt.

Die Größe des `raw`-Elementes der Internet Schicht wird durch den IPv6-Basisheader auf 40 Byte festgelegt.

Transportschicht Container

Äquivalent zu den bisher beschriebenen Schichten, enthält der Audit-Record für die Protokolle TCP, UDP, ICMP, ICMPv6, IGMP und IPIP die Strukturen der Protokoll-Header. Die Daten anderer Protokolle müssen entweder im `raw`-Element oder im dynamischen Zusatzpuffer abgelegt werden.

Obwohl die Protokoll-Erweiterung IPSec unter LINUX für empfangende Pakete oberhalb von IP behandelt wird, werden dessen Daten nicht im Container der Transportschicht gespeichert. Für IPSec werden neben den Daten der IP-Schicht die Informationen der SA im Record vermerkt. Da diese sowohl IPv4- als auch IPv6-Adressen enthalten können und bei der Verwendung von IPv6-Adressen der Speicherplatz des Containers nicht ausreichend ist, werden die Daten der SA immer im dynamischen Zusatzpuffer hinterlegt.

Auf der Transport-Schicht ist das `raw`-Element auf 20 Byte beschränkt.

3.3.1.3 Dynamischer Zusatzpuffer

Der dynamische Zusatzpuffer wurde aus der LOSA-Implementierung übernommen und dient der Speicherung von Zusatzinformationen, für die kein Eintrag im Record vorgesehen ist. Das Format des Pufferinhaltes ist, wie in Abbildung 10 dargestellt, auf die Angabe der Anzahl der Pufferelemente und deren Größe beschränkt.

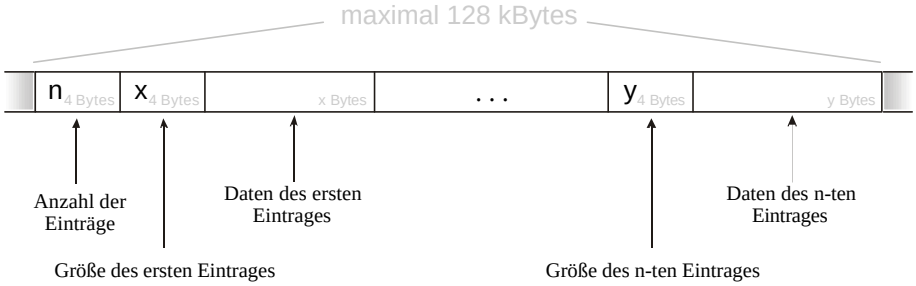

Abbildung 10 : Format des dynamischen Zusatzpuffers

Die genaue Anzahl und die Struktur der Puffer-Elemente ist abhängig vom Audit-Event und muss bei der Auswertung des Audit-Trails entsprechend interpretiert werden. Eine genaue Beschreibung ist im Abschnitt 3.3.2 zu finden. Für den Großteil der protokollierten Ereignisse wird der Zusatzpuffer nicht benötigt.

3.3.2 Netzwerk-Events

Bereits zu Beginn des Abschnittes wurde darauf eingegangen, dass bei einem Kern-integrierten Netzwerk-Monitor die Analyse Funktionen des Netzwerk-Stacks verwendet werden. Ein protokolliertes Audit-Event ist somit das Ergebnis eines aufgetretenen Fehlers innerhalb der Analyse der Funktionen des Stacks. Durch HoNA werden keine zusätzlichen Analysefunktionen hinzugefügt, um vom Netzwerk-Stack unerkannte Fehler aufzudecken.

In diesem Abschnitt erfolgt die Beschreibung aller protokollierten Audit-Events. Die hier vorgestellten Ereignisse repräsentieren nicht die gesamte Bandbreite der Protokollierungsmöglichkeiten. Die Funktionen des Netzwerk-Stacks enthalten wesentlich mehr Analysepunkte, jedoch sind die meisten aus dem Blickwinkel einer Sicherheitsanalyse nicht notwendig und wurden aus diesem Grund in dieser Arbeit nicht berücksichtigt.

Ein HoNA-Event setzt sich aus einem Major- und Minor-Event zusammen. Eine derartige Gruppierung der Audit-Events wurde notwendig, da die Implementierung von LOSA nur eine fest vorgegebene Anzahl von Events protokollieren kann. Insgesamt existieren 10 Major-Events. Für jedes dieser Major-Events können bis zu 16 Minor-Events, auch als Sub-Events bezeichnet, definiert werden. Nicht für jedes Major-Event wurden 16 Sub-Events definiert.

3.3.2.1 Audit-Events der Host-an-Netz Schicht

Die Host-an-Netz Schicht wird bei LINUX fast ausschließlich durch die Treiber der Netzwerk-Karten implementiert. Wenn die Funktionen der Kartentreiber die Analyse des Netzwerk-Paketes abgeschlossen haben, übergeben sie es der Funktion `net_rx_action(...)`. Sie entscheidet, wie mit dem Paket verfahren werden soll.

Auf eine Protokollierung von Fehlern, die innerhalb der Kartentreiber auftreten, wurde verzichtet, da sie keine wesentlichen Informationen für eine spätere Analyse enthalten und selten ein Zeichen für IT-Sicherheitsverletzungen sind. Allerdings muss man bereits an dieser Stelle differenzieren, da beim Einsatz von IP over ATM der gesamte für ATM notwendige Protokoll-Stack innerhalb der Host-an-

Netz Schicht liegt. Es kann also durchaus notwendig sein, zusätzliche Audit-Events innerhalb dieser Schicht anzusiedeln. Jedoch wurde ATM im Rahmen dieser Art nicht betrachtet, so dass lediglich zwei Audit-Events der Host-an-Netz Schicht zugeordnet werden können.

AUE_NO_PROTO

Das Audit-Event `AUE_NO_PROTO` wird innerhalb der Funktion `net_rx_action()` generiert und beinhaltet nur das Sub-Event `AUSE_PROTO_BAD_TYPE`. Das Audit-Event wird immer dann generiert, wenn ein Paket eingetroffen ist, welches das Zielsystem nicht verarbeiten kann, d.h. das dafür notwendige Protokoll dem System nicht bekannt ist. Bspw. würde jedes IPv6-Paket ein solches Event verursachen, wenn das Internet Protokoll der Version 6 nicht in der aktuellen Konfiguration des Systems unterstützt wird.

Da bereits bis zu dieser Schicht ein nicht unwesentlicher Teil an Ressourcen verbraucht wird, kann mittels eines einfachen Paket-Flooding über ein nicht unterstütztes Protokoll ein DoS-Angriff ausgeführt werden, ohne das der Administrator eine Möglichkeit der Abwehr besitzt. Bei der Protokollierung des Events wird die Ethernet-Adresse im Audit-Record abgelegt, so dass der Administrator die Möglichkeit bekommt, den Verursacher zu identifizieren.

AUE_ARP_RCV

ARP-Anfragen werden unter LINUX durch die Funktion `arp_rcv(...)` behandelt. Innerhalb dieser Funktion erfolgt die Analyse des Paket-Headers hinsichtlich Fehlern und die Überprüfung der Anfrage bzgl. der Police-Konfiguration des Rechners.

Für das *Address Resolution Protocol* wurden acht verschiedene Minor-Events definiert, die im Folgenden vorgestellt werden.

AUSE_ARP_BAD_SIZE
: Dieses Event wird immer dann generiert, wenn auf dem Rechner ein Paket eintrifft, dessen Größenangabe für die Hardware-Adresse nicht mit der zu erwartenden Größe übereinstimmt oder wenn die übermittelte Größe für die IP-Adresse ungleich vier Bytes ist.[13]

AUSE_ARP_DENY
: LINUX unterstützt die Möglichkeit einzelne Protokolle auf Netzwerk-Geräten (Interfaces) zu sperren. Trifft ein ARP-Paket auf einem gesperrten Interface ein, wird dieses Audit-Event generiert.

AUSE_ARP_NOT_IP
: Innerhalb des ARP-Headers wird die zu verwendende IP-Version übertragen. Ist diese verschieden von IPv4, wird ein Audit-Event mit `AUSE_ARP_NOT_IP` generiert.

AUSE_ARP_BAD_DEV
: Da die Pakete des ARP nur innerhalb eines Netzsegmentes übertragen werden, kann man immer davon ausgehen, dass die Hardware-Adresse vom gleichen Typ sein muss wie die des Interfaces, auf dem das Paket eingetroffen ist. Sind diese beiden Typen verschieden voneinander, erfolgt die Generierung eines Audit-Events.

AUSE_ARP_NOT_ETHER
: Ist das Paket auf einem Ethernet-Device eingetroffen, muss die ARP-Anfrage ebenfalls eine Ethernet-Adresse verlangen.

---

[13] Mittels ARP werden nur IPv4-Adressen aufgelöst. Somit muss die Größe der IP-Adresse immer 4 Byte betragen. Das Auflösen von IPv6-Adressen erfolgt mittels ICMP der Version 6.

AUSE_ARP_BAD_TYPE

Die aktuelle Implementierung von LINUX unterstützt nur ARP-Pakete, die entweder ein ARP-Request oder ein ARP-Reply enthalten. Alle anderen Anfragen werden verworfen. Für ein solches Paket würde das Audit-Event AUSE_ARP_BAD_TYPE generiert werden.

AUSE_ARP_BAD_DEST

ARP-Anfragen können nur auf Unicast-Adressen bezogen werden. Alle Anfragen mit anderen Zieladressen können nicht beantwortet werden. Dies gilt ebenfalls für Adressen aus dem Netz 127.0.0.0. Hierbei handelt es sind um eine Adresse für ein sogenanntes Loopback-Interface, welche keine ARP-Anfragen stellt.

AUSE_ARP_BAD_SOURCE

Enthält das Paket im Payload keine Quelle-Adresse, obwohl es sich um ein ARP-Reply handelt, oder können die Routing-Funktionen keinen Pfad zu der übermittelten Quelle-Adresse finden, wird ein Audit-Event mit dem Minor-Event AUSE_ARP_BAD_SOURCE generiert.

3.3.2.2 Audit-Events der Internet-Schicht

Auf der Internet Schicht werden die Protokoll IP, IPv6, ICMP, ICMPv6, IGMP, IPIP und die IP-Erweiterung IPSec überwacht. Für jedes Protokoll wurde ein Major-Event festgelegt. Die Unterscheidung der aufgetretenen Fehler innerhalb des Protokolls erfolgt wiederum mittels Minor-Events.

AUE_IP_RCV

Die Bearbeitung von Paketen der IP-Schicht verteilt sich innerhalb des Netzwerk-Stacks auf mehrere Funktionen und ist in sich wiederum in Schichten bzw. Verarbeitungsebenen eingeteilt. In Abbildung 11 sind die notwendigen Funktionen zur Verarbeitung eines IPv4-Paketes dargestellt. Die Verarbeitung beginnt mit der Funktion `ip_rcv(...)` und endet für Pakete, die an den Rechner selbst gerichtet sind, in der Funktion `ip_local_deliver_finish(...)` und für weiterzuleitende Pakete in der Funktion `ip_forward(...)`.

In der hier vorgestellten Implementierung werden lediglich Pakete protokolliert, die an den Rechner selbst gerichtet sind. Somit enthält der Funktionszweig für weiterzuleitende Pakete keine HoNA-Funktionalität.

Im Folgenden werden die Minor-Events für das *Internet Protokoll* der Version 4 vorgestellt. Dabei werden die Funktionen genannt, in denen die Protokollierung des Ereignisses angestoßen wird, um sie in die in Abbildung 11 dargestellte Paketverarbeitung von LINUX einordnen zu können.

AUSE_IP_BAD_SIZE

Entspricht die Größe des Headers nicht dem erwarteten Wert, wird innerhalb der Funktion `ip_rcv(...)` das HoNA-Event AUSE_IP_BAD_SIZE generiert.

AUSE_IP_BAD_SRC

Dieses Event wird innerhalb des Routing-Codes von LINUX generiert, wenn ein Paket eingetroffen ist, dessen Quelladresse eine Broadcast-, Multicast- oder Loopback-Adresse ist, die Adresse aus dem *Zeronet*[14] stammt oder zur Quelladresse kein Pfad bekannt ist bzw. gefunden werden kann. All diese Kontrollen erfolgen für Unicast-Adressen in der Funktion

[14] In einem *Zeronet* werden alle Adressen des Formates 0.xx.yy.zz zusammengefasst. IP-Adressen mit 0 als Netznummer beziehen sich auf das aktuelle Netz, eine solche Adressierung ist unter LINUX nicht möglich.

`ip_route_input_slow(...)` und für Multicast-Adressen in der Funktion `ip_route_input_mc(...)`.

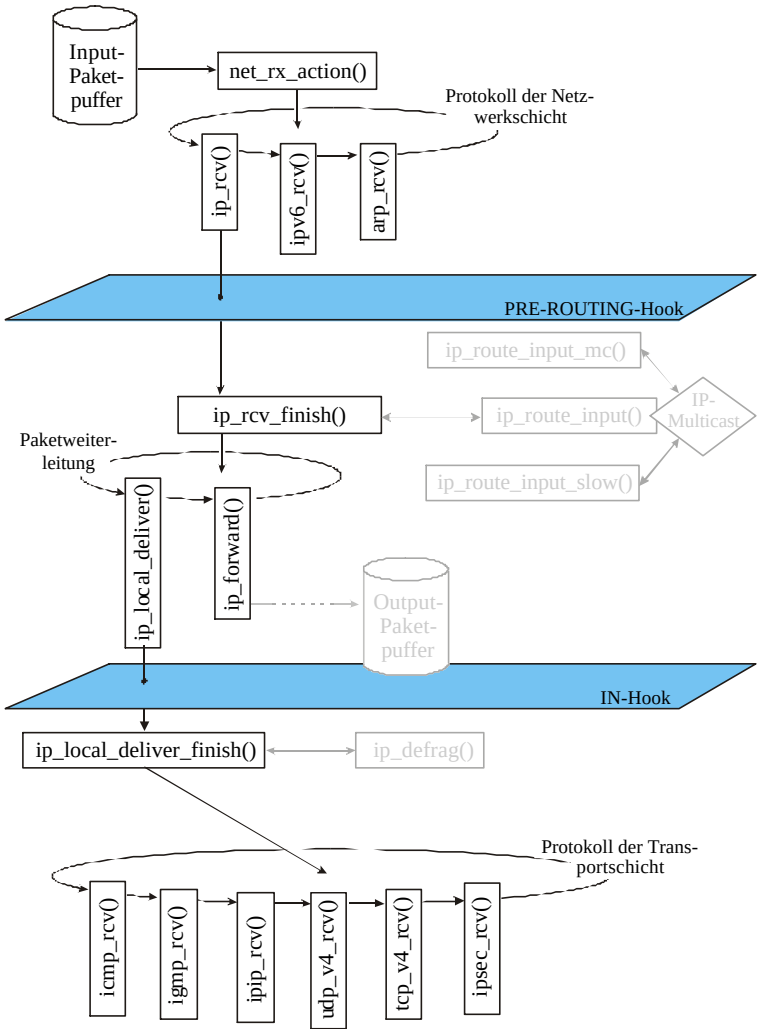

Abbildung 11 : Lokale Paketverarbeitung für IPv4

AUSE_IP_DENY

Sollte auf einem für IP gesperrtem Interface ein IP-Paket eintreffen, wird dieses Event generiert. Die Überprüfung erfolgt in den Funktionen `ip_route_input_mc(...)` und `ip_route_input(...)`.

AUSE_IP_BAD_DEST

Ist das Ziel eine Loopback-Adresse, stammt es aus dem Zeronet oder gehört es zu einer ungültigen Netzklasse, wird in der Funktion `ip_route_input_slow(...)` das Event `AUSE_IP_BAD_DEST` generiert.

AUSE_IP_BAD_TYPE

Pakete, die in der bisherigen Paketverarbeitung einem anderen Protokoll als dem IP zugeordnet wurden, aber trotzdem in die Routing-Funktionen von IP gelangt sind, werden an dieser Stelle verworfen. Dies erfolgt wiederum in den Funktionen `ip_route_input_mc()` und `ip_route_input_slow(...)`.

AUSE_IP_TOBIG

In der Funktion `ip_defrag(...)` werden die IP-Fragmente zusammengesetzt. Wird das endgültige Paket größer als 64 kByte, ist dieses zu groß und muss verworfen werden.

AUSE_IP_BAD_PROTO

Nach der Bearbeitung des Paketes auf der IP-Schicht wird dieses von der Funktion `ip_local_deliver_finish(...)` der Receive-Funktion der nächsten Protokollschicht übergeben. Enthält das Paket ein Protokoll, dass in der aktuellen Konfiguration des Rechners nicht unterstützt wird, kann das Paket an dieser Stelle nicht weiterverarbeitet werden.

AUSE_IP_BAD_VER

Neben der Größe des Headers überprüft die Funktion `ip_rcv(...)` die Korrektheit der übermittelten IP-Versionsnummer.

AUSE_IP_BAD_CHECK

Dieses Event wird innerhalb der Funktion `ip_rcv(...)` generiert, wenn der IP-Header eine inkorrekte Prüfsumme enthält.

AUSE_IP_SRR_DENY

So wie Protokolle für einzelne Geräte gesperrt werden können, ist es auch möglich das IP-Source-Routing zu unterbinden. Trifft ein Paket mit Source-Routing-Informationen auf einem solchen Interface ein, wird die Verarbeitung des Paketes innerhalb der Funktion `ip_rcv_finish(...)` verweigert.

AUSE_IP_SRR_FAILED

Sollte die Verarbeitung von Source-Routing-Paketen gestattet sein und tritt bei der Auswertung der Informationen des übermittelten Headers ein Fehler auf, wird in der Funktion `ip_rcv_finish(...)` das HoNA-Events `AUSE_IP_BAD_SRR` generiert.

AUSE_IP_SRR_RCV

Da Source-Routing an sich unüblich ist und ein nicht unwesentliches Sicherheitsrisiko darstellt, wird jedes Paket mit Source-Routing-Informationen innerhalb der Funktion `ip_rcv_finish(...)` protokolliert.

AUE_IPV6_RCV

Die Verarbeitung von IPv6-Paketen erfolgt innerhalb eines separaten Funktionszweiges des Netzwerk-Stacks. Der Aufbau des Zweiges ist mit dem der Version 4 aus Abbildung 11 vergleichbar. Die Namen der Funktionen sind ähnlich gewählt und ermöglichen somit eine bessere Orientierung innerhalb des Quell-Codes.

AUSE_IPV6_NO_ROUTE

Enthält das Paket einen next-Header-Eintrag, der nicht verarbeitet werden kann, muss das Paket verworfen werden.

AUSE_IPV6_BAD_VER

Entspricht die übertragene Versionsnummer nicht dem erwarteten Wert, wird die Weiterverarbeitung des Paketes innerhalb der Funktion `ipv6_rcv(...)` verweigert.

AUSE_IPV6_BAD_PROTO

Ähnlich dem Event für IPv4. Sollte das übertragene Protokoll der übergeordneten Schicht nicht bekannt sein, wird das Paket verworfen.

AUSE_IPV6_BAD_SIZE

Im Gegensatz zu IPv4 kann ein IPv6-Paket mehrere IP-Header enthalten. Sie werden als Erweiterungs-Header bezeichnet. Entspricht die Größe eines dieser Header nicht dem festgelegtem Wert, muss das Paket verworfen werden. Da die Auswertung der Erweiterungs-Header an verschiedenen Stellen des IPv6-Stacks erfolgt, ist die Generierung des Audit-Events ebenso verstreut. Um den verursachenden Header-Typ später rekonstruieren zu können, wird die ID des Header-Typs in den oberen 16 Bits des `pid`-Eintrages hinterlegt.

AUSE_IPV6_NO_DEST

Bisher unterstützt die IPv6-Implementierung von LINUX keine Destination-Header. Enthält das Paket trotzdem einen solchen Header, so wird es verworfen.

AUSE_IPV6_BAD_PARAM

Für den *hop-by-hop* Erweiterungs-Header werden nur die beiden Typen *Router-Alarm* und *Jumbo-Payload* unterstützt. Enthält das Paket einen anderen Header-Typ, wird die Verarbeitung an dieser Stelle abgebrochen.

AUSE_IPV6_BAD_TYPE

In der ersten Version des IPv6-Standards war vorgesehen, zwischen strikten und losen Vorgaben für das Routing zu unterscheiden. Die aktuelle Version sieht nur noch loses Routing vor. Trifft ein Paket ein, das striktes Routing fordert, so wird es verworfen.

AUSE_IPV6_TOBIG

Nach den fest vorgegebenen Elementen des Routing-Headers folgen die IP-Adressen der Rechner, über die das Paket übertragen werden soll. Stimmt die im Header angegebene Anzahl nicht mit der Anzahl der übertragenen IP-Adressen überein, wird das Paket nicht weitergeleitet.

AUSE_IPV6_BAD_ADDR

Ist eine der IP-Adressen des Routing-Headers eine Multicast-Adresse, wird das Paket ebenfalls nicht weiterverarbeitet.

AUSE_IPV6_FRAG_ERR

Tritt beim Zusammenbauen des IP-Paketes ein Fehler auf, wird das HoNA-Event `AUSE_IPV6_FRAG_ERR` generiert.

AUE_ICMP_RCV

Das ICMP nutzt IP für die Übertragung der Pakete. Erst wenn das Paket die IP-Schicht vollständig passiert hat, wird das Paket den Funktionen der ICMP-Schicht übergeben. In der Implementierung erfolgt der Aufruf dieser Funktionen gleichrangig mit den Funktionen für TCP und UDP, jedoch gehören ICMP nicht zur Transportschicht. Die einzelnen Schichten lassen sich in der Implementierung nicht so exakt vertikal anordnen, wie es im Modell oftmals dargestellt ist.

ICMP wird sowohl für IPv4 als auch für IPv6 genutzt. Obwohl beide Protokolle verschieden sind, enthalten sie doch gleichartige Ereignisse, so dass auf zwei separate Major-Events verzichtet wurde. Die folgenden Minor-Events gelten somit für ICMPv4 als auch für ICMPv6. Die genaue Interpretation des Ereignisses bei der Auswertung des Audit-Records muss anhand der IP-Version erfolgen.

AUSE_ICMP_BAD_SIZE

Entspricht die Größe des ICMP-Headers nicht dem erwarteten Wert, wird das Paket verworfen.

AUSE_ICMP_BAD_CHECK

Dieses Event wird immer dann generiert, wenn das Paket eine inkorrekte ICMP-Prüfsumme enthält.

AUSE_ICMP_BAD_TYPE

Sowohl für ICMPv6 als auch ICMPv4 ist nur eine bestimmte Anzahl von Nachrichtentypen zulässig. Enthält der Header einen anderen Typ, kann das Paket nicht weiterverarbeitet werden.

AUSE_ICMP_REQ

Da ICMP-Pakete innerhalb eines konfigurierten Netzwerkes nicht üblich sind, wird für die Version 4 jedes eingetroffene ICMP-Paket protokolliert. Bei der Version 6 wird mit diesem Minor-Event lediglich das Eintreffen eines Echo-Requests angezeigt, die Integration der anderen Typen kann zu einem späteren Zeitpunkt hinzugefügt werden.

Mittels ICMPv6 erfolgt auch das Management von IPv6-Gruppen, so dass die HoNA-Events des IGMP für die Version 6 unter das Major-Events `AUE_ICMP_RCV` fallen.

AUSE_ICMP_GM6_BAD_SIZE

Sollte der ICMP-Header IGMP-Informationen enthalten, so entspricht dessen Größe dem ICMP-Header plus 16 Byte für eine zusätzliche IP-Adresse. Ist die Größe des empfangenden Paketes verschieden davon, so wird dieses verworfen.

AUSE_ICMP_GM6_BAD_SRC

Ist die Quelleadresse der Anfrage keine Link-local-Adresse[15], wird die Anfrage nicht bearbeitet und das Paket verworfen.

AUSE_ICMP_GM6_DENY

Das Group-Management kann für einzelne Netzwerkgeräte untersagt werden. Trifft trotzdem ein Paket auf einem dieser Geräte ein, wird dieses HoNA-Event generiert.

AUE_IGMP_RCV

Für das Group-Management der Version 4 existiert ein separater Protokoll-Header, so dass dessen Bearbeitung auch in einem separaten Funktionszweig erfolgt.

AUSE_IGMP_BAD_SIZE

Dieses Event signalisiert das Eintreffen eines Paketes mit einer inkorrekten Header-Größe.

AUSE_IGMP_BAD_CHECK

Signalisiert das Eintreffen eines Paketes mit einer inkorrekte IGMP-Prüfsumme.

AUSE_IGMP_BAD_TYPE

Enthält das Paket einen unbekannten IGMP-Nachrichtentyp, wird dies mittels des HoNA-Events `AUSE_IGMP_BAD_TYPE` protokolliert.

AUSE_IGMP_DENY

Protokolliert das Empfangen eines IGMP-Paketes auf einen für IGMP gesperrtem Interface.

AUE_IPIP_RCV

Das IPIP-Protokoll dient dem Tunneln von IP-Paketen. Hierzu folgt nach dem ersten IP-Header ein weiterer IP-Header, an dessen Zieladresse das Paket weitergeleitet werden soll.

---

[15] Link-local-Adressen werden von einem Rechner für ein Interface verwendet, wenn keine Netzinformationen von einem Router vorliegen. Sie sind auf einen lokalen physikalischen Raum beschränkt und werden niemals von einem Router weitergeleitet.

AUSE_IPIP_NO_TUNNEL

Der Rechner, welcher IPIP-Pakete verarbeitet, speichert eine Liste gültiger IPIP-Tunnel. Kann das eingetroffene Paket keinem dieser Tunnel zugeordnet werden, wird die Weiterverarbeitung des Paketes abgebrochen.

AUE_IPSEC_RCV

In der für HoNA verwendeten Version 1.9 der IPSec-Implementierung FreeS/WAN erfolgt die Verarbeitung der IPSec-Header für empfangene Pakete in der Funktion `ipsec_rcv(...)`. Wie in Abbildung 11 dargestellt, wird diese Funktion nach der Abarbeitung des IP-Protokolls aufgerufen. Da IPSec-Pakete im Tunnel-Mode wiederum IP-Pakete enthalten können, wird der gesamte IP-Stack anschließend beginnend mit der Funktion `net_rx_action(...)` erneut durchlaufen. Somit entgehen auch getunnelte IP-Pakete nicht der Protokollierung mittels HoNA.

AUSE_IPSEC_BAD_TYPE

Enthält das Paket einen nicht unterstützten IPSec-Header, wird dieses Event generiert. Die verwendete Version unterstützt die Header AH, ESP und IPCOMP, welche wiederum bei der Konfigurierung des Kernels separat ausgewählt werden können.

AUSE_IPSEC_ESP_BAD_SIZE

Die Größe des ESP-Headers entspricht nicht den Vorgaben.

AUSE_IPSEC_AH_BAD_SIZE

Die Größe des AH-Headers entspricht nicht den Vorgaben.

AUSE_IPSEC_NO_SA

Für das eingetroffene Paket existiert keine gültige SA.

AUSE_IPSEC_INBOUND_BAD_SA

IPSec erlaubt die Aktivierung eines zusätzlichen Inbound-Checks, wobei die entschlüsselten Informationen einer zusätzlichen Kontrolle unterzogen werden. Wurde bei diesem Test keine gültige SA gefunden, wird dieses HoNA-Event generiert.

AUSE_IPSEC_LARVAL_STATE

Für das eingetroffene Paket konnte eine SA ermittelt werden, jedoch ist diese gesperrt und darf nicht verwendet werden.

AUSE_IPSEC_DEAD_STATE

Dieses Event ist ähnlich dem vorangegangenen HoNA-Event. In diesem Fall ist die SA bereits ungültig, jedoch noch nicht aus dem Speicher gelöscht.

Um das Übernehmen von aktuell verwendeten Schlüsseln verhindern zu können, sind diese nur für einen bestimmten Zeitraum gültig. Hierzu sind verschiedene Timer implementiert. Sie überwachen zwei Arten von Timeouts: einen sogenannten Soft-Timeout und einen Hard-Timeout. Wird ein Soft-Timeout erreicht, muss ein neuer Schlüssel ausgehandelt werden. Die Kommunikation kann bis zum Erreichen des Hard-Timeouts mit dem alten Schlüssel fortgeführt werden, danach werden alle weiteren Pakete verworfen. Eines der folgenden HoNA-Events wird nur generiert, wenn ein Hard-Timeout erreicht wurde. Soft-Timeouts sind für das Schlüsselmanagement notwendig und stellen keinen Fehler oder eine Sicherheitsverletzung dar.

AUSE_IPSEC_HARDBYTE_TO

Der Grenzwert für die Anzahl der übertragenen Bytes wurde erreicht.

AUSE_IPSEC_HARDADD_TO

Der Timeout für die Zusatzzeit wurde erreicht.

AUSE_IPSEC_USE_TO
Der Timeout für die Nutzungszeit wurde erreicht.

AUSE_IPSEC_PKT_TO
Der Grenzwert für die Anzahl der übertragenen Pakete wurde erreicht.

Nachdem eine gültige SA gefunden und die einzelnen Timer überprüft wurden, erfolgt die Überprüfung des zu verwendenden Algorithmus. Diese können wiederum beim Konfigurieren des Kernels aktiviert bzw. deaktiviert werden.

AUSE_IPSEC_BAD_AUTH
Der verwendete Authentifizierungsalgorithmus wird nicht unterstützt.

AUSE_IPSEC_ESP_BAD_TYPE
Der verwendete Verschlüsselungsalgorithmus wird nicht unterstützt.

Anschließend kann die Authentifizierung des Kommunikationspartners durchgeführt werden.

AUSE_IPSEC_AUTH_ERR
Die Authentifikation ist fehlgeschlagen.

Wie TCP unterstützt IPSec auch die Verwendung sogenannter Sliding-Windows, welche die Gültigkeit eines Paketes auf einen gewissen Zeitraum begrenzt. Sowohl der AH- als auch der ESP-Header beinhaltet eine Sequenznummer, die entsprechend überprüft werden kann.

AUSE_IPSEC_REPLAY
Die Sequenznummer des eingetroffenen Paketes ist nicht mehr gültig.

3.3.2.3 Audit-Events der Transport-Schicht

Auf der Transportschicht können die beiden Protokolle TCP und UDP mittels HoNA protokolliert werden. Für jedes Protokoll wurde ein Major-Event definiert und die Unterscheidung der Ereignisse erfolgt wiederum anhand der Minor-Events.

Obwohl TCP und UDP unabhängig vom genutzten Protokoll der Schicht drei sein sollten, ist deren Implementierung teilweise eng an diese Protokolle gekoppelt. Die TCP-State-Machine ist beinahe vollständig für beide Protokollversionen von IP implementiert, so dass das Einbringen der HoNA-Stubs ebenfalls zweimal erfolgen musste.

AUE_TCP_RCV

Die TCP-Implementierung wird im Wesentlichen durch die State-Machine bestimmt. Die auftretenden Ereignisse sind fast immer vom aktuellen Zustand abhängig. Um dies in die spätere Analyse mit einbeziehen zu können, wird der aktuelle Zustand der TCP-State-Machine in den oberen 16 Bits des `pid`-Eintrages abgelegt.

AUSE_TCP_BAD_DOFF
Da die im TCP-Header enthaltene Offset-Adresse für eine Vielzahl von Angriffen missbraucht werden kann, muss sie mit einer gewissen Vorsicht behandelt werden. Zeigt die Adresse bspw. in den Header des Paketes selbst, kann davon ausgegangen werden, dass das Paket fehlerhaft ist und nicht weiterverarbeitet werden muss.

AUSE_TCP_FULL_SYNQ
Das im Anhang A.2 beschriebene SYN-Flooding lässt sich nicht nur schwer unterbinden, jedoch kann beim Erreichen der maximal zulässigen offenen Verbindungen eine Meldung oder ein Audit-Event generiert werden.

AUSE_TCP_FULL_ACCEPTQ

Beim SYN-Flooding wird eine Verbindung nur unvollständig aufgebaut. Darüber hinaus könnte die Verbindung aber auch komplett aufgebaut und damit der Server blockiert werden. Ist die maximale Anzahl von offenen Verbindungen erreicht, müssen weitere einkommende Anfragen abgelegt werden.

AUSE_TCP_NO_SOCK

Jedes eingehende Paket, das an einen geschlossenen Port gerichtet ist, wird mit einem RST-Paket beantwortet. Da dies die Grundlage von Port-Scans ist, wird bei jeder Verbindungsanforderung an einen geschlossenen Port ein HoNA-Event generiert.

AUSE_TCP_BAD_FLAG

Befindet sich ein Socket im LISTEN-State, d.h. der zugehörige Prozess wartet auf eine Verbindungsanforderung, werden lediglich Pakete mit einem gesetzten SYN-Flag beantwortet. Um FIN-Scans oder ähnlich geartet Scans erkennen zu können, wird bei jedem Paket, dass an einen offenen Port gerichtet ist und bei dem nicht das SYN-Flag gesetzt ist, ein Audit-Event generiert.

AUSE_TCP_NO_ACK

Bei Paketen, die sich auf eine bestehende Verbindung beziehen, sind nur bestimmte Flags innerhalb des Paketes zulässig. Enthält das Paket ein anderes Flag, wird es nicht weiterverarbeitet.

AUSE_TCP_CON_EST

Signalisiert den kompletten, fehlerfreien Aufbau einer TCP-Verbindung.

AUSE_TCP_BAD_URG

Ist im aktuellen Paket das URG-Flag gesetzt, muss der *Urgent*-Zeiger des Headers ausgewertet werden. Tritt hierbei ein Fehler auf, wird dieses vom Netzwerk-Stack ignoriert. Um einen solchen Fehler später erkennen zu können, wird ein HoNA-Event generiert.

AUSE_TCP_BAD_SIZE

Ist die Größe des eingetroffenen Paketes ohne die Header der Schichten 2 und 3 kleiner als die erwartete Größe, wird das Paket nicht weiterverarbeitet. Beim Aufbau einer neuen TCP-Verbindung wird die Größe des TCP-Headers in der Socket-Struktur vermerkt, alle Paket dieser Verbindung sollten mindestens diese Größe haben.

AUSE_TCP_TO_LATE

Befindet sich der Socket im CLOSE-Zustand, werden keine eingehenden Anfragen bearbeitet und die zugehörigen Pakete verworfen.

AUSE_TCP_BAD_CHECK

Dieses HoNA-Event signalisiert das Eintreffen eines Paketes mit einer inkorrekten Prüfsumme im TCP-Header.

AUSE_TCP_AUTH_ERR

Seit der Kernel-Version 2.4.0 wurde mit der Einführung von Policy-Kontrollen in der Transportschicht begonnen. Tritt bei dieser Kontrolle ein Fehler auf oder entsprach das Paket nicht den geforderten Richtlinien, wird es nicht weiterverarbeitet.

AUSE_TCP_WRITE_ERR

Sollte bei einer bestehenden Verbindung ein Timeout auftreten, versucht der Rechner durch wiederholtes Senden eines Paketes die Verbindung zu überprüfen. Insgesamt werden drei Versuche durchgeführt. Wird der Empfang der Pakete vom Kommunikationspartner nicht bestätigt, wird die Verbindung geschlossen und ein Audit-Event generiert.

AUSE_TCP_OUT_OF_RES

Um den Ressourcenbedarf von verwaisten Verbindungen begrenzen zu können, wird diesen nur ein bestimmter Anteil an Ressourcen zugeteilt. Wird dieser Anteil überschritten, werden die Verbindungen geschlossen und alle von ihnen belegten Ressourcen freigegeben.

AUE_UDP_RCV

UDP ist ein verbindungsloses Protokoll und kann auf die Implementierung eines Zustandsautomaten verzichten. Demzufolge ist auch die Anzahl der HoNA-Events weitaus geringer.

AUSE_UDP_BAD_SIZE

Ist die Größe des eingetroffenen Paketes ohne die Header der Schichten 2 und 3 kleiner als die Größe des UDP-Headers, wird das Paket nicht weiterverarbeitet.

AUSE_UDP_NO_SOCK

Jede eingehende Verbindung, die an einen geschlossenen Port gerichtet ist, wird wie bei TCP mit einem RST-Paket beantwortet. Da auch dies bei UDP die Grundlage für Port-Scans ist, wird bei jeder dieser Anforderungen ein HoNA-Event generiert.

AUSE_UDP_BAD_CHECK

Dieses HoNA-Event signalisiert das Eintreffen eines Paketes mit einer inkorrekten Prüfsumme im UDP-Header.

3.4 Regulierung des Aufzeichnungsumfangs

Im vorangegangenen Abschnitt wurden alle Audit-Events vorgestellt, die mittels der hier beschriebenen HoNA-Implementierung protokolliert werden können. Bereits bei der Auswahl der Events wurde versucht, die Menge der anfallenden Audit-Daten so weit wie möglich zu beschränken. Hierbei handelt es sich jedoch um eine statische Analyse ohne die Möglichkeit die Einflüsse der verschiedenen Netzwerkumgebungen mit einbeziehen zu können. Nicht jedes Audit-Event ist für jede Netzwerkumgebung notwendig bzw. sinnvoll. Aus diesem Grund muss eine Möglichkeit zur dynamischen Selektion der Audit-Events geschaffen werden.

Im Folgenden werden die Selektionsmöglichkeiten von HoNA im Einzelnen vorgestellt. Hierzu erfolgt zunächst eine Erläuterung der Selektionskriterien und anschließenden die Beschreibung der Umsetzung mit Hilfe von Filterregeln.

3.4.1 Selektionskriterien

Klassische Netzwerk-Monitore bzw. Sniffer unterstützen meist eine Filterung der protokollierten Daten anhand des gesamten Paketinhaltes. So kann auch in den reinen Nutzdaten nach speziellen Schlüsselwörtern gesucht werden. Da die Funktionalität eines BS-Audits die Performance des Gesamtsystems nicht zu stark beeinflussen sollte, sind derartige Untersuchungen der Nutzdaten eines Paketes nicht möglich.

Die Selektionsmöglichkeiten eines BS-Audit beziehen sich idealerweise auf dessen Protokollierungsziele. Davon eignen sich insbesondere die folgenden drei Ziele:

- Wer hat das Audit-Event initiiert?
- Wie wurde das Ereignis ausgelöst?
- Welche Ressource war betroffen?

Die verbleibenden beiden Protokollierungsziele eines Audit-Systems, Wo und Wann die Aktion ausgeführt wurde, werden entweder implizit beantwortet (Wo) oder können durch die Steuerung des Audit-Systems eingehalten (Wann) werden.

Wer hat das Audit-Event initiiert?

Hierzu muss zunächst die Beschreibung des Verursachers geklärt werden. Darüber hinaus muss es möglich sein, anhand dieser Beschreibung zwischen vertrauenswürdigen und sicherheitsgefährdenden Verursachern unterscheiden zu können. Bei Netzwerk-Events können die Verursacher nur durch die Quelladressen beschrieben werden. Als Quelladresse gilt die MAC-Adresse, die IP-Adresse und der Quellport des eingetroffenen Paketes.

Die MAC-Adresse gilt nur innerhalb eines Netzsegmentes, so dass die Anzahl der möglichen Rechner bereits durch die Architektur des Netzwerkes beschränkt ist. Darüber hinaus werden bisher nur zwei Audit-Events unterhalb der IP-Ebene generiert, so dass die MAC-Adresse bei nahezu allen Audit-Events auf die IP-Adresse abgebildet werden kann. Sicherlich könnte man an dieser Stelle anmerken, dass mittels IP-Spoofing eine mögliche Protokollierung einer Aktion umgangen werden kann. Dies ist im Wesentlichen richtig, jedoch ist die Manipulierung der MAC-Adresse einfacher als das Verschicken von gefälschten IP-Paketen, so dass auch ein MAC-Filter mittels Spoofing umgangen werden kann. Spoofing ist ein allgemeines Sicherheitsproblem der Host-basierte Netzwerkzugriffskontrolle, welches auch an dieser Stelle nicht gelöst werden kann.

Die Möglichkeit, Filteregeln auf Portnummern zu beziehen, beschränkt sich auf Protokolle mit Socket-Adressen. Dies wären, entsprechend der im Abschnitt 3.3.2 vorgestellten Übersicht, die beiden Audit-Events `AUE_TCP_RCV` und `AUE_UDP_RCV`. Damit ist die Anzahl der möglichen Audit-Events, bei denen die Portnummer berücksichtigt wird, bereits durch das Design des Audit-Systems stark eingeschränkt. Da eine Selektion anhand der Portnummer die Performance des Systems nicht wesentlich beeinflusst, wurde diese Möglichkeit in dem hier vorgestellten Konzept implementiert. Es bleibt dem Nutzer überlassen, ob er davon Gebrauch macht oder nicht.

Wie wurde das Ereignis auslöst?

Beim klassischen BS-Audit wird die Frage nach dem wie, meist durch das Ereignis selbst, den finalen Ausführungsstatus der Aktion, den verwendeten Argumenten und dem aktuellen Environment beantwortet. Von diesen Kriterien lässt sich nur das Ereignis und der finale Ausführungsstatus numerisch und damit leicht auswertbar darstellen, so dass die Selektion, wenn überhaupt vorhanden, meist auf diese beiden Kriterien beschränkt wird.

Bei HoNA ist der finale Ausführungsstatus nicht auswertbar, da die protokollierten Ereignisse nicht auf Aktionen beruhen, die ausgeführte werden müssen. Ein HoNA-Event resultieren vielmehr aus dem finalen Status einer Überprüfung oder eines Ereignisses. Damit ist der Ausführungsstatus bereits im Event enthalten.

Den Inhalt der Protokoll-Header und die Konfiguration bzw. den Status des Netzwerk-Stacks kann man als die verwendeten Argumente bzw. das aktuelle Environment auffassen. Da der Inhalt der Header mitunter sehr komplex und von Protokoll zu Protokoll verschieden ist, lassen sich hier nur wenige bis gar keine gemeinsamen Selektionskriterien finden. Ähnliches gilt für das Environment, so dass eine Selektion anhand der Argumente und des Environments nicht effektiv durchführbar ist.

Aufgrund dessen erscheint die Selektion der Audit-Events hinsichtlich des „Wie“ nur anhand der Major- und der Minor-Event-ID für sinnvoll und wurde in dem hier vorgestellten Konzept auch auf diese beiden Kriterien beschränkt.

Welche Ressource war betroffen?

Während beim BS-Audit die Ressource durch den Namen, die Inode und die Geräte-Nummer beschrieben werden kann, lassen sich beim Netzwerk-Audit die Ressourcen nur in Form von Zieladressen beschreiben. Wie bei den Quelleadressen lassen sich diese Adressen in die MAC-Adresse, die IP-Adresse und die Portnummer einteilen.

Die MAC- und die IP-Adresse ist durch die Adresse des Zielrechners fest vorgegeben. War das Paket nicht direkt an den Rechner adressiert, auf dem das Event aufgetreten ist, bspw. wenn dieser als Router oder Firewall fungiert, so hat das Paket nichtsdestotrotz eine Aktion auf diesem ausgelöst. Demnach ist der Firewall oder der Router und nicht der Zielrechner des Paketes die betroffene Ressource, so dass eine Selektion anhand der Zieladresse des Paketes nicht korrekt wäre. Im HoNA-Konzept wird aus diesem Grund auf eine Selektion anhand er Zieladresse des Paketes verzichtet.

Die Portnummer bestimmt den betroffenen Prozess, welcher auf dem Rechner als eine Ressource aufgefasst werden kann. Eine Selektion hinsichtlich der Zielportnummer erscheint in jedem Fall sinnvoll, da hiermit Angriffe geben Applikationen, die auf dem Zielsystem ausgeführt werden, gezielt protokolliert werden können.

3.4.2 Filterregeln von HoNA

Das Konzept zur Selektion der zu speichernden Ereignisse wurde von der ACL-Implementierung von LOSA abgeleitet (*ACL*, *Access Control List*), so dass die Regeln für HoNA-Events ebenfalls mittels sogenannter ACL-Regel gefiltert werden. Jede Regel beschreibt, ob ein Ereignis, welches durch die Major-Event-ID, die Minor-Event-ID, die IP-Adresse des Quellrechners, den Quellport- und den Zielportnummer beschrieben wird, aufgezeichnet werden soll oder nicht. Beim Eintreten eines Events wird der Inhalt des verursachenden Paketes mit den Einträgen der Regelbasis verglichen.

Die ACL-Regeln der LOSA-Implementierung werden mittels der Audit-Bibliothek und des Steuerungsprogramms in das Audit-Modul im BS-Kern übertragen und dienen dort der Regulierung der Aufzeichnung von Ressourcen-bezogenen Audit-Events. Für die Umsetzung einer effizienten Selektion anhand der im vorangegangenen Abschnitt beschriebenen Kriterien, wurde für HoNA-Events eine ähnliche Struktur in das Audit-Konzept integriert.

Dieser Abschnitt beschreibt die verfügbaren Regeln zur Konfiguration der HoNA-Regelbasis. Dabei werden die im Abschnitt 3.4.1 beschriebenen Kriterien noch einmal im praktischen Kontext erläutert.

3.4.2.1 Konfigurationsregeln

Die in diesem Abschnitt vorgestellten Konfigurationsregeln werden vom Steuerungsprogramm, welches ebenfalls im Rahmen diese Arbeit erweitert wurde, verwendet. Sie sind nicht fest mit dem Audit-Modul verbunden und können durch eine andere Implementierung, welche die Schnittstellen der Audit-Bibliothek bzw. des Kernel-Moduls einhält, ersetzt werden.

Eine ACL-Regel zur Regulierung des Aufzeichnungsumfangs von HoNA hat folgende Struktur:

```
setnetrule EVT:SUBEVT ACTION IP_ADDR PORTS
```

wobei die einzelnen Elemente folgende Bedeutung haben:

`EVT:`

Ist das Major-Event, auf welches sich die Regel bezieht. Wird für die Event-ID das Schlüsselwort `any` verwendet, bezieht sich die Regel auf alle HoNA-Events. Außerdem können mehrere Event-IDs durch

ein Komma getrennt in einer Regel angegeben werden. Mittels eines Bindestriches wird ein Bereich von Event-IDs spezifiziert. Bspw. bezieht sich die Regel:

```
setnetrule 80,82-84:any on inet (141.43.3.0-141.43.23.0)/24 any
```

auf die Events 80 und 82 bis 84.

SUBEVT:

Ist das Minor-Event, auf welches sich die Regel beziehen soll. Die Angabe des Minor-Events kann wie beim Major-Event mittels des Schlüsselwortes `any`, eines Kommas bzw. eines Bindestriches auf mehrere Events bezogen werden.

ACTION:

Beschreibt die eigentliche Aktion der Regel. So lässt sich die Protokollierung von Events gezielt ein- und ausschalten. Hierbei sind die folgenden beiden Schlüsselwörter zulässig:

`on` : das Event wird protokollieren und
`off` : das Event wird nicht protokollieren.

IP_ADDR

Dient der Selektion der HoNA-Events anhand der Quelleadresse des Paketes. Eine IP-Adresse einer Filterregel besteht aus der Angabe der Adressfamilie, der IP-Adresse selbst und dem CIDR-Block (*CIDR, Classless Inter Domain Routing*). Die CIDR-Blockmaske identifiziert die Netzmaske, welche auf die IP-Adresse der Regel angewandt wird, um die Adresse des eingetroffenen Paketes zu testen. Bei der Verwendung von IPv4-Adressen identifiziert die CIDR-Blockmaske 24 ein Klasse C Netzwerk, 16 ein Klasse B Netzwerk und 32 identifiziert eine Rechneradresse.

Bei der Angabe der IP-Adresse können die Adressfamilien IPv4 und IPv6 verwendet werden. Die in der Regel verwendete Adressfamilie muss mittels der Schlüsselwörter `inet` (IPv4) bzw. `inet6` (IPv6) vor der Angabe der Adresse festgelegt werden. Die genaue Darstellung von IPv6-Adressen ist in [9] beschrieben. Die Angabe der CIDR-Blockmaske ist bei beiden IP-Versionen zwingend erforderlich.

Um die Anzahl der Regeln zu minimieren, ist es möglich in einer Regel mehrere Adressen anzugeben. Diese müssen mit einem Komma getrennt werden. Außerdem ist es möglich, mittels eines Bindestriches Bereiche zu definieren. So aktiviert bspw. die obige Regel die Protokollierung aller `TCP_RCV`-Events aller Rechner aus den Netzen 141.43.3.0 bis 141.43.23.0.

Die Verwendung des Schlüsselwortes `any` bei der Angabe der Adresse ist gleichzusetzen mit der Angabe der Adresse 0.0.0.0/0. Dies ist sowohl für IPv4- als auch für IPv6-Adressen möglich.

PORTS

Die Angaben von Portnummern ist nur bei den Events `AUE_TCP_RCV` und `AUE_UDP_RCV` sinnvoll, da nur diese Protokolle eine Adressierung über Portnummer durchführen. Bei allen anderen Events wird die Portnummer ignoriert.

Zur Beschreibung der betroffenen Ressource bei den Protokollen TCP und UDP besteht jede Portangabe aus einer Ziel- und einer Quellportnummer. Dabei wird die Richtung mittels des Richtungsoperators ‘`->`‘ festgelegt. So beschreibt

```
5623 -> 23
```

ein Paket vom Port 5623 zum Port 23. Außerdem kann mittels des Richtungsoperators ‚<>‘ eine bidirektionale Verwendung der Portangabe veranlasst werden. Wie bei den übrigen Angaben kann auch für die Angabe der Portnummern eine Aufzählung und der Bereichsoperator verwendet werden.

3.4.2.2 Ordnung doppelter Regeln

Durch die Angabe von Bereichen und durch die Verwendung des Action-Feldes können Regeln definiert werden, die sich wiedersprechen. Um dem entgegenzuwirken, muss eine Ordnung festgelegt werden, anhand welcher entschieden werden kann, welche Regel für das aufgetretene Ereignis zutreffend ist bzw. angewendet werden soll.

IP-Adresse

Hierbei wird zunächst die Anzahl der möglichen Rechner im Netzwerk zu Grunde gelegt. Die Ordnung ist gleichbedeutend mit der Ordnung der CIDR-Blöcke. D.h. werden die Regeln:

```
setnetrule any:any on inet any any

setnetrule any:any off inet 141.43.3.0/24 any
```

eingefügt, so wird für ein Paket von einem Rechner mit der IPv4-Adresse 141.43.3.165 kein Audit-Record generiert da die zweite Regel den kleineren Bereich an Rechnern abdeckt. Er beinhaltet lediglich 256 verschiedene Rechner, während sich die erste Regel auf alle Rechner bezieht.

Für IPv6-Adressen wird die gleiche Ordnung verwendet, nur dass die Anzahl der möglichen Subnetze von 33 auf 129 Ebenen ansteigt. ACL-Regeln für IPv4- und IPv6-Adressen werden getrennt abgelegt und verwaltet, zwischen ihnen besteht keine Relation.

Port-Adresse

Existiert zudem die Regel:

```
setnetrule 83:any on inet 141.43.3.0/24 any <> 1024,
```

wäre eine Ordnung über die Anzahl, der in einer Regel abgedeckten Rechner, nicht mehr eindeutig. In einem solchen Fall wird die Größe des Bereiches der Portnummern hinzugezogen. Dabei berechnet sich die Größe des Bereichs aus der Anzahl der möglichen Portnummern. Bspw. würde die Regel:

```
setnetrule 83:any on inet 141.43.3.0/24 1024 -> any,
```

der Regel:

```
setnetrule 83:any on inet 141.43.3.0/24 any <> 1024,
```

vorgezogen werden, da sie nur 65536 Portnummern abdeckt, während mit der zweiten Regel 2 * 65536 Ports adressiert werden können.

Event-ID

Kommt nach der Selektion mit Hilfe der IP-Adressen und der Portnummern immer noch mehr als eine Regel in Frage, wird die Anzahl der möglichen Event-IDs hinzugezogen. Diese berechnet sich aus:

mögliche Major-Events x mögliche Minor-Events

und liefert in jedem Fall ein eindeutiges Ergebnis.

Kapitel 4

Implementierung

Die Implementierung von HoNA für LINUX lässt sich in zwei Komponenten unterteilen. Dies ist zum einen die Erweiterung des LINUX BS-Kerns, im Folgenden auch als Audit-Modul bezeichnet, und die Erstellung von Nutzerapplikationen zur Steuerung des Audit-Moduls im Kernel. In diesem Abschnitt wird die Implementierung beider Komponenten erläutert. Auf den Trail-Viewer, der eigentlich nicht zur HoNA-Implementierung gehört, wird nur kurz eingegangen.

4.1 Kernel-Erweiterung

Die Kernel-Erweiterung basiert zum Großteil auf der Implementierung des Audit-Moduls von LOSA. Dieses stellte bereits ein Grundgerüst für das Puffermanagement und für das Sicherheitskonzept bereit. Eine Erweiterung des Sicherheitskonzeptes waren für die Integration von HoNA nicht notwendig und ist demzufolge nicht Bestandteil dieser Arbeit. Das Puffermanagement konnte nicht ohne weiteres für die Verwaltung der HoNA-Records verwendet werden und wurde fast gänzlich neu gestaltet bzw. neu implementiert. In diesem Abschnitt erfolgt, neben der Beschreibung der Erweiterung des Puffermanagements von LOSA, die Erläuterung des Interfaces zum Netzwerk-Stack, das Konzept zur Sicherstellung der Vollständigkeit des Audit-Trails und die Verwaltung der ACL-Regeln.

4.1.1 Schnittstelle zum Linux-Netzwerk-Stack

Bereits im Abschnitt 3.2.2 wurde die Schnittstelle von HoNA zum Netzwerk-Stack von LINUX kurz erläutert. Im Wesentlichen besteht sie aus zwei Funktionsaufrufen, die innerhalb einer `#define`-Präprozessor-Direktive gekapselt sind. Für ein TCP-Event würde dies bspw. wie folgt aussehen:

```
#ifdef CONFIG_AUDIT
      if (audit_net_doit(AUE_TCP_RCV, AUSE_TCP_BAD_SIZE, skb,
                         AUDIT_GET_FAM(tp)))
            audit_tcp_rcv(skb, AUSE_TCP_BAD_SIZE, sk);
#endif
```

CONFIG_AUDIT

Die `#define`-Präprozessor-Direktive kapselt die gesamte Funktionalität des Audit-Moduls und wird beim Konfigurieren des Kernels gesetzt. Sollte in der Konfiguration die Unterstützung des Audit-Moduls nicht aktiviert werden, entspricht der Kernel im Wesentlichen dem Standard-Kernel, wie er aus dem Internet heruntergeladen werden kann.

audit_net_doit

Die Funktion `audit_net_doit(...)` ist für die Durchsetzung der Filtermechanismen verantwortlich und innerhalb der Datei `ctl.c` implementiert. Die ersten beiden Parameter der Funktion sind das Major- und das Minor-Event. Die Entscheidung, ob das Audit-Event aufgezeichnet werden soll, wird zusätzlich mit Hilfe der Absenderadresse und der Portnummern getroffen. Hierzu muss der Funktion ein Zeiger auf das aktuelle Paket übergeben werden. Das letzte Argument der Funktion spezifiziert die Protokollfamilie des verwendeten Internet-Protokolls. Bisher werden die

beiden Familien `AF_INET` und `AF_INET6` unterstützt. Die Angabe der Protokollfamilie ist notwendig, da für jede Familie eine separate Regelbasis vorhanden ist.

Audit-Funktion: `audit_tcp_rcv(...)`

Die Implementierung der Audit-Funktionen von HoNA ist in den Dateien `audit_tcpip_l2.c`, `audit_tcpip_l3.c` und `audit_tcpip_l4.c` im Verzeichnis `audit` des *Kernel-Trees* zu finden. Das Verzeichnis `audit` enthält alle Dateien des Audit-Moduls und wird als separates Objekt in den Kernel eingebunden. Dies ist eine der Grundlagen für eine mögliche Implementierung als dynamisch ladbares Kernel-Modul.

Die Anzahl und die Bedeutung der Parameter der Audit-Funktionen ist abhängig von der jeweiligen Funktion. Jedem Major-Event ist eine bestimmte Funktion zugeordnet. Sie kann aber mehrere Minor-Events verarbeiten. Demzufolge erwartet sie immer einen Zeiger auf das betroffene Paket und das Minor-Event. Eine Ausnahme bildet die Funktion `audit_tcp_to(...)`. Sie dient der Protokollierung von TCP-Timeouts. Da diese nicht bei der Verarbeitung eines Paketes aufgerufen wird sondern beim Eintreten eines Timeouts eines TCP-Sockets, steht kein Netzwerkpaket zur Verfügung. Die notwendigen Daten werden in diesem Fall aus den betroffenen Socket entnommen.

4.1.2 Puffermanagement von HoNA

Da es aus Performancegründen unmöglich ist, jeden generierten Audit-Record sofort auf ein permanentes Medium zu speichern, werden die Audit-Records für einen gewissen Zeitraum im Kernel-Speicher abgelegt. In der Implementierung von LOSA wurde nur eine Record-Typ unterstützt, demzufolge war der Aufbau und das Management des Puffers auch auf diesen Typ zugeschnitten. Zwar existierten bereits Einträge für die Unterscheidung des Record-Typs, jedoch wurden diese nicht genutzt und waren für die Verwaltung von mehreren Record-Typen nicht ausreichend. In dem hier vorgestellten Konzept wurde das Puffermanagement von LOSA dahingehend erweitert, dass es weitestgehend unabhängig vom verwendeten Record-Typ ist und das Einbringen eines neuen Record-Typs ohne Änderungen am eigentlichen Puffermanagement möglich ist.

4.1.2.1 Konfigurierung der Audit-Record-Puffer

Der Nutzer entscheidet, wie viele Audit-Records im Speicher gehalten werden. Der Audit-Dämon wird beim Erreichen einer *Watermark*[16] in einem Aufruf versuchen, alle im Audit-Puffer gespeicherten Audit-Records auf ein permanentes Speichermedium zu übertragen. Die Entscheidung hinsichtlich der Größe des Record-Puffers wird durch drei Faktoren beeinflusst:

1. Umso kleiner der Puffer desto schlechter ist die Gesamtperformance des Systems. Mit unter müssen im **`normal`**-Modus einzelne Events verworfen werden, da der Audit-Dämon nicht rechtzeitig die Speicherung ausführen konnte.

2. Der Inhalt der Audit-Records wird ständig verändert, so dass der genutzte Speicherbereich nicht ausgelagert werden kann und demzufolge physikalisch vorhanden sein sollte. Der benötige Speicherbereich wird vom Nutzerspeicher abgezogen und steht damit allen anderen Anwendungen nicht mehr zur Verfügung.

3. Sollte das System ausfallen, ohne dass der Audit-Dämon die verbleibenden Audit-Records speichern konnte, sind diese Daten verloren. Im Falle eines Angriffs ist die Anzahl der verlorenen Audit-Events mitunter entscheidend für die Reproduzierbarkeit der Vorgänge.

[16] Die Watermark entspricht 90 Prozent der Puffergröße. D.h. sind nur noch 10 Prozent der Audit-Records im Puffer frei, wird der Audit-Dämon aufgeweckt.

In der Praxis hat sich eine Audit-Puffer mit 500 Records bewährt. Selbst bei großer Last muss nur ein kleiner Teil an Audit-Events verworfen werden. Im **`secure`**-Modus werden auch bei kleineren Puffern keine Audit-Events verworfen, jedoch kann die Gesamtperformance des Systems nachhaltig beeinträchtigt werden.

4.1.2.2 Aufbau des Record-Puffers

Um einen flexiblen Aufbau zu erreichen, wurde ein Audit-Record, wie in Abbildung 12 dargestellt, in drei Elemente unterteilt. Das erste Element eines Audit-Records wird durch die Struktur `audit_record` beschrieben und enthält einen `void`-Zeiger auf die eigentlichen Daten des Audit-Records. Diese sind in den Typ-spezifischen Puffern gespeichert. Für Netzwerk-Events sind dessen Objekte vom Typ `audit_net_record`. Sollten die Einträge des Typ-spezifischen Puffers nicht ausreichen, können zusätzliche Daten in einem sogenannten dynamischen Puffer eingetragen werden.

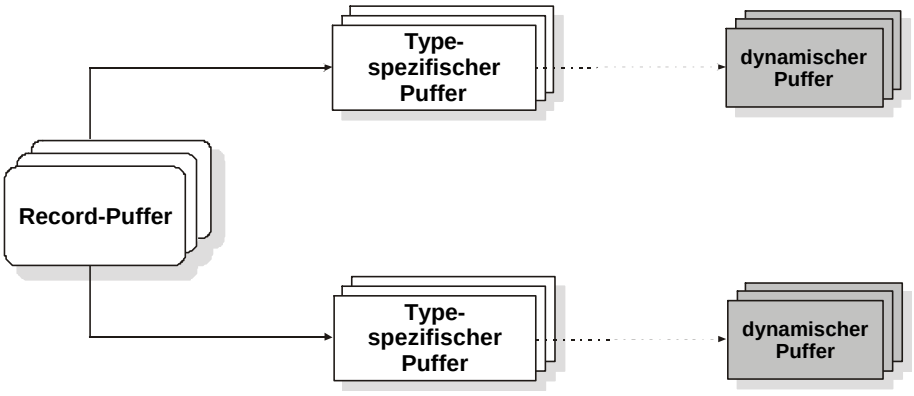

Abbildung 12 : Audit-Record-Puffer

Für jeden Record-Typ muss ein Typ-spezifischer Puffer definiert werden. Diesem ist es wiederum freigestellt, ob er zusätzlich einen dynamischen Puffer nutzt. Die Anzahl der verwendeten dynamischen Puffer ist damit abhängig von der Implementierung und der Konfiguration des Typ-spezifischen Puffers. In der derzeitigen Implementierung nutzt jeder Typ-spezifische Puffer einen dynamischen Puffer, dessen Verwendung in der Konfiguration des Audit-Dämons explizit aktiviert werden muss.

Audit-Record Puffer

Der Audit-Record-Puffer wird nur einmal angelegt. Er bildet die oberste Schicht des Puffermanagements in BS-Kern. Alle Typ-unabhängigen Funktionen, wie bspw. die Funktion `audit_record2char(...)` zum Speichern eines Audit-Records, arbeiten auf dessen Pufferelementen. Er wird durch die Datenstruktur `audit_record` beschrieben, welche folgende Elemente enthält:

`type`

Gibt den Typ des Records an und ist eine Zahl zwischen Null und `AU_REC_TYPES – 1` Mittels dieses Eintrages werden die Funktionen zum Bearbeiten des zugehörigen Typ-spezifischen Puffers adressiert.

`state`

Beschreibt den aktuellen Zustand des Records. Mögliche Werte sind `AU_FREE`, `AU_USED`, `AU_UNUSED` und `AU_FULL`. Ein Rekord hat den Status:

- `AU_FREE`: wenn aktuell keine Daten in ihm abgelegt sind,

- AU_USED: wenn er gerade von einer Audit-Funktion gefüllt wird,
- AU_UNUSED: wenn er infolge eines Audit-Events gefüllt wurde, aber nicht gespeichert werden soll und
- AU_FULL: wenn er gefüllt wurde und gespeichert werden kann.

data
Verweist auf ein Element des Typ-spezifischen Puffers. Der Typ des Puffers wird mittels des Eintrages type beschrieben.

next, prev
Die Zeiger next und prev dienen der Implementierung des doppelt-verketteten Ring-Puffers.

Darüber hinaus enthält die Struktur noch die Typ-abhängigen Elemente date, pid, event und hostname des Audit-Record. Eine Beschreibung der Elemente erfolgte bereits im Abschnitt 3.3.1.

Typ-spezifische Puffer

Jeder Record-Typ stellt eine eigene Struktur zur Speicherung der Audit-Events bereit. Um den Aufbau und die Verwaltung der Typ-spezifischen Puffer unabhängig von der Implementierung des Puffermanagements zu gestallten, muss für jeden Record-Typ eine Pufferbeschreibung gefüllt werden. Im Folgenden wird die Pufferbeschreibung des HoNA-Record-Puffers vorgestellt, die Implementierung des Host-Record-Puffer ist ähnlich.

Pufferbeschreibung

Die Pufferbeschreibung wird in dem abstrakten Datentyp (ADT) audit_buffer_info abgelegt. Er enthält die Attribute und Methoden für die Verwaltung des Puffers. Die einzelnen Pufferbeschreibungen werden in dem globalen Array au_buf_infos abgelegt, welches mittels des Defines AUDIT_BUFFER_INIT(...) initialisiert wird.

In der Datenstruktur audit_buffer_info sind die folgenden Elemente zu füllen:

state
Der Eintrag state beschreibt den aktuellen Zustand des Puffers. Bisher werden die folgenden Zustände unterstützt:

- AU_UNDEF: das Objekt wurde noch nicht initialisiert und darf nicht verwendet werden,
- AU_FREE: das Objekt wurde initialisiert und enthält noch freie Records,
- AU_SEND_ALRM: der Puffer des Objektes wurde bis zur Watermark gefüllt und es konnte erfolgreich ein Signal an den Audit-Dämon geschickt werden,
- AU_FULL: alle Elemente des Puffers sind mit Daten gefüllt, so dass keine zusätzlichen Audit-Records mehr in diesem Puffer abgelegt werden können.

type
Der Eintrag type enthält den Typ des Puffers in Form eines numerischen Wertes. Der Wert ist zugleich der Index des Puffers für das globale Array au_buf_infos.

rec_size
Enthält die Größe eines Audit-Records.

buf_size
: Enthält die Anzahl der Elemente im Puffer.

buf_wm
: Speichert die Watermark des Puffers. Sind nur noch weniger als `buf_wm` Elemente des Puffers frei, wird ein `ALRM`-Signal an den Audit-Dämon gesendet und der aktuelle Zustand des Puffers auf `AU_SEND_ALRM` gesetzt.

free_recs
: Enthält die Anzahl der freien Pufferelemente.

dropped
: Speichert die Anzahl der verworfenen Record-Anfragen seit dem Start des Systems. Eine Record-Anfrage muss verworfen werden, wenn der Puffer keine freien Records mehr enthält und das Anfordern eines neuen Records fehlschlägt.

Die folgenden beiden Elemente dienen der Adressierung des Puffers und werden nur innerhalb dessen Implementierung verwendet.

buf_addr
: Enthält die Adresse des ersten Pufferelementes und dient dem Löschen des Puffers.

cur_rec
: Enthält die Adresse des nächsten freien oder die des ersten vollen Elementes, je nachdem ob der Puffer komplett gefüllt ist oder noch freie Elemente enthält.

Für alle Operationen auf den Pufferelementen beinhaltet der ADT Zeiger entsprechende Funktionen. Die Implementierung der Funktionen ist Puffer-spezifisch und nur durch die bereitzustellende Funktionalität beschränkt.

Die folgenden vier Funktionen dienen der Arbeit auf dem Puffer selbst.

init(struct audit_buffer_info *, unsigned int)
: Mittels der Funktion `init(...)` wird der Puffer initialisiert. Nach dem Aufruf der Funktion sollte der Puffer soweit eingerichtet sein, dass er Daten aufnehmen kann.

destroy(struct audit_buffer_info *)
: Die Funktion `destroy(...)` löscht den gesamten Puffer. Da nach dem Aufruf keine weitere Funktion mehr aufgerufen wird, sollte die Funktion den gesamten angeforderten Speicher freigeben und die Pufferbeschreibung auf die initialen Werte zurücksetzen.

reduse(struct audit_buffer_info *)
: Für Rekonfigurationsoperationen kann es notwendig sein, den ganzen Puffer zu löschen und anschließend einen neuen anzulegen, damit dem System für diese Operation ausreichend Speicher zur Verfügung steht. Hierzu kann mittels der Funktion `reduse(...)` der alte Puffer auf die minimale Größe reduziert werden. Dabei werden alle nicht genutzten Pufferelemente gelöscht und die verbleibenden Elemente neu verkettet.

ctl(struct audit_buffer_info *, unsigned long, unsigned long)
: Die `ctl(...)`-Funktion ist die allgemeine Managementschnittstelle zum Puffer. Hiermit können die Einstellungen der Watermark und des dynamischen Zusatzpuffers verändert werden.

Die folgenden Funktionen beinhalten die Operationen zum Arbeiten mit den Pufferelementen. Vor dem Aufruf einer dieser Funktionen muss sichergestellt sein, dass der Puffer bereits initialisiert wurde.

`get_free(struct audit_buffer_info *)`

Die Funktion `get_free(...)` liefert das nächste freie Pufferelement. Sind keine freien Elemente mehr vorhanden, gibt die Funktion einen Null-Zeiger zurück. Sollte bei der Record-Anfrage die Watermark des Puffers erreicht werden, sendet die Funktion ein Signal an den Audit-Dämon.

`get_new(void)`

Sollten im Puffer keine freien Elemente mehr zu Verfügung stehen, kann mittels der Funktion `get_new(...)` ein neues Element angefordert werden. Das neue Element ist nicht Bestandteil des Typ-spezifischen Puffers und muss von einer anderen Instanz verwaltet werden.

`rec2chr(struct audit_buffer_info *, char *, void **)`

Die Funktion `rec2chr(...)` erwartet drei Argumente. Das erste Argument ist ein Zeiger auf die Pufferbeschreibung, das zweite Argument ein Zeiger auf den String-Puffer und das dritte Argument enthält die Adresse des zu speichernden Audit-Records. Der String-Puffer sollte mindesten eine Größe von `rec_size + 128 kBytes` haben.

Die Funktion gibt als Ergebnis die Anzahl der kopierten Bytes zurück, setzt den Zustand des bearbeiteten Records auf `AU_FREE` und aktualisiert die Daten der Pufferbeschreibung. Ist der übergebene Record nicht Bestandteil des Puffers, sondern zuvor mittels der Funktion `get_new(...)` angefordert worden, wird dessen Speicher freigegebenen und die Zeiger auf Null gesetzt.

`free_rec(struct audit_buffer_info *, void **)`

Die Funktion `free_rec(...)` gibt das übergebene Pufferelement frei und verwirft dessen Daten. Für ein Element des Puffers wird lediglich dessen Zustand auf `AU_FREE` gesetzt und die Daten der Pufferbeschreibung aktualisiert. Wurde der übergebene Puffer mittels der Funktion `get_new(...)`angefordert, wird der Speicher des Elementes freigegeben und der Zeiger auf Null gesetzt.

Sollten keine freien Pufferelemente mehr zur Verfügung stehen, muss die Generierung von Audit-Events solange unterbrochen werden, bis der Inhalt des Puffers gespeichert werden konnte. Für die Durchsetzung dieser Unterbrechung sind die folgenden beiden Funktionen zu implementieren.

`buf_suspend(void)`

Wird immer dann aufgerufen, wenn kein Pufferelement mehr verfügbar war und das Audit-Event nicht verworfen werden soll. Eine geeignete Strategie für die Durchsetzung der Operation ist abhängig vom aktuellen Aufführungskontext. Bisher gibt es zwei unterstützte Kontexte.

1. Die Generierung des Audit-Events erfolgt innerhalb des Kontextes eines Prozesses. Dies ist bei allen Host-Events möglich, da diese bei der Ausführung eines Systemsrufs protokolliert werden.
2. Das Audit-Event wird von einer Funktion des Netzwerk-Stacks generiert. Der Netzwerk-Stack wird nicht innerhalb des Kontextes eines Prozesses abgearbeitet sondern innerhalb eines sogenannten *Bottom-Halfs*[17].

[17] *Bottom Halfs* enthalten die vergleichsweise zeitaufwendigen Operationen einer Interrupt-Behandlung. Für jeden Interrupt muss eine sogenannte Interrupt-Service-Routine (ISR) definiert werden. Wurde ein Interrupt aufgelöst, wird seine zugehörige ISR unmittelbar aufgerufen und die aktuell ausgeführte Operation suspendiert. Um die Suspendierung einer Operation in einem vertretbaren Rahmen zu halten, führt die ISR nur die notwendigsten Operationen aus und signalisiert dem System, dass noch weitere Arbeiten verbleiben. Beim nächsten Prozesswechsel werden die angesammelten Bottom-Halfs ausgeführt, bevor die CPU einem neuen Prozess zugeordnet wird.

Was genau in diesen Fällen passiert, wird im Abschnitt 4.1.3 erläutert.

buf_wake_up(void)

Die Funktion `buf_wake_up(...)` ist das Gegenstück zu `buf_suspend(...)` und wird aufgerufen, wenn wieder davon ausgegangen werden kann, dass freie Records zur Verfügung stehen. Dies ist bspw. immer nach dem Aufruf der Funktion `audit_service(...)` der Fall (siehe Abschnitt 4.1.2.5).

Audit-Net-Puffer

Für HoNA-Events werden die Elemente des HoNA-Record-Puffers gefüllt. In der aktuellen Implementierung kann dieser Puffer einmal angelegt werden und wird dynamisch mit dem Audit-Record-Puffer verknüpft. Die Anzahl der Elemente des Puffers kann durch den Nutzer festgelegt werden. Die Struktur ist durch den ADT `audit_net_record` vorgegebenen. Die Datenstruktur enthält folgende Elemente:

protocol

Das Element `protocol` enthält das verwendete Protokoll der Internet Schicht. Für Ethernet-Pakete ist dieses identisch mit dem `protocol`-Element des Ethernet-Headers.

dev_type

Beschreibt den Typ des Gerätes, auf dem das Paket eingetroffen ist. Bisher werden lediglich Ethernet-Geräte unterstützt. Bei allen anderen Gerätetypen wird der Eintrag auf `ARPHRD_VOID` gesetzt. Das `mac`-Element des Host-an-Netz Schicht-Containers enthält in diesem Fall keine Daten.

mac_prot_size

Enthält die Größe des Host-an-Netz Schicht-Containers. Sind im Container keine Daten gespeichert, wird der Eintrag auf Null gesetzt.

nh_prot_size

Siehe `mac_prot_size` für den Container der Internet Schicht.

n_proto_size

Siehe `mac_prot_size` für den Container der Transportschicht.

namelen

Enthält die Länge des Gerätenamens.

Bis zum Eintrag `namelen` haben alle Elemente der Struktur eine feste Größe. Beim Speichern und beim Laden werden diese Einträge mit einer Kopieroperation in bzw. aus dem Trail übertragen. Die genaue Anzahl der zu kopierenden Bytes ist im Define `AU_NET_REC_FIX` vermerkt.

mac

Der Eintrag `mac` enthält den Container der unterstützen Protokolle der Schicht zwei des TCP/IP-Models. Bisher wird lediglich das Ethernet-Protokoll unterstützt. Das genaue Protokoll des Containers muss dem Element `dev_type` entnommen werden.

Bei einigen Audit-Events der Protokolle TCP und UDP kann es passieren, dass der Eintrag `mac` keine Daten enthält, obwohl das Paket auf einem Ethernet-Device eingetroffen ist. Die Funktion `tcp_v4_rcv(...)` löscht den Verweis auf das Gerät und verhindert damit eine Protokollierung der Daten der Schicht zwei.

nh

Die Header der Protokolle IPv4, IPv6 und ARP werden im Eintrag `nh` abgelegt. Das verwendetet Protokoll kann den Einträgen `protocol` oder `mac.ethernet.protocol`, so es sich um ein Ethernet-Paket handelt, entnommen werden.

h

Enthält die Protokoll-Header der Transportschicht sowie aller Protokolle, die das IP zur Übertragung ihrer Pakete verwenden. Bisher werden hier die Header der Protokolle TCP, UDP, ICMP, IGMP und IPIP abgelegt. Bei der Auswertung des Trails ist das Protokoll den Header-Informationen der Internet Schicht zu entnehmen.

name

Speichert den Namen des Gerätes, auf dem das Paket eingetroffen ist. Die Implementierung von LINUX unterstützt Gerätenamen bis zu einer Länge von 15 Zeichen.

Zur Verwaltung der einzelnen Pufferelemente enthält die Struktur noch die Einträge `next` und `state`. Der Eintrag `next` dient der Verkettung der Pufferelemente und `state` enthält den aktuellen Zustand des Elementes. Hierbei werden wiederum die Werte `AU_FREE`, `AU_USED`, `AU_UNUSED` und `AU_FULL` unterstützt.

Dynamischer Puffer

Der dynamische Zusatzpuffer ist aus dem Environment-Puffer der LOSA-Implementierung hervorgegangen. Dort wurde er lediglich für die Speicherung des `arg`- und des `env`-Argumentes bei der Protokollierung des Systemrufes `execve(...)` verwendet. Da diese beiden Argumente bis zu 128 kByte belegen dürfen, ist die Größe des Zusatzpuffers ebenfalls auf 128 kByte beschränkt.

Die Implementierung des Puffers gliedert sich in zwei Komponenten: eine Struktur zur Beschreibung des aktuellen Pufferzustandes und die Elemente des Puffers.

Pufferbeschreibung

Die Pufferbeschreibung beinhaltet die aktuellen Eigenschaften des Puffers. Dazu gehört die Größe der angeforderten Speicherbereiche (`page_size` und `page_order`), der Zustand des Puffers (`state`), die Watermark (`wm`) und die Anzahl der Elemente (`size`).

Die einfachste Art innerhalb des Kernels Speicher anzufordern, besteht mittels der Funktion `get_free_pages(...)`. Sie erwartet zwei Argumente. Das erste Argument gibt die Priorität vor (siehe [13]), das zweite Argument bestimmt die Ordnung des zu reservierenden Speicherblocks. Im LINUX-Kernel sind Ordnungen von null bis fünf zugelassen, damit lassen sich Blöcke der Größen 4 bis 128 kByte anfordern. Innerhalb eines Zusatzpuffers werden immer nur Speicherseiten einer Größe verwendet. Die zugehörige Ordnung und die daraus resultierende Größe wird in den Attributen `page_order` und `page_size` abgelegt.

Die Bedeutung der Watermark des Zusatzpuffers unterscheidet sich von der der anderen Puffer. Die Pufferelemente werden immer dynamisch angefordert und beim Speichern ihrer Inhalte werden sie wieder freigegebenen. Um den dadurch entstehenden Overhead in Grenzen zu halten, kann mittels der Watermark das Freigeben einzelner Elemente unterbunden werden. D.h. bei einer Watermark mit dem Wert `x`, werden beim Speichern alle bis auf `x` Elemente freigegeben. Die `x` Elemente werden lediglich als frei gekennzeichnet.

Elemente des Puffers

Die Elemente des Puffers werden durch den ADT `audit_dyn_buf_entry` vorgegeben. Die eigentlichen Daten werden in den mittels `get_free_pages(...)` angeforderten Speicherseiten abgelegt. Die Struktur enthält nur die notwendigen Zeiger auf diese Speicherseiten.

Die Implementierung ist im Wesentlichen von LOSA übernommen worden. Allerdings ist es jetzt möglich bis zu `AU_MAX_DYN_BUF_ENTRIES` (24) verschiedene Einträge in einem Objekt zu speichern. Die Anzahl der Einträge kann dynamisch verändert werden und ist somit nicht mehr auf die zwei Einträge `arg` und `env` beschränkt. Die Adressierung der einzelnen Einträge erfolgt über das in der Struktur gespeicherte Array `entry`. Um auch binäre Daten speichern zu können, wird die Größe der Einträge in dem Array `entry_len` innerhalb des Objektes abgelegt.

Die Erläuterung der übrigen Elemente und der Verkettung der einzelnen Speicherseiten ist der Beschreibung der LOSA-Implementierung zu entnehmen [6] und wird an diese Stelle ausgelassen.

4.1.2.3 Initialisierung des Puffers

Bevor der Audit-Dämon mit der Protokollierung des Systems beginnen kann, müssen die einzelnen Puffer eingerichtet werden. In der LOSA-Implementierung konnte dies, da nur ein Puffer verfügbar war, beim Aktivieren des BS-Audit erfolgen. In der jetzigen Implementierung müssen zunächst die Puffer mittels der Funktion `audit_init_buffer(...)` initialisiert werden. Diese erwartet einen Zeiger auf ein Array mit den Konfigurationsdaten der einzelnen Puffer.

Konnten die Konfigurationsdaten aus dem Nutzer- in der Kernel-Speicher übertragen werden, wird die Anzahl der benötigten Audit-Puffer-Elemente ermittelt. Dies ist die Summe aller Elemente der anzulegenden Typ-spezifischen Puffer. Anschließend können der Audit-Puffer und der Typ-Puffer initialisiert werden. Eine Verknüpfung der einzelnen Puffer erfolgt an dieser Stelle noch nicht. Die dynamischen Zusatzpuffer müssen anschließend separat initialisiert werden. Erfolgt dies nicht, werden alle Daten, die in diesen Puffer abgelegt werden, nicht aufgezeichnet.

Anschließend kann mittels der Funktion `audit_on(...)` das BS-Audit aktiviert werden.

4.1.2.4 Anforderung eines Audit-Records

Nachdem die Audit-Puffer initialisiert und das BS-Audit gestartet wurden, wird mit der Protokollierung der Systemaktivitäten begonnen. Die Audit-Funktionen fordern bei jedem zu protokollierenden Event mittels der Funktion `audit_get_free_record(...)` einen Audit-Record vom Puffermanagement an. Die Funktion erwartet zwei Parameter: den Typ des Audit-Records und die Priorität der Operation. Bisher werden die beiden Typen Host- und HoNA-Record unterstützt, die Anforderung eines anderen Typs erzeugt eine Syslog-Nachricht. Die Priorität gibt vor, wie zu verfahren ist, wenn kein Record des geforderten Typs mehr zur Verfügung steht. Mögliche Werte sind:

- `0`: der aktuell laufende Prozess darf unterbrochen werden und
- `1`: der Prozess darf nicht unterbrochen werden.

Der Anforderung eines Records innerhalb der Funktion `audit_get_free_record(...)` verläuft folgender Maßen. Zunächst wird der geforderte Typ des Puffers überprüft, ob er überhaupt in der aktuellen Konfiguration unterstützt wird. Ist dies der Fall, wird der Zustand des Audit-Puffers aus der globale Variable `au_buf_state` ermittelt. Ist dieser bereits voll, wird das Event verworfen, wenn sich das System nicht im **`secure`**-Mode befindet, der Prozess nicht unterbrechbar ist oder das Event vom Audit-Dämon generiert wurde. Andernfalls wird der laufenden Prozess mittels der Funktion `buf_suspend(...)`unterbrochen.

Um konkurrierende Zugriffe gegenseitig ausschließen zu können, muss das Puffermanagement an dieser Stelle mittels eines Spinlocks gesichert werden. Anschließend kann auf den ersten freien Record zugegriffen werden. Die globale Variable `au_first_free` verweist immer auf diesen Record. Da der Prozess für einen gewissen Zeitraum im Lock gefangen sein kann, muss der Zustand des Puffers

nach dem Aufwecken des Prozesses erneut überprüft werden. Ist dieser gleich AU_FREE, wird die globale Variable au_first_free auf dessen Nachfolger weitergerückt. Ist der Zustand ungleich AU_FREE, wurde der letzte freie Record zwischenzeitlich durch einen anderen Prozess belegt. Da sich der Prozess nun innerhalb eines Spinlocks befindet, kann er nicht mehr unterbrochen werden und die Bearbeitung der Anforderungen muss auf einem anderen Weg erfolgen. Mittels der Funktion audit_new_header(...) wird ein neuer Audit-Record erstellt. Sollte hierfür kein Speicher zur Verfügung stehen, muss die Protokollierung des Events an dieser Stelle abgebrochen werden.

Nachdem ein freier Audit-Record ermittelt wurde, muss ein Typ-spezifische Record angefordert werden. Dies erfolgt mittels der Funktion get_free(...) des Puffers. Sollte dieser keine freien Records enthalten, muss mittels der Funktion free_rec(...) ein neuer Record erstellt werden. War dies nicht möglich, muss das Event verworfen werden.

Nachdem sowohl ein freier Audit- als auch ein freier Typ-spezifischer Record ermittelt werden konnte, werden die für alle Audit-Events gültigen Daten in den Audit-Record kopiert. Hierzu gehört die aktuelle Systemzeit, der Rechnername und der Record-Typ. Anschließend wird der Record als AU_USED gekennzeichnet und der anfordernden Funktion mittels des Return-Wertes zurückgegeben.

4.1.2.5 **Speicherung des Puffers**

Das Speichern der Daten wird vom Audit-Dämon initiiert. Dieser wird vom BS-Kern mittels des Signals SIG_ALRM darüber informiert, dass die Daten des Puffers gespeichert werden müssen. Daraufhin ruft der Audit-Dämon die Funktion audit_service(...) mit dem Datei-Deskriptor des aktuellen Audit-Trails und dem Limit der maximal zu speichernden Bytes als Parameter auf.

Die Funktion audit_service(...) ermittelt aus dem ihr übergebenen Datei-Deskriptor die file-Struktur, die die write(...)-Funktion des darunter liegenden Dateisystems enthält. Da die file-Struktur Bestandteil des virtuellen Dateisystems ist, ist die audit_service(...)-Funktion nicht auf bestimmte Dateisysteme beschränkt und sogar in der Lage, die Daten in einen Socket zu schreiben. Damit wird es leicht möglich, die Audit-Trails zentral auf einem Log-Server zu verwalten.

Die Verwendung der write(...)-Funktion des virtuellen Dateisystems macht es erforderlich, dass die Daten des Audit-Records zwischenzeitlich in den Nutzerspeicher kopiert werden müssen. Aus diesem Grund muss der Audit-Dämon beim Aktivieren des BS-Audit eine Adresse auf einen ausreichend großen Speicherbereich dem Audit-Modul übergeben. Die Größe des Puffers ist abhängig von den unterstützten Audit-Record-Formaten und ist in dem Define AU_WRITE_BUFFER_SIZE abgelegt.

Zum Speichern des Audit-Records müssen dessen Daten in einen String-Puffer kopiert werden. Dies erfolgt mittels der Funktion audit_record2char(...), die wiederum die rec2chr(...)-Funktionen der einzelnen Typ-spezifischen Puffer nutzt. Zuvor wird überprüft, ob sich in dem String-Puffer noch die Daten eines Records befinden. Ist dies der Fall, wird dieser als erstes gespeichert, erst danach kann ein neuer Record aus dem Audit-Puffer übernommen werden. Konnte der Record in den String-Puffer kopiert werden, liefert die Funktion audit_record2char(...) die Anzahl der kopierten Bytes zurück. Anschließend kann die Anzahl mit dem übergebenen Limit verglichen werden. Sollte die Größe des Records nicht mehr im Limit liegen, werden die Daten im String-Puffer belassen. Sie befinden sich nicht mehr im Audit-Puffer und müssen aus diesem Grund beim nächsten Aufruf von audit_service(...) als erstes gespeichert werden. Mit diesem Mechanismus wird erreicht, dass das Limit in jeden Fall eingehalten werden kann, ohne das ein Record verworfen werden muss, obwohl der benötigte Speicherplatz für den aktuellen Record erst nach dem Löschen aus dem Audit-Puffer bekannt ist. In der LOSA-Implementierung war dies nicht der Fall, hier wurde der Record gespeichert, obwohl das Limit womöglich überschritten wurde.

Der Aufruf von `audit_record2char(...)` erfolgt solange, bis das Trail-Limit erreicht wurde oder keine Records mehr vorhanden sind. Letzteres wird durch die Funktion `audit_record2char(...)` mittels des Fehlers `ENODATA` signalisiert.

4.1.3 Sicherung der Vollständigkeit des Audit-Trails

Eine der wesentlichen Anforderungen an ein BS-Audit ist die Sicherstellung der Vollständigkeit des Audit-Trails. Sollte es während des Betriebs notwendig sein, einzelne Audit-Events zu verwerfen, da keine Ressourcen zur Speicherung zur Verfügung stehen, kann später nicht nachvollzogen werden, ob eine IT-Sicherheitsverletzung stattgefunden hat oder nicht.

In der hier vorgestellten Implementierung wurden zwei Mechanismen integriert, welche die Vollständigkeit der protokollierten Daten im Audit-Trail sicherstellen sollen. Aber auch mit diesen beiden Mechanismen kann es passieren, dass Events verworfen werden müssen. Warum dies nicht verhindert werden kann, wird in diesem Abschnitt an den entsprechenden Stellen dargelegt.

4.1.3.1 Unterbindung der Generierung von zusätzlichen Audit-Events

Eine Möglichkeit zur Einhaltung der vorhandenen Ressourcen besteht durch das Unterbinden der Generierung von Audit-Events. Die Protokollierung des Audit-Events wird entweder durch einen Systemruf oder durch die Verarbeitung eines Netzwerk-Paketes initiiert. Ist es möglich diese Operationen für einen gewissen Zeitraum auszusetzen, wird ebenso die Generierung von Audit-Events unterbrochen.

Um das Verhalten des Systems nicht zu stark zu beeinflussen, muss das Unterbrechen der Operationen für einen Nutzer völlig transparent erfolgen. Für die Protokollierung der Systemrufe bzw. die Überwachung der Paketverarbeitung sind hierzu verschiedene Ansätze notwendig.

Unterbrechen von Systemrufen

Die Ausführung eines Systemrufs erfolgt immer im Kontext eines Prozesses, welcher sich in verschiedenen Ausführungszuständen befinden kann. Ist es nicht möglich die Anforderung nach einem freien Audit-Record zu erfüllen, besteht die Möglichkeit den zugehörigen Prozess solange zu unterbrechen, bis wieder ein freier Audit-Record zur Verfügung stehen.

Das Unterbrechen eines Prozess ist auf zwei Wegen möglich. Entweder wird durch die Interrupt-Behandlung des Systems dem Prozess die Kontrolle über den Systemprozessor entzogen oder der Prozess gibt selbst die Kontrolle ab. Da die Interrupt-Behandlung nicht ohne weiteres verändert werden kann, wird vom Audit-System die zweite Möglichkeit genutzt. Ein Systemruf wird immer von einem Prozess ausgeführt, damit erfolgt auch die Anforderung eines freien Audit-Records im Kontext eines Prozesses. Kann diese nicht erfüllt werden, wird der Aufruf der Funktion `sleep_on(...)` veranlasst. Damit gibt der Prozess die Kontrolle des Prozessors solange ab, bis er von einer externen Instanz wieder aufgeweckt wird. Der Aufruf von `sleep_on(...)` erfolgt bei der Anfrage nach einem Host-Records durch die Funktion `buf_suspend(...)` des Host-Puffers. Die Anforderung eines Netz-Records erfolgt nicht im Kontext eines Prozesses, demzufolge ist dieser Weg für Netzwerk-Events nicht möglich.

Stehen wieder freie Records zur Verfügung wird die Funktion `buf_wake_up(...)` des Puffers aufgerufen. Die Funktion des Host-Puffers weckt daraufhin alle zuvor schlafen gelegten Prozesse auf. Jeder Prozess setzt seine Arbeit unmittelbar nach dem Aufruf von `sleep_on(...)` fort, also mitten in der Anforderung nach einem freien Audit-Record. Der Kontext des zuvor ausgeführten Systemrufs wurde wiederhergestellt und kann nun protokolliert werden. Für den Nutzer bleibt dies völlig transparent.

Wichtig ist, dass sichergestellt werden kann, dass der Audit-Puffer geleert wird und die Funktion `buf_wake_up(...)` aufgerufen wird. Anderenfalls bleibt mitunter das gesamte System stehen, ohne das es wieder in Gang gesetzt werden kann. Um dies zu verhindern, werden die Aktionen des Audit-Dämons nicht zwingend protokolliert, d.h. steht kein Audit-Record zur Verfügung, wird das Event verworfen und eine Syslog-Nachricht generiert. Damit kann sichergestellt werden, dass der Audit-Dämon nicht unterbrochen wird und nach einer endlichen Zeit den Prozessor zugeteilt bekommt, um den Audit-Puffer zu leeren und die schlafenden Prozesse aufzuwecken.

Anhalten der Paketverarbeitung

Die Generierung von HoNA-Events erfolgt im Rahmen der Paketverarbeitung durch den Netzwerk-Stack. Die Verarbeitung eines Paketes beginnt in der Funktion `net_rx_action(...)`. Hier werden die Pakete aus dem Paketpuffer des aktuellen Prozessors genommen und vollständig verarbeitet. Wurde ein Paket aus dem Puffer genommen und mit dessen Verarbeitung begonnen, ist es nicht mehr möglich diese zu unterbrechen, ohne dass das Paket verworfen werden muss. Ziel ist also die Unterbrechung der Paket-Verarbeitung in der Funktion `net_rx_action(...)`. Hierzu wurde innerhalb der Funktion eine zusätzliche Abfrage eingefügt, die vor dem Entnehmen des Paketes aus dem Paketpuffer überprüft, ob ausreichend Audit-Records für die Protokollierung eines möglichen Ereignisses zur Verfügung stehen. Erst wenn dies sichergestellt wurde, wird das Paket verarbeitet.

Demzufolge signalisiert die Funktion `buf_suspend(...)` des HoNA-Puffers dem Netzwerk-Stack, dass mit der Verarbeitung eines neuen Paketes nicht mehr begonnen werden darf. Die Funktion `buf_wake_up(...)` reaktiviert die Verarbeitung von Netzwerk-Paketen.

Treffen während der Deaktivierung des Netzwerk-Stacks Pakete für das System ein, werden diese von der Netzwerkkarte wie sonst auch entgegengenommen und in den Paketpuffer kopiert. Um im Falle einer starken Netzlast nicht alle Ressourcen des Systems zu verbrauchen, ist dieser Puffer auf 300 Pakete beschränkt. Ist der Puffer komplett gefüllt, werden alle weiteren Pakete verworfen. Dies bedeutet, dass für diese Pakete auch keine Audit-Events generiert werden können. Was wiederum auch nicht notwendig ist, da diese Pakete auf dem System niemals eine Aktion auslösen werden. Bei der Verwendung von verbindungsorientierten Protokollen ist zudem sichergestellt, dass ein verworfenes Paket ein weiteres Mal gesendet wird.

4.1.3.2 Peak-Puffer

Bei der Generierung einzelner Audit-Events ist es mitunter nicht möglich die Ausführung zu unterbrechen. Um auch für diese Ereignisse einen Audit-Record zu erhalten, wurde die bereits zuvor erwähnten `free_rec(...)`-Funktion eingefügt. Die Verwaltung dieser zusätzlichen Audit-Records erfolgt mittels des sogenannten Peak-Puffers.

Beim Peak-Puffer handelt es sich um eine einfach-verkettete Liste von Audit-Records, die über die globale Variable `au_additional` adressiert wird. Die Records sind mittels des `next`-Zeigers der Struktur `audit_record` verbunden. Die Liste ist wiederum in einzelne Sessions unterteilt.

Peak-Puffer Session

Sollte der Audit-Record-Puffer keinen freien Record mehr enthalten, wird eine neue Session im Peak-Puffer angelegt. Sie enthält alle zusätzlichen Audit-Records bis zum nächsten Freiwerden des Audit-Puffers. Der Beginn einer Session wird mittels des `prev`-Zeigers gekennzeichnet. Dieser zeigt hierzu auf den Record selbst.

Sollte der Audit-Puffer wiederum vollständig gefüllt sein, bevor der Peak-Puffer gespeichert werden konnte, wird eine neue Session angehangen. Sie wird entsprechend gekennzeichnet und muss beim Speichern des Puffers an der richtigen Stelle im Trail abgelegt werden.

Speichern des Peak-Puffers

Das Speichern des Peak-Puffers erfolgt immer nach dem Speichern des Audit-Puffers. Enthält dieser keine Daten mehr, werden die Records des Peak-Puffers verarbeitet, sofern welche vorhanden sind. Begonnen wird mit dem Record, der sich hinter dem Zeiger `au_additional` befindet. Wurde der Inhalt des Records in den String-Puffer kopiert, wird der Speicherbereich des Records freigegeben. Dies erfolgt solange, bis der `prev`-Zeiger des aktuellen Records auf sich selbst zeigt. An dieser Stelle würde eine neue Session des Peak-Puffers beginnen.

Eine neue Session kann nur enthalten sein, wenn zuvor der Audit-Puffer neu gefüllt wurde, während der erste Peak-Puffer gespeichert wurde. Aus diesem Grund muss vor dem Speichern der nächsten Session der Audit-Puffer verarbeitet werden, um die Reihenfolge der Audit-Records einzuhalten.

Puffer-Rekonfiguration

Neben der Kompensation von zusätzlichen Record-Anforderungen wird der Peak-Puffer auch bei der Rekonfigurierung des Audit-Puffers verwendet. Soll die Größe eines Typ-spezifischen Puffers geändert werden, muss der aktuelle Puffer gelöscht werden. Es ist nicht möglich neue Elemente in einen Puffer hinzufügen oder überflüssige Elemente zu löschen.

Da der aktuelle Audit-Puffer möglicherweise noch Daten enthält, wird der gesamte Puffer in eine Session des Peak-Puffers umgewandelt. Hierzu markiert der erste volle Record den Beginn einer neuen Session. Die `prev`-Zeiger der übrigen Records werden auf Null gesetzt, so dass sie als Elemente des Peak-Puffers erscheinen. Die Records des neu zu konfigurierenden Typ-spezifischen Puffers werden ebenfalls als Peak-Puffer-Element markiert. Beim nächsten Aufruf der Funktion `audit_service(...)` werden die Elemente des ursprünglichen Audit-Puffers und des Typ-spezifischen Puffers als Peak-Puffer-Elemente behandelt und ihr Speicherbereich freigegeben.

Da während des Speicherns die alten Audit-Records freigegeben werden, muss zuvor ein neuer Audit-Puffer erstellt werden. Dieser ersetzt den Puffer der ursprünglichen Konfiguration und wird bei den nächsten Record-Anfragen gefüllt. Zuletzt muss nur noch sichergestellt werden, dass der alte Puffer, der jetzt eine Session des Peak-Puffers ist, vor dem neuen Audit-Puffer und der letzten Session des Peak-Puffers gespeichert wird. Hierzu wird mittels der globalen Variable `au_use_add` der `audit_service(...)`-Funktion signalisiert, dass sie den Peak-Puffer vor dem Record-Puffer speichern soll.

4.1.4 Implementierung der Filterregeln

Wie schon bei der Implementierung der Audit-ACL von LOSA, werden die Filterregeln der HoNA-Implementierung zur Durchsetzung der Aufzeichnungsgranularität im Audit-Modul abgelegt. Hierzu müssen die Daten mit Hilfe des Steuerungsprogramms **`aca`** dem Audit-Modul übergeben werden. Im Kernel-Modul werden die Daten in einer eigens dafür angelegten Datenstruktur verwaltet. Die notwendigen Datentypen und Accessor-Methoden sind Bestandteil der HoNA-Erweiterung und auch nur in dieser gültig.

Dieser Abschnitt beinhaltet der Beschreibung der hierfür notwendigen Datenstrukturen und Accessor-Methoden. Die Erläuterung der notwendigen Änderungen am Steuerungsprogramm wird hier ausgelassen.

4.1.4.1 Datenstrukturen

Alle für die Durchsetzung der HoNA-Filterregeln notwendigen Informationen werden in den Strukturen `audit_ip_acl`, `audit_ip_rng` und `audit_port_acl` abgelegt. Darüber hinaus wird die Struktur `audit_acl_msk` für die Speicherung der Selektionsmasken verwendet.

Die einzelnen Regeln werden, wie in Abbildung 13 dargestellt, in einer Baumstruktur verwaltet. Für die Speicherung von IPv4- und IPv6-Regeln werden verschiedene Bäume verwendet. Zwar bestehen beide Bäume aus den gleichen Strukturen, jedoch sind deren Einträge teilweise anders belegt und lassen sich nur schwer ordnen, so dass eine Speicherung in nur einem Baum nicht möglich ist. Die Konten werden absteigend zu den Blättern anhand der Netzmaske sortiert. Knoten, die sich auf einer Ebene befinden, enthalten IP-Adressen der gleichen Netzmaske. Der Wurzelknoten enthält die Netzmaske 0.0.0.0 bzw. den CIDR-Block 0. Die Netzmaske der Blätter ist abhängig von den eingefügten Regeln und der Adressfamilie.

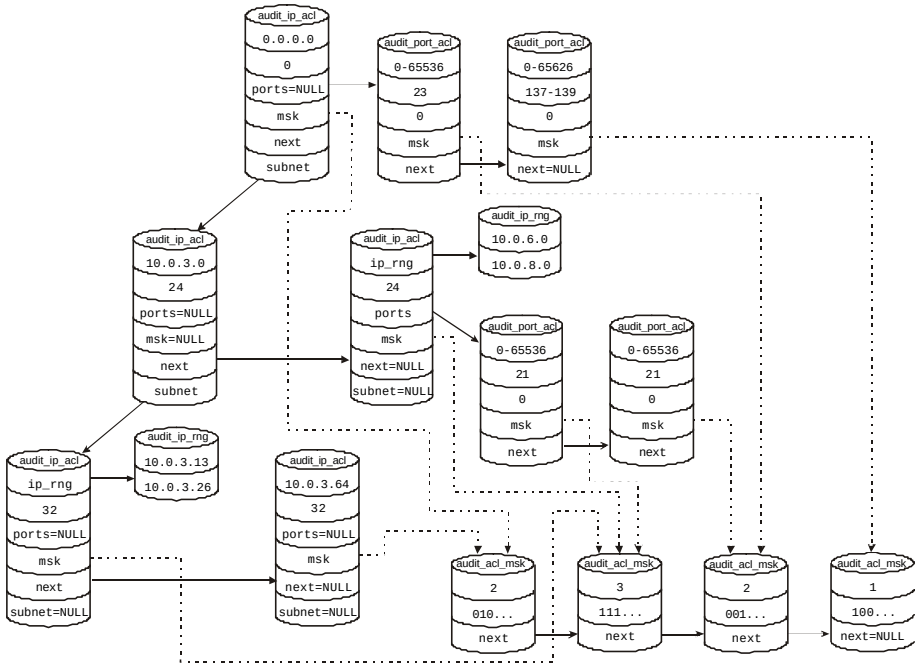

Abbildung 13 : Datenstrukturen der ACL-Regeln

struct audit_ip_acl

Die Baumstruktur wird aus Objekten der Struktur `audit_ip_acl` aufgebaut. Jedes Objekt enthält die Regeln zu einer IP-Adresse oder zu einen Bereich von IP-Adressen. Die Struktur hat folgende Elemente:

`union addr`

Enthält die IP-Adresse der Regel. Bezieht sich die Regel auf eine IPv4-Adresse, wird diese hier direkt eingetragen. IPv6-Adressen und IP-Bereiche benötigen wesentlich mehr Speicherplatz und werden aus diesem Grund in einer separaten Struktur gespeichert. Das Union enthält für diese Fälle einen Zeiger auf ein Objekt der Struktur `audit_ip_rng`.

`__u8 cidr`

Da eine IPv6-Adresse 16 Bytes belegt, wird nur der CIDR-Block gespeichert. Dieser benötigt lediglich 1 Byte.

`__u8 direc`

Der Eintrag `direc` enthält die Beschreibung von `addr`. Ist der Eintrag für ein IPv4-Objekt auf Eins gesetzt, enthält die Union eine IPv4-Adresse anderenfalls einen Zeiger auf ein Objekt der Struktur `audit_ip_rng`. Die Adressen einer IPv6-Regeln werden nie in dem `audit_ip_acl`-Objekt direkt abgelegt. Für sie gibt der Eintrag `direc` an, ob im `audit_ip_rng`-Objekt ein Bereich oder nur eine IPv6-Adresse abgelegt ist.

`struct audit_acl_msk *msk`

Für Events, die keine Portnummern beinhalten, oder für Regeln, die sich auf alle Portnummern beziehen, wird die Selektionsmaske direkt mit dem `audit_ip_acl`-Objekt verknüpft.

`struct audit_port_acl *ports`

Regeln, die sich auf einzelne Portnummern beziehen, werden in einer separaten Struktur verwaltet. Die Objekte werden an das betreffende `audit_ip_acl`-Objekt gehangen und benötigen somit keinen Eintrag zur Speicherung der IP-Adresse.

Für die Realisierung des Baums enthält die Struktur zusätzlich die Zeiger `next`, `prev`, `subnet` und `net`. Dabei verweisen die Zeiger `next`, `prev` auf den Nachfolger bzw. Vorgänger der selben Ebene, während `subnet` und `net` für die Adressierung der darunter bzw. darüber liegenden Ebene verwendet werden.

`struct audit_ip_rng`

Die Struktur `audit_ip_rng` enthält lediglich die zwei Unions `low` und `high`, die entweder eine IPv4- oder eine IPv6-Adresse speichern. Welche IP-Adresse aktuell verwendet wird, ist durch die Zuordnung des Objektes zu der jeweiligen Regelbasis vorgegeben. In der Regelbasis für IPv6-Adressen enthält die Struktur `audit_ip_acl` immer ein `audit_ip_rng`-Objekt. Enthält die Regel keinen Bereich sondern nur eine IPv6-Adresse, wird diese in dem Union `low` abgelegt.

Die Verwendung eines Unions ist normalerweise nicht speichereffizient, da immer soviel Speicher angefordert wird, wie das größte Element der Union benötigt. Für ein `audit_ip_rng`-Objekt bedeutet dies, dass immer 32 Bytes bereitgestellt werden, obwohl ein IPv4-Objekt nur 8 Bytes benötigen würde. Dies kann aber ohne weiteres akzeptiert werden, da im BS-Kern mittels der Funktion `kmalloc(...)` kleinere Objekte als 32 Bytes nicht angefordert werden können. Auch wenn man mittels der Funktion `kmalloc(...)` nur ein Objekt für 8 Byte anfordert, erhält einen Speicherbereich für 32 Byte.

`struct audit_port_acl`

Bei den Protokollen TCP und UDP werden die einzelnen Verbindungen mit Hilfe von Portnummern den Prozessen zugeordnet. Das Regelwerk von HoNA gestattet es, Portnummern in die Regeln mit einzubeziehen. Diese werden in Objekten vom Typ `audit_port_acl` gespeichert.

Die Einträge `low_sport` und `high_sport` der Struktur speichern die Quellportnummern und die Einträge `low_dport` und `high_dport` die Zielportnummern. Wurde kein Bereich sondern lediglich eine Portnummer angegeben, sind die Werte der `low`- und der `high`-Portnummern identisch. Regeln, in denen das Schlüsselwort `any` für die Angabe der Portnummer verwendet wurde, beziehen sich auf alle Portnummern von 0 bis 65536. Gilt dies für den Quell- als auch für den Zielport wird kein `audit_port_acl`-Objekt erzeugt, da diese Regel im `audit_ip_acl`-Objekt gespeichert werden kann.

Ist der Quellport, sowohl als solcher als auch als Zielport zu interpretieren, enthält der Eintrag `direc` den Wert 1, anderenfalls ist dieser immer auf Null gesetzt.

Die Selektionsmaske wird wiederum in einem `audit_acl_msk`-Objekt gespeichert. Das betreffende Objekt wird über das Element `msk` adressiert.

Enthält ein `audit_ip_acl`-Objekt mehrere Regeln für verschiedene Portnummern oder verschiedene Events, werden diese in separaten `audit_port_acl`-Objekten verwalten. Die einzelnen Objekte sind mittels des `next`-Zeigers in Form einer einfach-verketteten Liste verbunden. Beim letzten Element enthält `next` einen Null-Zeiger. Die einzelnen Objekte werden sortiert

eingefügt, wobei die Sortierung wie folgt geschachtelt ist: `low_sport`, `high_sport`, `low_dport`, `high_dport`.

struct audit_acl_msk

Die `audit_acl_msk`-Objekte werden in einer separaten Liste verwaltet, wobei jedes Objekt mehrfach verwendet werden kann. Neben den Netz-ACL-Objekten nutzen die ACL-Objekte für Ressourcennamen und die Audit-Strukturen der Systemprozesse der LOSA-Implementierung die globale Liste.

Ein `audit_acl_msk`-Objekt enthält die folgenden Elemente:

`count`
: Ist der Referenzzähler des Objektes. Nur wenn der Eintrag eine Eins enthält, darf es gelöscht oder verändert werden. Ansonsten muss bei einer Änderung ein neues Objekt erstellt bzw. ein anderes passendes Objekt gesucht werden.

`val`
: Enthält eine 32 Byte große Bitmaske zur Speicherung der Selektionsmaske. Die Anordnung der Events in der Maske bzw. die Bedeutung der einzelnen Bits ist abhängig vom Objekttyp, der die Maske verwendet. Für HoNA-Events wird die Bitmaske in eine 16x16-Bit Matrix aufgeteilt, wobei die Zeilen für die HoNA-Major und Spalten für die Minor-Event verwendet werden. Ein Event wird protokolliert, wenn das zugehörige Bit in der `val`- und in der `perm`-Maske auf Eins gesetzt ist.

`perm`
: Die `perm`-Maske ist immer in Verbindung mit der `val`-Maske zu nutzen und hat die gleiche Größe. Die Adressierung der einzelnen Bits ist wiederum abhängig vom Objekttyp. Ist das Bit der `perm`-Maske auf Eins gesetzt, so gilt der Wert der `val`-Maske für dieses Objekt. Dies ist notwendig, da in der `val`-Maske auch einzelne Bits verwendet und diese auf Null gesetzt werden können.

Die einzelnen Objekte werden in einer doppelt-verketteten Liste verwaltet, die über die globale Variable `acl_msk_list` adressiert wird. Der Zugriff auf die Elemente der Liste erfolgt über spezielle Accessor-Methoden, die intern automatisch neue Objekte hinzufügen bzw. überflüssige Objekte löschen.

4.1.4.2 Überprüfung eines Audit-Events

Die Überprüfung eines HoNA-Events, ob dieses aufgezeichnet werden soll oder nicht, beginnt immer mit der Funktion `audit_net_doit(...)`. Sie erwartet das Major- und das Minor-Event sowie einen Zeiger auf das Paket und dessen Adressfamilie. Letzteres ließe sich zwar auch aus dem Paket ermitteltn, ist so aber effizienter auszuwerten. Die Funktion liefert als Ergebnis eine Eins, wenn das Event aufgezeichnet werden soll, oder eine Null zurück. Im Falle eines Fehlers wird immer eine Eins zurückgegeben, damit das Ereignis aufgezeichnet wird, da bzgl. der Regelbasis keine Aussage getroffen werden konnte.

Anhand der Adressfamilie entscheidet die Funktion, welche Regelbasis durchsucht werden soll. Das eigentliche Durchsuchen der Regelbasis erfolgt mittels einer der beiden Funktionen `acl_check_net_ipv4(...)` und `acl_check_net_ipv6(...)`. Die Funktionen erwarten das Major- und das Minor-Event sowie einen Zeiger auf das Paket. Ist während der Suche ein Fehler aufgetreten, geben die Funktionen den negativen Wert des Fehlercodes zurück.

Die Suche beginnt immer im Wurzelknoten mit dem CIDR-Block 0. Initial wird die gesamte `perm`-Maske des Wurzelknotens auf 1 gesetzt. Damit wird sichergestellt, dass für jedes Event eine gültige

Regel vorhanden ist. Der initiale Werte für die `val`-Maske wird der Konfiguration des Kernels entnommen. Standardmäßig ist dieser auf Eins gesetzt, so dass ohne das Hinzufügen von Regeln alle Ereignisse aufgezeichnet werden. Der Wert der `val`-Maske des Wurzelknotens wird als vorläufiges Ergebnis der Suche gespeichert.

Wurden für den Wurzelknoten Ports definiert und unterstützt das dem Ereignis zugrundeliegende Protokoll Portnummern, muss die Liste der `audit_port_acl`-Objekte durchsucht werden. Wurde ein Objekt in der Liste gefunden, dessen Portnummer mit dem Event übereinstimmt und dessen `perm`-Maske signalisiert, dass das Objekt für das Event gültig ist, wird der Wert aus dem gefundenen Objekt als vorläufiger Rückgabewert übernommen. Die Objekte der Liste sind aufsteigend anhand der Quellportnummer sortiert. Die Suche kann abgebrochen werden, wenn kein Objekt mehr vorhanden oder eine Portnummer des aktuellen Objektes größer als die entsprechende Portnummer des Paketes ist.

Enthält der Wurzelknoten einen `subnet`-Eintrag, muss dieser anschließend nach einem gültigen Objekt durchsucht werden. Dabei wird zunächst mit der Suche nach einer passenden IP-Adresse bzw. IP-Bereich begonnen. Die Objekte innerhalb einer Ebene sind anhand der Adressen sortiert, d.h. die Suche kann abgebrochen werden, wenn die IP-Adresse größer ist bzw. der IP-Bereich oberhalb der Quelladresse des aktuellen Paketes liegt oder kein weiteres Objekt auf dieser Ebene vorhanden ist. Konnte ein passendes Objekt gefunden werden und ist dieses für das Event gültig bzw. existiert ein gültiges `audit_port_acl`-Objekt, wird der bisher gespeicherte Rückgabewert durch den Wert des neuen Objektes ersetzt. Enthält der Knoten wiederum einen `subnet`-Eintrag, wird die Suche in diesen Objekten fortgesetzt. Es ist nicht möglich, dass zu einem CIDR-Block zwei `audit_ip_acl`-Objekte existieren, die sich auf die gleiche Quelladresse beziehen. Demzufolge sind alle Objekte, die sich hinter dem `next`-Zeiger des aktuellen Objektes befinden, unrelevant.

Die Suche wird solange fortgesetzt, bis das aktuelle `audit_ip_acl`-Objekt auf eine Host-Adresse verweist oder kein gültiges Objekt mehr vorhanden ist. In diesem Fall ist der zuletzt gespeicherte `val`-Wert das Ergebnis der Suche.

4.1.4.3 Einfügen von Regeln

Bereits mehrfach wurde darauf eingegangen, dass für IPv4- und IPv6-Adressen getrennte Regelbasen angelegt werden. Aus diesem Grund muss bereits vor dem eigentlichen Einfügen ermittelt werden, ob die neue Regel eine IPv4- oder eine IPv6-Adresse anhält. Dies erfolgt bereits im Steuerungsprogramm und wird im Audit-Modul durch die Verwendung verschiedener Funktionen implizit vorgegeben.

Das Einfügen einer ACL-Regel erfolgt immer mittels der Funktion `sys_acl_ctl(...)`. Sie erwartet die ID der aufzurufenden Funktion und ein Array mit den notwendigen Parametern. Die eigentlichen Funktionen zum Einfügen der neuen Regel werden in Arrays von Funktionszeigern verwaltet. Der Index der Funktion innerhalb des Arrays wird aus der übergebenen ID ermittelt und die Interpretation der Parameter erfolgt in dieser Funktion. Für das Einfügen einer Regel werden die folgenden Funktions-Arrays verwendet:

`acl_lock`
: Sichert die zugehörige Regelbasis mittels eines Spinlocks.

`acl_check_perm`
: Überprüft die Zugriffsrechte der Regelbasis. Für HoNA-Events benötigt der aufrufende Prozess die Capability `AU_ADMIN`.

`acl_get_entry`
: Ermittelt ein passendes ACL-Objekt, in dem die neue Regel eingetragen werden kann. Ist bisher kein passendes Objekt vorhanden, erstellt die Funktion ein neues.

acl_set

Aktualisiert die Selektionsmaske des Objektes. Für Regeln, die sich auf eine Portnummer beziehen, muss diese Funktion zuvor ein `audit_port_acl`-Objekt ermitteltn bzw. erstellen.

acl_unlock

Löst den Spinlock der Regelbasis.

Der Großteil der Arbeit zum Einfügen einer HoNA-Regel wird innerhalb der Funktion `acl_get_entry_ipv4(...)` bzw. `acl_get_entry_ipv6(...)` erbracht. Sie wird über das Array `acl_get_entry` adressiert. Ist bereits ein passendes Objekt vorhanden, so wird dieses zurückgegeben. Andernfalls muss ein neues Objekt erstellt werden. Hierbei kann es mitunter notwendig werden, dass der Baum stark umstrukturiert werden muss. Insbesondere wird dies notwendig, wenn eine neue Regel mit einem IP-Bereich eingefügt werden soll.

In Abbildung 14 besteht die Regelbasis lediglich aus Regeln mit direkten IP-Adressen, doch schon in diesem Fall muss der Baum beim Einfügen einer neuen Regel, im Beispiel die Netzadresse 141.4.3.128/25, umstrukturiert werden, da die betreffende Netzmaske nicht vorhanden ist. Zunächst muss die Ebene ermittelt werden, wo die neue Regel eingetragen werden muss. Alle Regeln, die unterhalb der neuen Ebene liegen (im Beispiel 141.43.3.1/32 und 141.43.3.165/32), müssen auf die Objekte der neuen Ebene aufgeteilt werden. Für Regeln, die nicht in das Subnetz der neuen Regel fallen (141.43.3.1/32), müssen zusätzliche Objekte erstellt werden. Deren `perm`-Maske wird auf Null gesetzt, so dass sie bei einer Suche nicht berücksichtigt werden.

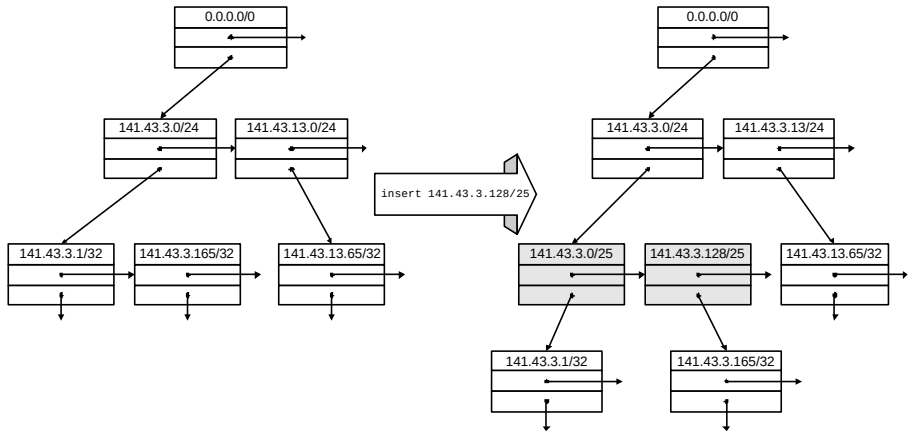

Abbildung 14 : Einfügen einer neuen IPv4-ACL-Regel

Wesentlich umfangreicher wird das Einfügen von IP-Bereichen oder das Einfügen von IP-Adressen in eine Regelbasis mit IP-Bereichen. Überschneiden sich die neuen Regeln mit bereits vorhandenen Regeln, muss diese in einzelne Regeln zerlegt werden. Bspw. würde ein Objekt mit dem CIDR-Block 32 und dem IP-Bereich 141.43.3.1-141.43.3.64 beim Einfügen der Adresse 141.43.3.32/32 in die drei Objekte 141.43.3.1-141.43.3.31, 141.43.3.32 und 141.43.3.33-141.43.3.64 zerlegt werden. Alle drei Objekte erhalten die Selektionsmaske des ursprünglichen Objektes. Wären zusätzlich Sub-Netze vorhanden, was im Beispiel nicht möglich ist, müssten diese auf die neuen Objekte aufgeteilt werden. Die Selektionsmaske des Objektes mit der IP-Adresse 141.43.3.32 muss zusätzlich mit der Maske des neuen Objektes verknüpft werden.

Wurde ein `audit_ip_acl`-Objekt gefunden bzw. eingefügt, wird mittels der Funktion `ip_acl_update_msk(...)` die `perm`- und die `val`-Maske des Objektes aktualisiert bzw. die Liste der `audit_port_acl`-Objekte nach einem passenden Objekt durchsucht und dieses aktualisiert. Existiert kein passendes Objekt, wird dieses erzeugt, initialisiert und sortiert eingefügt.

Die Funktionszeiger des Arrays `acl_set` verweisen auf Adressfamilien-spezifische Funktionen, die wiederum die Funktion `ip_acl_update_msk(...)` aufrufen.

4.2 Die Audit-Bibliothek `libaudit`

Die Audit-Bibliothek ist die Schnittstelle für Nutzerapplikationen zum Audit-Modul im BS-Kern. Die im Folgenden aufgeführten Funktionen sind eine reine Erweiterung der von LOSA bereitgestellten Audit-Bibliothek. Für das Einbringen der HoNA-Funktionalität war es notwendig, die Initialisierung des Audit-Systems und der Verwaltung der ACL-Regeln zu überarbeiten sowie die Leseoperationen für die Audit-Trails zu erweitern. Die Änderungen werden im Folgenden beschrieben.

4.2.1 Verwaltung des Audit-Systems

Für die Initialisierung des Audit-Systems sind zwei Schritte notwendig: zum einen das Einrichten der Audit-Puffer und zum anderen das Einbringen der ACL-Regeln. Die notwendigen Operationen zum Setzen und Auslesen der ACL-Regel werden als ACL-Submodul zusammengefasst. In diesem Abschnitt wird nur auf die Verwaltung der Audit-Puffer und das Aktivieren der Protokollierung eingegangen. Die Beschreibung des ACL-Submoduls erfolgt im nächsten Abschnitt.

Verwaltung der Audit-Puffer

Für die Verwaltung der Audit-Puffer bietet die Audit-Bibliothek die drei Funktionen `audit_buf_init(...)`, `audit_buf_destroy(...)` und `audit_buf_config(...)`.

`audit_buf_init(unsigend int size, unsigned long *arg)`

Die Funktion `audit_buf_init(...)` dient der Initialisierung des Audit-Puffers. Sie erwartet als erstes Argument die Anzahl der zu konfigurierenden Puffer und als zweites Argument einen Zeiger auf ein Array mit den zugehörigen Daten.

Jedes Element des übergebenen Arrays sollte einen Zeiger auf ein `int`-Array mit zwei Elementen enthalten, dessen erstes Element den Typ des zu initialisierenden Puffers und als zweites Element die Anzahl der Audit-Records enthalten sollte. Sind in dem der Funktion `audit_buf_init(...)` übergebenen Array mehrere Elemente gespeichert, die sich auf ein und den selben Puffertyp beziehen, wird die Anzahl der Records der einzelnen Einträge zusammenaddiert und ein Puffer mit der ermittelten Größe eingerichtet.

War die Ausführung der Funktion erfolgreich, kann die Protokollierung mittels der Funktion `audit_on(...)` aktiviert werden.

`audit_buf_destroy(void)`

Die Funktion `audit_buf_destroy(...)` löscht alle eingerichteten Puffer und gibt den von ihnen benötigten Speicher frei. Die Ausführung der Funktion ist nur möglich, wenn die Protokollierung zuvor deaktiviert wurde.

`audit_buf_config(int buf, unsigned int cmd, unsigned long args)`

Für die Verwaltung der dynamischen Zusatzpuffer und zur Rekonfigurierung der Audit-Puffer wird die Funktion `audit_buf_config(...)` bereitgestellt. Die Nutzung der Funktion ist nur möglich, wenn zuvor der entsprechende Puffer mittels `audit_buf_init(...)` eingerichtet wurde.

Das erste Argument der Funktion enthält den zu konfigurierenden Puffertyp, das zweite Argument enthält eines der folgenden Kommandos:

- `AU_BUF_SET_SIZE`: dient der Rekonfigurierung der Puffergröße. Als drittes Argument wird die neue Größe des Puffers erwartet,

- `AU_BUF_SET_WM`: verändert die Watermark des Puffers. Die Watermark muss als Prozentzahl der insgesamt verfügbaren Pufferelemente angegeben werden,
- `AU_DYN_BUF_SET_WM`: erlaubt das Setzen der Watermark des dynamischen Zusatzpuffers. Das Argument enthält die Anzahl der permanent im Speicher zuhaltenden Pufferelemente,
- `AU_DYN_BUF_SET_ORDER`: dient der Rekonfigurierung der Größe der angeforderten Speicherseiten. Das übergebene Argument sollte eine Zahl zwischen Null und Fünf sein. Eine genaue Erläuterung des Parameters erfolgte bereits im Abschnitt 4.1.2.2.

Die Verwaltung des Audit-Puffers muss mit dem gleichen Programm erfolgen, welches später die Protokollierung aktiviert. Anderenfalls wird der Zugriff auf den Puffer verweigert.

Aktivierung und Deaktivierung des Audit-Systems

Für die Aktivierung und die Deaktivierung der Protokollierung werden die Funktionen `audit_on(...)` und `audit_off(...)` bereitgestellt.

Die Funktion `audit_off(...)` erwartet keine Parameter und deaktiviert die Protokollierung des Systems. Die Puffer des Audit-Systems werden nicht deaktiviert und der von ihnen benötigte Speicher wird nicht freigegeben. Audit-Events, die bis zu diesem Zeitpunkt noch nicht gespeichert wurden, verbleiben im Audit-Puffer.

Die Funktion `audit_on(...)` erwartet zwei Argumente: einen Zeiger auf einen String-Puffer, der für die Speicherung der Audit-Records benötigt wird (siehe Abschnitt 4.1.2.5), und die Größe des String-Puffers. Wurden die Audit-Puffer zuvor erfolgreich eingerichtet, sollte nach dem Aufruf der Funktion `audit_on(...)` die Protokollierung des Systems laufen. Dies kann mittels des Eintrages `audit` im Wurzelverzeichnis des `proc`-Dateisystems überprüft werden.

4.2.2 ACL-Submodul

Das ACL-Submodul unterstützt zwei Operationen: das Einfügen von Regeln und das Erfragen einer Selektionsmaske der Audit-Struktur eines Prozesses. Für HoNA-Events ist bisher nur das Einfügen von Regeln möglich. Bereits im Abschnitt 4.1.4 wurde beschrieben, dass die ACL-Regeln für IPv4- und IPv6-Adressen getrennt verwaltet werden. Schon in der Audit-Bibliothek sind für beide Adressfamilien verschiedene Funktionen implementiert. Die richtige Interpretation der neuen Regel obliegt demzufolge dem Anwendungsprogramm.

Einfügen von Regeln

Beim Einfügen der Regeln muss zwischen IPv4- und IPv6-Adressen sowie zwischen direkten Adressen und IP-Bereichen unterschieden werden. Somit sind vier verschiedene Funktionen zum Einfügen der Regeln notwendig. Die Varianten der IPv4- und der IPv6-Funktionen unterscheiden sich lediglich im Datentyp der übergebenen Adresse, ansonsten ist deren Schnittstelle identisch.

Einfügen von direkten Adressen

Für das Einfügen einer Regel, die sich auf eine IP-Adresse bezieht, werden die beiden Funktionen `acl_set_ipv4(...)` und `acl_init_ipv4(...)` bereitgestellt. Die Funktionen für IPv6-Adressen enden dementsprechend auf `_ipv6`.

`acl_set_ipv4(...)`

Die Funktion `acl_set_ipv4(...)` dient dem Aktivieren bzw. Deaktivieren der Protokollierung eines HoNA-Events innerhalb des ACL-Baums. Sie erwartet als Parameter

einen Zeiger auf die IP-Adresse, den CIDR-Block, die Ports, deren Richtungsoperator, die Major- und die Minor-Event-ID und die Protokollierungsart.

Der CIDR-Block muss für IPv4-Adressen zwischen Null und 32 liegen und für IPv6-Adresse zwischen Null und 128.

Für die Angabe der Portnummern ist der untere sowie der obere Grenzwert des Bereichs der Ziel- und der Quellportnummern notwendig. Sollen alle Portnummern adressiert werden, muss der Bereich entsprechend mit 0 und 65536 angegeben werden. Wird bei der Protokollierung des Events die Portnummer nicht mit einbezogen, werden die Parameter der Portnummern ignoriert.

`acl_init_ipv4(...)`

Mittels der Funktion `acl_init_ipv4(...)` kann die komplette Selektionsmaske eines ACL-Objektes gesetzt werden. Sie wird hauptsächlich für die Initialisierung einzelner Objekte verwendet. Die Argumente der Funktion sind identisch mit denen der Funktion `acl_set_ipv4(...)`, lediglich die Parameter für die Angabe der Major- und Minor-Events sowie die Protokollierungsart wird durch einen Zeiger auf eine Selektionsmaske ersetzt.

Einfügen von Adressebereichen

Für das Einfügen von Regeln, die sich auf einen Bereich von IP-Adressen beziehen, werden die Funktionen `acl_set_ipv4_rng(...)` und `acl_init_ipv4_rng(...)` bereitgestellt. Die Bedeutung und die Parameter der Funktionen sind identisch mit denen der Funktionen für einzelne IP-Adressen. Es wird lediglich ein zusätzlicher Parameter, der das Ende des Adressbereiches kennzeichnet, als zweites Argument übergeben.

4.3 Nutzerapplikationen

Zur Nutzung des Audit-Systems ist die Implementierung von grundlegenden Nutzerprogrammen erforderlich. In Abbildung 8 ist eine Übersicht über das gesamte Audit-System und das Zusammenwirken der einzelnen Komponenten dargestellt. Die notwendigen Programme wurden zum Großteil aus der LOSA-Implementierung übernommen. Beim Arbeiten und beim Weiterentwickeln des Systems hat sich gezeigt, dass einzelne Bestandteile noch einmal verändert werden mussten. In diesem Abschnitt erfolgt die Beschreibung der für HoNA notwendigen Erweiterungen an diesen Komponenten. Darüber hinaus wird kurz auf Änderungen, die im Rahmen der HoNA-Implementierung an den Programmen vorgenommen wurden, eingegangen.

4.3.1 Der Audit-Dämon `auditd`

Die notwendigen Änderungen am Audit-Dämon beschränken sich fast ausschließlich auf die Integration des neuen Puffermanagements. Hierzu mussten die ursprünglich verwendeten Funktionen durch die in Abschnitt 4.2.1 beschriebenen Funktionen ersetzt werden. Dies führte wiederum dazu, dass der Ablauf der Initialisierung des Systems verändert werden musste. Entsprechend den Anforderungen konfiguriert der Audit-Dämon nun zunächst die Audit-Puffer, bevor er die Protokollierung mittels `audit_on(...)` aktiviert.

Darüber hinaus war es notwendig, die Grammatik der Konfigurationsdatei und der Shell-Kommandos geringfügig zu verändern. Die neue Grammatik unterstützt die Konfiguration von verschiedenen Record-Typen. Alle übrigen Parameter blieben unverändert, da sich die zugehörigen Komponenten nicht geändert haben.

4.3.2 Das Steuerungsprogramm `aca`

Das Steuerungsprogramm ist das allgemeine Konfigurationsinterface zum Audit-Modul. Wie bereits im Abschnitt 3.2.2 beschrieben, fasst das Steuerungsprogramm die Funktionalität des ursprünglichen Steuerungsprogramms mit der des Applikation-Wrappers und des Konfigurationsoptimiers zusammen. Damit muss nur noch ein Programm gepflegt werden. Zudem erfordert die Zusammenführung der Programme eine einheitliche Grammatik für die Shell-Kommandos und die Konfigurationsdateien.

Im Folgenden wird zunächst auf die Integration der neuen ACL-Regeln eingegangen. Anschließend werden die Änderungen, die für die Zusammenführung der Programme notwendig waren, beschrieben.

4.3.2.1 Integration der HoNA-Regeln

Bevor eine Regel in das Audit-Modul übertragen werden kann, muss sie interpretiert und auf syntaktische Korrektheit überprüft werden. Beides sind Aufgaben des Steuerungsprogramms. Für die Überprüfung der syntaktischen Korrektheit wurde mittels der Programme `lex` und `yacc` eine Grammatik erstellt. Die Beschreibung der Grammatik ist in den beiden Dateien `lexprog.l` und `yaccprog.y` zu finden. Die Daten werden mittels der Programme `lex` und `yacc` in Programmcode der Programmiersprache C übersetzt. Alle Regeln, die dieser Grammatik genügen, werden in ihre Komponenten zerlegt und anschließend als Parameter den Bibliotheksfunktionen übergeben.

Enthalten die Regeln Aufzählungen von Event-IDs, IP-Adressen oder Portnummern, so müssen diese Regeln zerlegt und einzeln in den Kernel übertragen werden. Die Funktionen der Audit-Bibliothek erlauben nur die Angabe von Bereichen von IP-Adressen, Portnummern und Event-IDs, so dass alle anderen Arten von Aufzählungen und Bereichen nicht direkt in das Audit-Modul übertragen werden können. Sind die Aufzählungen von IP-Adressen und Portnummern fortlaufend, d.h. könnten sie eigentlich als Bereiche zusammengefasst werden, so wird dies vom Steuerungsprogramm nicht ausgeführt. Dies hat zwei Gründe, zum einem nimmt das Programm dem Administrator nicht die gesamte Arbeit ab und zum anderen kann es nicht entscheiden, ob die Angabe einzelner Adressen vom Administrator nicht explizit gewollt ist.

Neben der Auflösung von Aufzählungen ist das Steuerungsprogramm für die Umwandlung von Schlüsselwörtern wie `any` verantwortlich. Diese müssen vor dem Aufruf der Bibliotheksfunktionen in ihren entsprechenden numerischen Werte umgewandelt werden. Dies bedeutet für:

- *die Event-ID*: die Verwendung einer kompletten Selektionsmaske,
- *die Major- oder Minor-Event-ID*: die Umwandlung in eine Aufzählung der einzelnen Event-IDs,
- *die IP-Adresse*: die Verwendung der Adresse 0.0.0.0/0 für IPv4 und ::/0 für IPv6,
- *den Port-Eintrag*: die Umwandlung in den Bereich von 0 bis 65536 für die Ziel- und die Quellportnummern, sowie deren Bi-direktionale Verwendung und
- *die Ziel- oder die Quelleportnummer*: dessen Konvertierung in den Bereich von 0 bis 65536.

Bisher können die HoNA-Regeln nicht in den Konfigurationsdateien verwendet werden. Sie müssen manuell mittels des Steuerungsprogramms in das Audit-Modul übertragen werden.

4.3.2.2 Integration des Applikation-Wrappers und des Konfigurationsoptimiers

Wie bereits erwähnt, erleichtert die Integration des Applikation-Wrappers und des Konfigurationsoptimiers die Wartung und Erweiterung des Programmpaketes. Am Funktionsumfang

der Programme hat sich nichts geändert, einzig die Grammatik der Konfigurationsdatei, die durch den Konfigurationsoptimiers verarbeitet wurde, musste geringfügig verändert werden.

Bisher wurden für die Konfigurationsdateien und für die Shell-Kommandos zur Konfigurierung der ACL-Regeln verschiedene Grammatiken mit gleichen Schlüsselwörtern verwendet. Da dies bei der Integration in ein Programm nicht mehr möglich und an sich nicht besonders bedienerfreundlich ist, wurden die Grammatiken zusammengeführt. Dies hat zudem den Vorteil, dass Sprachelemente, die zuvor nur in der Konfigurationsdatei verwendet werden konnten, jetzt auch in der Shell ausgeführt werden können.

4.3.3 Der Trail-Viewer `atv`

Der eigentliche Beweggrund zur Implementierung des Trail-Viewers war die Notwendigkeit eines Test-Clients zur Darstellung der Audit-Daten. Die Implementierung der einzelnen Komponenten erfolgte immer dann, wenn bestimmte Teile oder Protokolle in das Audit-Modul bzw. in die Audit-Bibliothek integriert wurden und getestet werden mussten. Allerdings ließ sich die korrekte Funktionsweise einzelner Audit-Funktionen leicht mittels eines Editors anhand des Binär-Codes des Audit-Trails überprüfen, so dass auf die Implementierung im Trail-Viewer verzichtet werden konnte. Aus diesem Grund ist die Vollständigkeit des Trail-Viewers bzgl. der Darstellung aller Audit-Events nicht immer gegeben.

Die Erweiterung des Trail-Viewers beschränkt sich auf die Darstellung einzelner HoNA-Events. In der derzeitigen Implementierung können lediglich die Header-Informationen des Ethernet-Protokolls und der Protokolle IPv4, TCP, UDP und ICMP sowie der IPSec-Erweiterung dargestellt werden. Suchfunktionen für HoNA-Events sind bisher nicht implementiert. Die angebotenen Suchfunktionen beschränken sich auf die Audit-Records des Host-orientierten Audits.

Kapitel 5

Leistungsfähigkeit der Implementierung

Nachdem im Kapitel 3 das Design eines Kern-integrierten Netzwerk-Monitors beschrieben und im Kapitel 4 auf dessen Implementierung eingegangen wurde, erfolgt in diesem Abschnitt eine Bewertung der Leistungsfähigkeit der Implementierung. Um die Leistung besser bewerten zu können, werden die Ergebnisse von verschiedenen Monitoren verglichen. Als Vergleichsobjekte dienten die Programme `ethereal` und `snort`, welche nur den jeweils lokalen Netzverkehr eines Netzwerk-Interfaces aufzeichneten bzw. überwachten (*non-Promiscuous* Mode).

Zunächst erfolgt eine Bewertung der Performance insbesondere des Datendurchsatzes und die dabei verursachte Auslastung des Systems bei der Übertragung einer regulären Datei. Anschließend werden verschiedene Angriffe gegen ein überwachtes System ausgeführt und die protokollierten Daten untersucht.

5.1 Untersuchung einer regulären Datenübertragung

Zur Messung des Datendurchsatzes wurde eine Datei mittels des *File Transfer Protocol* (FTP) übertragen. Die Datei `foo` befand sich jeweils lokal auf den Testsystemen im `tmp`-Verzeichnis, so dass keine zusätzlichen Belastungen auf dem Netzwerk entstehen konnten.

Eine Beschreibung der verwendeten Systeme und des Aufbaus des Testnetzes ist im Anhang B zu finden. Im Anhang B sind ebenso die verwendeten LINUX- und Softwareversionen aufgeführt.

Bei der verwendeten Datei handelte es sich um einen 105.013.300 Byte großen Audit-Trail. Die Datei war kleiner als der Dateisystempuffer auf den einzelnen Systemen, so dass nicht mit Sicherheit gesagt werden kann, ob die Datei wirklich auf die Festplatte geschrieben wurde. Da der Test zudem mehrfach wiederholt wurde, kann mitunter davon ausgegangen werden, dass ein nicht unwesentlicher Teil der Datei nur zwischen den Systemspeichern der Rechner übertragen wurde. Demzufolge spielen die verwendeten Festplatten nur eine untergeordnete Rolle. Die Messdaten aus Tabelle 2 entsprechen damit im Wesentlichen der Zeit, die die Systeme benötigt haben, um die Daten durch den Netzwerk-Stack zu verarbeiten und zu analysieren. Bei den angegebenen Werten handelt es sich um die Übertragungszeit, die vom ftp-Client nach Abschluss der Übertragung ausgegeben wird.

Übertragungsrichtung	ohne Monitor	HoNA	`Snort`	`ethereal`
System1* ⇒ System3	22.2 sec	21.1 sec	24,5 sec	32,3 sec
System3 ⇒ System1*	9,55 sec	9,77 sec	9,82 sec	10,4 sec
System2* ⇒ System3	30,4 sec	31,6 sec	40,8 sec	88,5 sec
System3 ⇒ System2*	27,7 sec	28,8 sec	30,3 sec	44,8 sec

Tabelle 2 : Messergebnis des Datendurchsatzes

Auf den in Tabelle 2 mit einem * gekennzeichneten Systemen lief jeweils der Netzwerk-Monitor. Als Client-System wurde immer das System 2 verwendet, um eine bessere Vergleichbarkeit der Messdaten zu erreichen.

HoNA

Wie aus der Tabelle 2 ersichtlich ist der Datendurchsatz des Systems, auf dem der Kern-integrierte Monitor lief, vergleichbar mit dem Durchsatz des Systems ohne Monitor. Dies gilt sowohl für das leistungsstarke System 1 als auch für das Notebook (System 2). Das Ergebnis ist nicht überraschend, da während der Übertragung nur sehr wenige Audit-Events protokolliert wurden. Der Großteil der protokollierten Ereignisse ist dem Host-orientiert Audit zuzuordnen und damit unabhängig von HoNA.

Dadurch dass kaum Audit-Events protokolliert wurden, war auch ein Anstieg bei der CPU-Auslastung nicht messbar.

Im Audit-Trail sind drei wesentliche Ereignisse zu finden. Das erste protokollierte Event weist darauf hin, dass eine TCP-Verbindung etabliert wurde. Durch das zweite Event wird sichtbar, dass der ftp-Dämon die Verbindung entgegengenommen hat. Der daraufhin gestartete `ftp`-Dämon hat die Datei `/tmp/foo` geöffnet. Diese Events zeigen eindeutig: Wer, was, wann, wie mit welcher Ressource ausgeführt hat.

snort

Das IDS `snort` erzeugt einen wesentlich höheren Overhead als HoNA. Dies ist leicht zu begründen, da die Pakete durch `snort` ein zweites Mal untersucht werden müssen und damit nahezu der doppelte Aufwand für die Bearbeitung eines Paketes anfällt. Einen nicht unwesentlichen Anteil an der Performance des IDS hat die Größe der Signatur-Datenbank. Im Test wurde die Datenbank des Originalpaketes verwendet, sie enthält ungefähr 700 Regeln. Mit zunehmender Größe der Datenbank ist anzunehmen, dass sich die Gesamtperformance des Systems weiter verschlechtern wird.

Die CPU-Auslastung bei der Übertragung der Datei lag weit über dem normalen Wert. Auf dem Dual-System war eine CPU fast vollständig ausgelastet und auf dem Notebook war das gesamte System komplett ausgelastet.

In den Protokolldaten ist nur wenig zu finden. `snort` protokolliert mit den Standard-Signaturen nicht den Aufbau einer regulären FTP-Verbindung. Ebenso bleibt die Übertragung der Datei unbemerkt. Sicherlich lässt sich eine Regel definieren, die den Verbindungsaufbau anzeigt, jedoch bleibt das Was, nämlich welche Datei übertragen wurde, schwer nachweisbar.

ethereal

Der Netzwerk-Monitor `ethereal` erzeugte den größten Overhead bei der Protokollierung des Paketstroms. Die Analyse der einzelnen Pakete erfolgt mittels sogenannter Analyse-Module, die einzeln freigeschalten werden können. Je nachdem, wie viele von diesen Modulen aktiviert sind, verändert sich der Ressourcenbedarf bei der Auswertung eines Paketes. In der Standardeinstellung sind alle Module aktiviert, so dass ein sehr großer Ressourcenbedarf bei der Analyse des Paketstroms notwendig ist. In manchen Fällen wurde mehr als die doppelte Zeit zur Übertragung der Datei benötigt. Werden keine Filter definiert, speichert das Programm `ethereal` zudem alle Pakete auf der Festplatte ab, wodurch ein zusätzlicher Overhead ersteht. `snort` untersucht zwar auch alle Pakete, verwirft sie jedoch, wenn er mit ihrer Bearbeitung fertig ist.

Durch die Verwendung einer graphischen Benutzeroberfläche und insbesondere durch die vollständige Analyse des Paketstroms ist das System vollständig ausgelastet, so dass auf dem Notebook kaum weitere Arbeiten ausgeführt werden können. Auf dem Dual-System ist nur ein Prozessor vollständig ausgelastet, die andere CPU ist nur zu 10 Prozent belegt, was ungefähr der CPU-Last zum Übertragen

der Datei entspricht. Doch selbst dies stellt eine Mehrbelastung dar, die in den wenigsten Systemumgebungen so hingenommen werden kann. Läuft das Dual-System beispielsweise zusätzlich als Server, können die Dienste nur mit einer sehr eingeschränkten Leistung angeboten werden.

Die protokollierten Daten geben ausführlich Auskunft über die Datenübertragung. So lässt sich neben dem Dateinamen und dem Benutzernamen sogar das übertragene Passwort ermitteln. Ob dies gewünscht ist, bleibt dem Administrator überlassen. Jedoch ist die Menge der protokollierten Daten so groß, dass sich die wesentlichen Informationen nur schwer ermitteln lassen.

Fazit

Beim regulären Datenverkehr zwischen zwei Rechnern kann mitunter ein weitaus höheres Datenaufkommen als im Test auftreten, so dass Netzwerk-Monitore wie `ethereal` eine unvertretbare Mehrbelastung auf dem System erzeugen. Das IDS `snort` begrenzt durch die Signaturen zwar die Menge der protokollierten Daten, jedoch besteht dabei immer die Gefahr, dass wesentliche Informationen verloren gehen. Zudem ist die CPU-Auslastung auch bei `snort` sehr hoch, so dass das System bei hohem Datenaufkommen und vielen Signaturen schnell an seine Grenzen stößt.

Wirkungslos sind die Programme `snort` und `ethereal`, wenn die Daten verschlüsselt übertragen werden. Sie sehen nur AH- und ESP-Pakete und können keine Auskunft über den Inhalt des Datenstroms geben. Mittels HoNA können die entschlüsselten Paketinhalte untersucht werden, außerdem erhält man mitunter noch Informationen über die Protokollabläufe innerhalb von IPSec.

5.2 Beispielangriffe

Um den Aufzeichnungsumfang der einzelnen Monitore bewerten zu können, wurden eine Reihe von Beispielangriffen durchgeführt und anschließend die protokollierten Informationen verglichen. Dieser Abschnitt beschreibt die Ergebnisse des Tests. Als Vergleichsobjekte wurden wiederum die Programme `snort` und `ethereal` herangezogen.

5.2.1 Port-Scans

Für die Port-Scans wurde das Programm `nmap`, welches im Internet unter [23] heruntergeladen werden kann, verwendet. Das Programm ist in der Lage eine Vielzahl verschiedener Scans durchzuführen und die Quelladresse zu verbergen. Ausgeführt wurde jeweils ein SYN- und ein FIN-Scan. Eine Beschreibung der Scans ist im Anhang A.1 zu finden.

ethereal

Das Programm `ethereal` protokolliert alle Pakete, so dass bereits durch die Vielzahl der eingetroffenen Pakete leicht erkannt werden kann, dass der Rechner möglicherweise gescannt wurde. Sucht man mit Hilfe der vorhandenen Funktionen nach SYN- und SYN-ACK-Paketen, wird das Ergebnis noch deutlicher.

Dadurch, dass `ethereal` das gesamte Paket protokolliert, ist zusätzlich der Ethernet-Header sichtbar. Hat man die Möglichkeit die Ethernet-Adresse mit der IP-Adresse zu vergleichen[18], wird zudem deutlich, dass die Quelladresse manipuliert wurde.

[18] Dies ist natürlich nur möglich, wenn sich Angreifer und Opfer im selben Netzsegment befinden. Wurden die Pakete auf ihren Weg zum Ziel geroutet, ist es unmöglich ein verfälschen der IP-Adresse zu erkennen.

Die Menge der protokollierten Daten ist sehr groß, so dass eine Scan-Attacke schnell in der Menge der Pakete untergehen kann. Mit entsprechenden Filtern kann das Problem geringfügig umgangen werden. Jedoch besteht dann immer die Gefahr, andere Sicherheitsverletzungen durch das Hinzufügen von Filtern zu übersehen.

snort

Das IDS `snort` verfügt über einen separaten Präprozessor zum Erkennen von Scan-Attacken. Dieser legt die protokollierten Daten in einer separaten Datei ab. Dort wird jeweils ein Eintrag für jedes eingetroffene Paket abgelegt. Dieser besteht aus der Quell- und Zieladresse, dem Protokoll, einem Zeitstempel und dem gesetzten Flag im TCP-Header. Eine Möglichkeit den Ethernet-Header zu sehen, besteht nicht, so dass gefälschte Quelladressen nicht erkannt werden können.

Durch die Begrenzung der protokollierten Daten auf die Quell- und die Zieladresse, das Protokoll, sowie einen Zeitstempel und den Flags ist das Datenaufkommen bereits stark reduziert und kann somit auch über einen längeren Zeitraum gespeichert werden.

HoNA

Die Informationen, die aus den Trails von HoNA gewonnen werden können, sind vergleichbar mit den Informationen von `ethereal`. Im Trail sind auffallend viele Minor-Events vom Typ `AUSE_TCP_NO_SOCK` zu finden. Jeder Audit-Record dieses Events enthält den Ethernet-, den IP- und den TCP-Header, sofern das Paket auf einem Ethernet-Device eingetroffen ist. Damit lässt sich leicht erkennen, welcher Scan-Typ durchgeführt wurde und unter Umständen, ob die Quelladresse verfälscht wurde.

Obwohl im Audit-Record die kompletten Paket-Header enthalten sind, ist die Menge der protokollierten Daten nicht wesentlich größer als bei `snort`. Das IDS `snort` konvertiert die protokollierten Daten in eine lesbare Form und fügt zusätzliche Beschreibungen und Textformatierungen hinzu, auf die bei der binären Ablage im Audit-Trail mittels HoNA verzichtet wird.

5.2.2 DoS-Attacken

Es war leider kein freiverfügbares Programm zu finden, dass all die hier aufgeführten DoS-Attacken ausführen konnte. Unter Windows kann das Programm *CyberCop Scanner* der Firma *Network Associates Inc.* verwendet werden. Jedoch ist dieses nicht freiverfügbar und auch nicht sonderlich preiswert, so dass im Rahmen dieser Arbeit auf mehrere kleinere Programme zurückgegriffen werden musste. Die Programme für die hier durchgeführten DoS-Attacken sind in [10] zu finden. Eine kurze Beschreibung der Attacken erfolgt im Anhang A.2.

ethereal

Das Programm `ethereal` unterscheidet keine Pakete hinsichtlich einer sicherheitskritischen Relevanz, so dass die durchgeführten Attacken anhand der übertragenen Pakete selbst erkannt werden müssen. Hierzu sind fundierte Kenntnisse über die Angriffe selbst notwendig.

Land

Die Land-Attacke basiert auf der Übertragung von Paketen mit gleicher Quell- und Zieladresse. Diese Pakete können innerhalb des Datenstroms erkannt werden, wie dies geschieht, ist dem Anwender überlassen und aufgrund der Menge der Daten nicht leicht.

Ping of Death

Für das Erkennen dieser Attacke muss nach fragmentierten IP-Paketen gesucht werden. Die Attacke kann aber erst dann erkannt werden, wenn man die Fragmente zusammenfügt, was das Programm `ethereal` nicht tut. So muss man sagen, dass dieser Angriff ohne eine tiefergehende Analyse nicht erkannt werden kann.

Teardrop2

Beim Erkennen einer teardrop-Attack wird man vor das gleiche Problem wie beim Ping of Death gestellt. Die UDP-Pakete müssen erst zusammengesetzt oder zumindest muss die Korrektheit des übermittelten Offsets überprüft werden[19]. Da dies mittels `ethereal` nicht automatisch möglich ist, kann diese Attacke nur schwer erkannt werden.

SYN-Flooding

Da für ein SYN-Flooding nur SYN-Pakete verwendet werden und sich der Angriff nicht über mehrere Paketfragmente hinzieht, kann dieser Angriff innerhalb des Paketstroms leicht erkannt werden. Das Resultat ist ähnlich einem SYN-Scan.

snort

Der Aufzeichnungsumfang von `snort` ist direkt abhängig von der Signatur-Datenbank und der Konfiguration. Wie bereits erwähnt, wurde für die durchgeführten Tests die Signatur-Datenbank von der `snort`-Homepage verwendet. Für die meisten Anwender wird dies der Einstieg sein und wie sich im Test gezeigt hat, ist der Aufzeichnungsumfang bereits sehr umfangreich. Es wird eher mehr aufgezeichnet als gewünscht, so dass die Regeln vom Nutzer diesbezüglich nachgebessert werden müssen.

Alle protokollierten Daten, die nicht zu einem Port-Scan gehören, werden anhand der IP-Adresse des Absenders sortiert. Für jeden Rechner wird ein separates Verzeichnis angelegt, wo die einzelnen Angriffstypen oder Unstimmigkeiten in einzelnen Dateien abgelegt werden.

Land

Wie bereits erwähnt zeichnet sich Land durch Pakete mit gleicher Quell- und Zieladresse aus. Diese werden von `snort` nicht direkt erkannt. Das betroffene System sendet beim Eintreffen eines solchen Paketes eine `ICMP_PORT_UNRCH`-Nachricht, die von `snort` protokolliert wird.

In den Log-Dateien findet man den Protokoll-Header des Paketes, welches das Senden einer `ICMP_PORT_UNRCH`-Nachricht verursacht hat. Dort wird schnell deutlich, dass die Ziel- mit der Quelladresse identisch ist, was wiederum ein eindeutiges Zeichen für eine mögliche Land-Attack ist. In dem protokollierten Header findet man unter Umständen auch die MAC-Adresse des Absenders und kann diesen ausfindig machen.

Ping of Death

Obwohl ICMP keine Ports unterstützt, scheint ein Rechner, der übergroße ICMP-Pakete verarbeiten kann, diese mit einer `ICMP_PORT_UNRCH`-Nachricht zu beantworten. Demzufolge ist ein *Ping of Death*-Angriff auch innerhalb der Log-Daten vom `snort` so zu erkennen. Wiederum werden die Protokoll-Header vollständig abgespeichert, so dass sie für eine spätere Analyse hinzugezogen werden können.

---

[19] Zwar wird in [5] gesagt, dass ein Paket mit der ID 242 im IP-Header eindeutig auf ein teardrop-Attack hindeutet, jedoch konnte dies im Test nicht nachvollzogen werden. Zudem ist es sehr fragwürdig, ob irgend eine Regelmäßigkeit anhand der ID im IP-Header erkannt werden kann.

Teardrop2

Eine teardrop-Attack wird von `snort` ebenfalls in Form einer `ICMP_PORT_UNRCH`-Nachricht protokolliert. Eine genaue Analyse der Ursache muss wiederum anhand der Protokoll-Header erfolgen.

SYN-Flooding

Für eine erfolgreiche Durchführung einer SYN-Flooding Attacke wird eine Verbindung zum Internet oder zu einem ausreichend großem LAN benötigt, da die Quelladressen variieren müssen, damit das Angriffsziel die Pakete nicht bereits vorzeitig verwirft. In den Protokoll-Daten vom `snort` ist ein SYN-Flooding wiederum nur über Umwege zu erkennen. Zum einen ist auffällig, dass der Rechner sehr viele Verbindungsanforderungen von verschiedenen Rechnern hatte. Dies ist leicht anhand der Vielzahl der Verzeichnisse im `snort`-Log-Verzeichnis zu erkennen. Des weiteren scheint das Angriffsziel von diesen Rechnern keine Bestätigung für den Verbindungsaufbau zu erhalten, so dass `snort` sehr viele ICMP-Nachrichten der Typen `ICMP_HST_UNRCH` und `ICMP_TTL_EXCEED` protokolliert.

HoNA

Das Konzept von HoNA ist so ausgelegt, dass es nur Fehler bzw. auffällige Ereignisse innerhalb des Netzwerk-Stacks protokolliert, so das davon ausgegangen werden muss, dass der Umfang der protokollierten Daten weitaus geringer ist. Ziel des Tests war es zu überprüfen, ob die protokollierten Daten ausreichen, um eine ähnliche Aussage treffen zu können, wie es bei den anderen Monitoren möglich war.

Mittels der Konfiguration des Kernels wird festgelegt, wie die ACL-Datenbank initialisiert werden soll. So der Nutzer es nicht anders wünscht, wird per Voreinstellung alles protokolliert. Damit kann nicht das Problem entstehen, dass bei der Standardkonfiguration bereits wichtige Ereignisse deaktiviert sind. Der Nutzer wird eher wie bei der Nutzung von `snort` zusätzliche Ereignisse deaktivieren müssen, damit der Umfang der protokollierten Daten nicht zu groß wird.

Land

Die Land-Attack kann eindeutig anhand der protokollierten `AUSE_IP_BAD_SRC`-Events erkannt werden. Für jedes eingetroffene Paket wird ein solches Audit-Event abgelegt. Innerhalb des Audit-Records findet man die übrigen Protokoll-Header, so dass unter Umständen auch der richtige Absender ermittelt werden kann.

Ping of Death

Ein Paket der Ping of Death Attacke entscheint im Audit-Trail als reguläres ICMP-Paket. Da die reale Größe durch die Verwendung eines `short int` –Wertes abgeschnitten wird, kann das Paket zu einem späteren Zeitpunkt nicht als Ping of Death Attacke wiedererkannt werden. Da jedoch jedes ICMP-Paket in einer konfigurierten Netzwerkumgebung unüblich ist, sollte das Paket auffallen.

Teardrop2

Pakete der teardrop-Attacke haben eine manipulierte Offset-Adresse innerhalb des IP-Headers. Dies wird vom Netzwerk-Stack erkannt und mittels HoNA im Audit-Trail mit einem `AUSE_UDP_BAD_SIZE`-Event vermerkt. Tritt ein solches Event auf, besteht immer der begründete Verdacht auf eine teardrop-Attacke. Mittels der übrigen Daten des Audit-Records und der anderen Audit-Events kann dieser Verdacht weitergehend verifiziert werden. In den durchgeführten Tests waren immer mehrere manipulierte UDP-Pakete übertragen worden, so dass auch durch die Häufigkeit des Ereignisses der Verdacht noch weiter bestätigt werden kann.

SYN-Flooding

Die Beispielangriffe bzgl. des SYN-Floodings konnten mittels HoNA nur schwer erkannt werden. Das Problem liegt wohl auch darin begründet, dass die Angriffe niemals erfolgreich waren, so dass das eigentlich Event, welches auf ein SYN-Flooding hindeutet, nie protokolliert wurde.

Vielmehr waren häufig Audit-Records mit den Minor-Events `AUSE_IP_BAD_SRC` und `AUSE_ICMP_REQ` zu finden. Ersteres trat vor allen Dingen immer dann auf, wenn keine Verbindung zum Internet bestand und die Quelladressen in den Paketen des Angreifers nicht aufgelöst werden konnten.

5.2.3 Spoofing

Für das IP-Spoofing wurde `mendax` verwendet. Wie in Anhang A.3 beschrieben, führt das Programm zunächst ein Sequence Number Sampling aus, um an die notwendigen Sequenznummern zu gelangen.

Sequence Number Sampling

Beim Sequence Number Sampling werden RST-Pakete verwendet, um an die notwendigen Sequenznummern zu gelangen. Sowohl beim Programm `ethereal` als auch bei HoNA fallen diese Pakete auf. HoNA protokolliert dies mittels des Events `AUSE_TCP_BAD_REQ`. Bei `ethereal` müssen die Pakete aus dem Datenstrom herausgesucht werden.

In der Signatur-Datenbank von `snort` existiert direkt die Regel IDS007 zur Erkennung des Sequence Number Sampling. Ob diese Regel möglicherweise auch auf andere Angriffe zutrifft, konnte bei den durchgeführten Tests nicht ermittelt werden. Es ist jedoch davon auszugehen, dass dies nicht so ist, da dies einzige anhand eines RST-Paketes, welches an einen geschlossenen Port gerichtet ist, erkannt werden kann.

mendax

Das eigentliche Spoofing, nämlich der Aufbau einer TCP-Verbindung, konnte nicht erfolgreich durchgeführt werden. Bei der verwendeten LINUX-Versionen konnten die Sequenznummern nicht vorgesagt werden, so dass der Aufbau einer TCP-Verbindung immer fehlgeschlagen ist. In den Protokolldaten sind demzufolge auch keine Informationen enthalten, die die Durchführung eines derartigen Angriffs bestätigten.

Kapitel 6

Zusammenfassung und Ausblick

Als Abschluss dieser Arbeit erfolgt noch einmal eine Zusammenfassung der gesammelten Erfahrungen und eine Bewertung des HoNA-Konzeptes. Im Kapitel 5 wurden bereits andere Konzepte mit HoNA verglichen und deren Leistungsfähigkeit bewertet. In diesem Abschnitt wird noch einmal versucht dies etwas differenzierter zu bewerten, bevor zu den möglichen Schwachstellen von HoNA und den offenen Arbeitspunkten übergegangen wird.

6.1 Bewertung des HoNA-Konzeptes

Ziel des HoNA-Konzeptes war die Erstellung eines Kern-integrierten Netzwerk-Monitors, der innerhalb des Netzwerk-Stacks des Betriebssystems mit einem möglichst geringem Overhead eine vollständige Protokollierung des Paketstroms gewährleistet. Das Gesamtkonzept ist dezentral konzipiert, so dass auf jedem sicherheitsrelevanten System ein Netzwerk-Monitor eingesetzt werden muss.

6.1.1 Integration in der BS-Kern

Nicht für jedes Betriebssystem lässt sich so leicht eine Integration von neuen Komponenten realisieren wie für LINUX. Eines der wesentlichen Probleme bei anderen Betriebssystemen ist das Fehlen des Quelltextes bzw. dessen schlechte Dokumentation. Oft sind dies die Geheimnisse der Hersteller und werden nur selten offengelegt. Eine der größten Stärken des Open-Source-Konzeptes ist die Möglichkeit, das System bzw. die Software den eigenen Anforderungen anpassen und dabei weit über die Nutzung von offenen Schnittstellen hinausgehen zu können.

Ein wesentlicher Teil von HoNA wurde in den BS-Kern integriert. Dies bietet einen größeren Schutz gegenüber potentiellen Angreifern, ermöglicht die Überwachung von verschlüsselten Protokollen ohne deren Integrität einzuschränken und spart durch die einmalige Auswertung des Paketes viele Ressourcen des Systems.

Schutz gegenüber potentiellen Angreifern

Alle Operationen des BS-Kerns laufen in einem geschützten Bereich und können durch Nutzeraktionen nicht beeinflusst werden. Die Funktionen von HoNA befinden sich im Netzwerk-Stack des BS-Kerns und sind damit unabhängig von allen Prozessen auf dem System. Einzig der Audit-Dämon, welcher zu Verwaltung der protokollierten Daten benötigt wird, kann durch einen Angreifer manipuliert werden. Ist es möglich, auch diesen aus dem Nutzerraum zu entfernen, besteht für einen Angreifer keine Möglichkeit, die Protokollierung des Systems zu manipulieren.

Überwachung von verschlüsselten Protokollen

In den vorangegangenen Abschnitten wurde bereits oftmals auf die Problematik eingegangen, die durch die Verwendung von verschlüsselten Paketen entsteht. Es wurde erläutert, dass einzig durch die Integration eines Monitors in das Ver- bzw. Entschlüsselungsmodul des BS-Kerns eine Überwachung dieser Protokolle gewährleistet werden kann.

Ein unüberwindbares Problem ist hierbei der Ressourcenbedarf zur Ver- bzw. Entschlüsselung der Pakete. Klassische Monitore müssten jedes Paket ein zweites Mal entschlüsseln, um eine Aussage über den Inhalt des Paketes treffen zu können. Jedoch haben die Tests aus Kapitel 5 gezeigt, dass die Systeme bereits bei der Überwachung von unverschlüsselten Paketströmen stark ausgelastet sind.

Einmalige Auswertung der Pakete

Die Auswertung bzw. Verarbeitung eines Datenpaketes erfordert einen nicht unwesentlichen Ressourcenbedarf. Durch den Einsatz klassischer Monitore wird dieser noch einmal erhöht. Für produktive Systeme bedeutet dies eine starke Einschränkung ihrer Leistungsfähigkeit. Die Verwendung eines zentralen Monitors wurde bereits im Kapitel 3 bewertet. Es wurde aufgezeigt, dass bei der heutzutage verwendeten Netzwerktechnik Probleme entstehen, die die Fähigkeiten eines zentralen Monitors stark vermindern oder die Bandbreite des gesamten Netzwerkes an diesen bindet.

Neben der Einschränkung der Leistungsfähigkeit des Systems besteht zudem das Problem, dass ein externer Monitor ein Paket mitunter anders interpretiert als der BS-Kern oder Seiteneffekte entstehen, die von außen nicht berücksichtigt werden können. Nur durch die Integration des Monitors in die Funktionen des Netzwerk-Stacks kann eine Überwachung gewährleistet werden, die den aktuellen Zustand bzw. dessen Verhalten beschreibt.

6.1.2 Vollständigkeit der Überwachung

Klassische Netzwerk-Monitore, die im Nutzerspeicher des Betriebssystems eine Überwachung des Datenverkehrs durchführen, können nicht erreichen, dass die Paketverarbeitung auf sie abgestimmt wird. In der in [5] beschriebenen HoNA-Implementierung für Windows wird dieses Problem deutlich. Der Netzwerk-Monitor wurde als Applikation implementiert und konnte bereits bei einem Datenaufkommen von 32MBps nicht mehr alle Pakete verarbeiten. Das Problem wird besonders akut, wenn der Monitor asynchron zur eigentlichen Paketverarbeitung des BS-Kerns betrieben wird und dieser die Verarbeitungsleistung des Monitors nicht mit einbezieht.

Die Protokollierung mittels eines Kern-integrierten Monitors erfolgt in der Paketverarbeitung des Betriebssystems. Alle Datenpakete, die der Netzwerk-Stack verarbeitet, werden zwingend in die Protokollierung mit einbezogen. Damit kann sichergestellt werden, dass keine Pakete ausgelassen werden und erst dann kann man Aussagen darüber treffen, was auf dem System passiert ist. Geht nur ein Paket verloren, kann nicht mehr sichergestellt werden, dass das System nicht angegriffen oder Opfer einer Sicherheitsverletzung geworden ist. Aus diesem Grund sind unvollständige Log-Daten für eine sicherheitstechnische Überwachung nahezu nutzlos.

Im Abschnitt 4.1.3 wurde eingeräumt, dass auch beim Konzept von HoNA theoretisch die Möglichkeit besteht, dass Ereignisse verloren gehen, wenn kein Audit-Record bereitgestellt werden kann. In einem solchen Fall, werden die Daten dem syslog-Dämon übergeben, der über ein eigenes Puffermanagement verfügt und damit die Wahrscheinlichkeit, dass keine Informationen über dieses Ereignis verfügbar sind, noch weiter verringert. In den bisherigen Tests konnte jedoch mit ausreichend großen Puffern kein Angriff durchgeführt oder eine Situation hergestellt werden, in der ein Event verworfen werden musste, da kein Audit-Record mehr zur Verfügung stand.

6.1.3 Dezentraler Einsatz der Monitore

Durch die feste Integration von HoNA in den Netzwerk-Stack von LINUX ist eine Überwachung von Paketen, die an andere Rechner des Netzwerkes gerichtet sind, nicht mehr möglich. Die Funktionen des Stacks verarbeiten nur Pakete, die für den Rechner selbst bestimmt sind. Damit ist ein zentraler Einsatz eines HoNA-basierten Netzwerk-Monitors nicht möglich.

Die Vor- und Nachteile eines zentralen Monitors wurden bereits im Abschnitt 2.2.1 erläutert. Beim Einsatz von HoNA ist nur eine dezentrale Überwachung der Rechner möglich. Jedoch ist HoNA nur

für die Protokollierung der Daten verantwortlich und nicht für deren Auswertung. Hierzu muss eine externe Applikation, vorzugsweise ein Intrusion Detection System, herangezogen werden. Hierfür eignet sich bspw. das AID-System, welches eine verteilte Audit-basierte Echtzeiterkennung von IT-Sicherheitsverletzungen ermöglicht [4]. Das Konzept wurde im Rahmen eines Forschungsprojektes an der Brandenburgischen Technischen Universität am Lehrstuhl Rechnernetze und Kommunikationssysteme entwickelt. Dort wurde die erste Implementierung von HoNA für das BS Solaris 2.5.1 von SUN Microsystems in die Überwachung mit einbezogen. Es wurde gezeigt, wie auch mit dezentral gesammelten Audit-Daten eine Überwachung eines gesamtes Netzwerkes realisiert werden kann.

Durch den dezentralen Einsatz der Monitore und der Auswertung auf einem zentralen IDS entsteht ein fehlertoleranteres und leistungsfähigeres System als das eines zentralen Netzwerk-Monitors. Der Ausfall eines Monitors beeinträchtigt nur das System, auf dem der Monitor betrieben wird und durch die Begrenzung der Pakete auf die, die an den Rechner selbst gerichtet sind, kann eine weitaus größere Datenmenge in Echtzeit protokolliert und analysiert werden. Es muss lediglich sichergestellt werden, dass das System, auf dem das IDS betrieben wird, fehlertolerant und ausfallsicher gestaltet ist. Fällt das IDS zeitweise aus, sind die meisten Systeme dieser Art immer noch in der Lage, die Audit-Daten nachträglich zu analysieren und eine mögliche Sicherheitsverletzung anzuzeigen.

6.2 Schwachstellen des Konzeptes und der Implementierung

Wie bei allen Konzepten und Implementierungen existieren auch bei HoNA Punkte, die noch nicht zufriedenstellend sind und verbessert werden könnten. In diesem Abschnitt werden ein paar dieser Punkte angesprochen. Jedoch hat sich gerade bei der Arbeit mit LOSA gezeigt, dass erst nach einem längeren Nutzungszeitraum Problem oder Verbesserungsvorschläge auftreten, die man eigentlich beseitigen möchte. Eben aus dem Mangel an Erfahrungen mit HoNA für LINUX kann dieser Abschnitt nur einen unvollständigen Ausblick auf die offnen Punkte von HoNA geben.

Im Rahmen dieser Arbeit wurde ein Software-Paket erstellt, was aus einem Kernel-Patch und vielen einzelnen Programmen besteht. Die Programme stellen eine reine Eigenentwicklung dar und können separat gepflegt und weiterentwickelt werden. Doch der Kernel-Patch ist und wird immer an die Entwicklung des Kernels gebunden sein. Bei einzelnen Erweiterungen des Kernels kann die Erstellung eines neuen Patches mitunter sehr aufwendig werden. Die Schnittstellen zum Kernel sind nicht so sauber implementiert, dass sie eine einfache Integration ermöglichen. An vielen Stellen sind genaue Kenntnisse über die Implementierung notwendig, um einen neuen funktionsfähigen Patch zu erstellen. Ein verbleibender Arbeitspunkt ist aus diesem Grund die Erstellung einer sauberen Schnittstelle zum Kernel, die eine leichte Integration von Weiterentwicklungen auf beiden Seiten ermöglicht.

Neben der Implementierung von HoNA war einer der wichtigsten Arbeitspunkte die Analyse des Netzwerk-Stacks hinsichtlich sicherheitskritischer Punkte, die für eine Protokollierung geeignet sind. Hierbei wurden nur die Protokolle der TCP/IP-Familie mit einbezogen und auf der Host-an-Netz Schicht nur das Ethernet-Protokoll analysiert. Bereits bei den ersten Tests hat sich gezeigt, dass im Netzwerk-Stack Punkte existieren müssen, die zusätzliche Fehler abfangen, die vom Audit-Modul nicht protokolliert wurden. So wurden Pakete durch das System verworfen bzw. nicht bearbeitet, ohne dass in den Audit-Trails ein Record vermerkt wurde, der auf das warum hinwies. Weitere Analysen des Quelltextes des Netzwerk-Stacks sind mitunter unumgänglich, um den Aufzeichnungsumfang noch aussagekräftiger gestalten zu können.

Darüber hinaus fehlt noch eine wesentliche Protokolleinheit, die die Leistungsfähigkeit von HoNA stark erweitern würde. Bisher werden nur die Schichten eins bis drei in die Protokollierung mit einbezogen. Alle Anwendungsprotokolle sind von der Analyse ausgeschlossen. Hierzu sind Erweiterungen an den netzspezifischen Funktionen der Systemapplikationen notwendig, wie sie bereits für den Kernel durchgeführt wurden. Ein Großteil der Angriffe auf IT-Systeme bezieht sich nicht auf das BS selbst sondern auf die Applikationen, so dass eine Überwachung dieser als ebenso wichtig anzusehen ist.

Anhang A

Untersuchte Netzwerkattacken

Um den Aufzeichnungsumfang von HoNA bewerten zu können, wurde ein kleiner Teil möglicher Netz-basierter Angriffe ausgeführt. Dieser Abschnitt enthält eine Beschreibung der einzelnen Angriffe. Die Auswertung bzgl. HoNA ist im Abschnitt 5.2 zu finden.

A.1 Port-Scans

Die Protokolle TCP und UDP bilden die Grundlage für viele Anwendungen bzw. Dienste im Internet oder LAN. Angriffe gegen diese Dienste erfordern immer das Wissen, auf welchem Rechner welcher Dienst momentan angeboten wird. Um dies zu ermitteln, werden sogenannte Port-Scans eingesetzt. Jeder aktive Dienst öffnet einen Port und wartet an diesem auf das Eintreffen einer Verbindungsanforderung. Port-Scans nutzen dies aus, um laufende Dienste zu ermitteln. Sie initiieren auf jedem möglichen Port einen Verbindungsaufbau und werten die Antwortpakete des Angriffsziels aus.

A.1.1 Connect-Scan

Beim Connect-Scan handelt es sich um die einfachste Form eines Port-Scans. Hierbei baut der Client mittels der BS-Systemfunktion `connect(...)` eine Verbindung zum Angriffsziel auf. War der Verbindungsaufbau erfolgreich, kann davon ausgegangen werden, dass ein Dienst an diesem Port wartet.

Der Vorteil des Connect-Scans ist die Nutzung von frei zugänglichen BS-Systemfunktionen. Dadurch ist es für jeden Nutzer ohne besondere Rechte möglich diesen Scan auszuführen. Alle anderen Scans erwarten root-Privilegien und können somit nicht von jedermann ausgeführt werden.

Das Problem dieses Scans liegt in seiner Einfachheit. Auf vielen Systemen wird der erfolgreiche Aufbau einer Verbindung protokolliert, so dass ein Connect-Scan bereits in der Log-Datei des syslog-Dämon zu finden ist.

A.1.2 TCP SYN-Scan

Um einer Protokollierung durch den syslog-Dämon zu entgehen, werden sogenannte SYN-Scans eingesetzt. Sie werden auch als Half-Scan bezeichnet, da sie den Verbindungsaufbau nur zur Hälfte ausführen. Hierzu wird nur eine SYN-Paket[20] an den Server geschickt. Je nachdem ob an diesem Port ein Dienst horcht, fällt die Antwort aus. Ist der Port offen, wird ein SYN-ACK-Paket entsprechend dem Dreiwege-Handshake des TCP an den Client zurückgeschickt. Ist der Port geschlossen, erhält der

[20] Hierbei handelt es sich um ein TCP-Paket, bei dem nur das SYN-Flag des TCP-Headers gesetzt ist. Im Folgenden wird diese Art der Paketbezeichnung auch im Bezug auf andere TCP-Flags erfolgen und ist an diesen Stellen synonym zu verstehen.

Client in der Regel ein RST-Paket. Mit diesen Informationen lässt sich eindeutig sagen, auf welchem Port ein Dienst läuft.

Der Vorteil des Scans liegt in seiner guten Tarnung. Mit regulären Mitteln eines BS ist er nicht zu erkennen. Selbst beim Einsatz eines IDS bleibt das Problem, einen solchen Port-Scan von einem fehlgeschlagenen Verbindungsaufbau zu unterscheiden. Werden beim Scan die einzelnen Ports in sehr kurzen Abständen getestet, kann leicht die Anzahl der eingehenden Verbindungsanforderungen von einer IP-Adresse überprüft werden. Beim Überschreiten eines Schwellwertes kann man davon ausgehen, dass der Rechner gescannt wurde. Allerdings existieren Abarten des Scans, bei denen die Ports über einen sehr langen Zeitraum mit wechselnden Quelladressen getestet werden. Diese Scans können eigentlich nicht erkannt werden.

Um diesen Scan durchführen zu können, werden root-Privelegien benötigt, die in einem sicheren Netzwerk nur die wenigsten Nutzer haben sollten. Anders ist dies bei der Verwendung von MS-Dos, dieses BS kennt keine Nutzerprivilegien, so dass jeder Nutzer einen Scan ausführen kann.

A.1.3 TCP FIN-Scan

Beim FIN-Scan wird lediglich das FIN-Flag des Paketes gesetzt. Offene Ports ignorieren dieses Paket, wohingegen geschlossene Ports ein RST-Paket an den Absender zurückschicken. Dieses Verhalten ist für das korrekte Funktionieren des TCP notwendig. Andere Systeme, bspw. Microsoft Windows, senden immer ein RST-Paket an den Absender zurück, unabhängig davon, in welchem Zustand sich der Port befindet.

Für die Erkennung von offenen Ports ist der SYN-Scan eigentlich besser geeignet. Jedoch existieren bereits viele sogenannte Netzwerk-IDS, die diesen erkennen und protokollieren. Bei solchen Systemen kann manchmal mittels eines FIN-Scans wiederum der Rechner unbemerkt untersucht werden. Anderseits besteht durch das unterschiedliche Verhalten der verschiedenen BS-Implementierungen die Möglichkeit, diese zu unterscheiden. So kann mittels eines FIN-Scans leicht ein UNIX- von einem Microsoft-System unterschieden werden.

Der Nachteil des Scans liegt in der Ungenauigkeit. Es können leider nicht alle Systeme untersucht werden. Zudem blockieren die meisten Paketfilter FIN-Pakete, wenn sie keiner Verbindung zugeordnet werden können und schicken an den Absender ein RST-Paket zurück. Demzufolge würde ein solcher Paketfilter wie ein Microsoft-System erscheinen.

A.2 Denial of Service Attacken

Grundsätzlich ist ein Denial of Service (DoS-) Angriff jede von einem Menschen oder einem Rechner initiierte Aktion, die Hardware oder Software eines anderen Rechners beeinträchtigt, das System unerreichbar macht und daraus folgend Dienste für berechtigte Nutzer verweigert. [10]

Viele der DoS-Angriffe nutzen Fehler, Grenzen oder Unstimmigkeit der Stack-Implementierung aus. Solange diese Fehler nicht erkannt und behoben werden, besteht für die betroffenen Systeme keine Möglichkeit einen von außen provozierten Systemausfall abzuwehren. Zu Zeiten des eCommerce und der vollständigen Automatisierung spielt die Verfügbarkeit von Systemen eine wesentlich Rolle. Der Ausfall eines Systems bedeutet nicht selten den Verlust von barem Geld.

Die hier vorgestellten DoS-Angriffe sind alle seit langem bekannt. Jedoch wird immer mal wieder versucht, ein Rechner mit einem dieser Angriffe zu schädigen. Das Erstaunliche ist, dass die Angreifer oftmals Erfolg haben, da die Systemadministratoren die bekannten Bug-Fixes oder Service-Packs noch nicht installiert haben.

A.2.1 Syn-Flooding

Der TCP-Verbindungsaufbau basiert im Wesentlichen auf einer festen Abfolge von empfangenen und gesendeten Paketen mit speziell gesetzten TCP-Flags. Wird diese Abfolge unterbrochen oder verändert, lässt sich oftmals ein unerwünschtes Verhalten auf dem Zielsystem provozieren.

Für jede Verbindung muss der Server eine gewisse Menge an Ressourcen zur Verfügung stellen. Jede Art der Ressourcenbindung stellt ein potentielles Ziel einer DoS-Attacke dar, so auch beim TCP-Verbindungsaufbau. Sendet ein Client lediglich ein SYN-Paket und unterrichtet den Server nicht davon, dass er die Verbindung nicht weiter aufbauen möchte, wird der Server die für die Verbindung notwendigen Ressourcen solange bereitstellen, bis ein gewisser Zeitraum verstrichen ist. Sendet der Client nun eine Vielzahl solcher SYN-Pakete, kann er alle Ressourcen des Servers belegen, so dass kein weiterer Client eine Verbindung aufbauen kann.

Die beschriebene Attacke basiert auf keinem Design- oder Implementierungsfehler des TCP, sondern nutzt vielmehr ein Designkriterium des Protokolls aus. Bei jedem Verbindungsaufbau ist es möglich, dass Pakete des Clients verloren gehen. Um auch in solchen Fällen eine Verbindung aufbauen zu können, muss dem Client die Möglichkeit gegeben werden, den Verlust des Paketes zu bemerken und ein weiteres Paket zu senden. Für diesen Zeitraum muss der Server die bereits angeforderten Ressourcen bereithalten.

A.2.2 Land

Die Land-Attacke erlaubt das komplette Blockieren eines Rechners mit nur einem TCP-Paket. Für den Angriff muss zunächst ein Dienst ermittelt werden, der Pakete von der eigenen Quelladresse akzeptiert. Ist ein solche Dienst gefunden, wird ein SYN-Paket mit gleicher Quell- und Zieladresse sowie gleichem Quell- und Zielport an diesen Dienst geschickt.

Viele Implementierungen geraten daraufhin in eine Endlosschleife und das System muss neu gestartet werden. Davon sind sowohl viele LINUX-Versionen als auch andere UNIX- und Microsoft-Systeme betroffen.

A.2.3 Ping of Death

Bei einigen älteren Versionen der Betriebssysteme LINUX und Windows ist eine fehlerhafte Implementierung der Funktionen zur Behandlung von IP-Fragmenten vorhanden. Sendet man ein Paket, das größer als die zulässigen 64kByte ist, stürzen diese Rechner ab.

A.2.4 Teardrop, Teardrop2

Ähnlich wie der Ping of Death nutzt `teardrop` bzw. `teardrop2` eine fehlerhafte Implementierung der IP-Fragmentbehandlung des Netzwerk-Stacks aus. In älteren Implementierungen von LINUX wurde zwar die Größe eines Paketes kontrolliert, jedoch nur dahingehend, ob sie zu groß sind. Sendete man ein Paket, das eine negative Fragmentlänge enthielt, verursachte dies in der Funktion `ip_glue(...)`, welche für das Zusammensetzen der einzelnen Fragmente verantwortlich ist, ein undefiniertes Verhalten. Was wiederum dazu führte, dass die Funktion `ip_frag_create(...)`, die anschließend aufgerufen wurde, große Datenmengen in das Paket kopierte, wodurch das System abstürzte.

Obwohl `teardrop` ursprünglich zum Angreifen von LINUX-Systemen implementiert wurde, verursachte dieser Angriff auch ähnliche Auswirkungen auf anderen Systemen. Mit kleinen Abänderungen in der Implementierung des Angriffes konnte auch auf Windows-Systemen der wohlbekannte „Blue-Screen“ erzeugt werden.

A.3 Spoofing

Als Spoofing wurde ursprünglich das Fälschen von Quelladressen bezeichnet. Hierbei wurde die eigene IP-Adresse durch eine andere erwünschte Adresse ausgetaucht, um Adress-basierte Authentifizierungsverfahren umgehen zu können. In letzter Zeit wurde dies immer mehr erweitert, so dass der Begriff Spoofing für alle Methoden zur Untergrabung von Authentifizierungs- und Identifizierungsverfahren verwendet wird. Dazu gehört unter anderem das ARP-, IP- und das DNS-Spoofing.

Da Spoofing-Angriffe mit einem Monitor schlecht zu erkennen sind, beschränkt sich die Arbeit auf die Beschreibung des Sequence Number Sampling und des rshd-Spoofing mittels des Programms `mendax`.

A.3.1 Sequence Number Sampling

Für die Entführung von TCP-Verbindungen ist es notwendig, Kenntnisse über die verwendete Sequenznummer zu haben, denn alle Pakete mit Sequenznummer außerhalb eines bestimmten Zeitrahmens werden vom Empfänger verworfen. Somit wird jede Datenübertragung zum Ziel fehlschlagen.

Hat ein Angreifer erst einmal die richtigen Sequenznummern erraten, ist es leicht möglich Adress-basierte Authentifizierungsverfahren zu umgehen. Besonders anfällig hierfür sind die sogenannten r-Utilities.

A.3.2 Mendax

Bei dem Programm `mendax` handelt es sich um ein einfach zu bedienendes LINUX-Tools zur Bestimmung von TCP-Sequenznummern und zum rshd-Spoofing. Der rsh-Dämon ist Bestandteil der r-Utilities und verwendet die Quelladresse eines Paketes für dessen Authentifikation. Das Programm `mendax` dient als Beispiel für einen Großteil der im Internet verfügbaren IP-Spoofing-Programme.

Für die Ausführung eines Angriffs werden drei Rechner benötigt: ein Rechner (A), auf dem der rsh-Dämon läuft, ein Rechner (B), der auf Rechner A zugreifen darf, und der Rechner (C) des eigentlichen Angreifers. Der Rechner A muss so konfiguriert sein, dass alle Zugriffe von Rechner B nur anhand der IP-Adresse authentifiziert werden.

Das Programm `mendax` wird nun versuchen, die von Rechner A verwendeten Sequenznummern zu erraten. War dies erfolgreich, wird der Rechner B mit Hilfe des SYN-Floodings vorrübergehend Außerbetrieb gesetzt und eine Verbindung zum Rechner A aufgebaut. Beim Verbindungsaufbau wird die IP-Adresse des Rechners B als Quelladresse in die Pakete eingetragen. Rechner A wird diese akzeptieren und die Verbindung aufbauen. Anschließend führt Rechner C den Befehl:

```
mv .rhosts .r; echo + + > .rhosts
```

aus. Danach kann jeder Rechner ohne die Eingabe eines Passwortes auf den Rechner A zugreifen.

Anhang B

Messbedingungen

Für die Untersuchung der Performanceverluste, die durch die HoNA-Implementierung und den Vergleichsimplementierungen entstanden sind, wurde in einem lokalen Netzwerk ein Testnetz eingerichtet, welches frei von äußeren Einflüssen war.

In diesem Abschnitt werden die verwendeten Testsysteme, die Netzstruktur und die verwendete Software beschrieben.

B.1 Testsysteme

Insgesamt wurden drei Rechner verwendet. Auf zwei dieser Systeme wurde jeweils die zu testende Software installiert. Das dritte System diente als Referenzsystem.

B.1.1 Testsystem 1 – Dual-Pentium

Hardware	System
Mainboard	Gigabyte 686 BXDS
CPU	2x Intel Pentium III (Katmai) 600 MHz
Arbeitsspeicher	256 MB SDRam (100 MHz)
Disk-Controller	Dual SCSI Adaptec 3940UW Onboard
Festplatten	Western Digital Enterprise 4360 UltraSCSI
	Fujitsu MAB 3091S UltraWideSCSI
	Quantum Atlas XP31080W WideSCSI
Netzwerk-Interface	Realtek RTL8139(B) PCI Fast Ethernet Adapter
	Realtek RTL8029 PCI Ethernet Adapter
Software	Linux RedHat 6.2
	Kernel 2.4.1

Tabelle 3 : Testsystem 1 – Dual Pentium

Die Installation von LINUX ist auf die drei Systemplatten wie folgt verteilt:

Mount-Punkt	Systemplatte
`/`	WDE 4360
`/usr`	Fujitsu MAB 3091 S
`/var`	Quantum XP31080W
`/tmp`	Quantum XP31080W

Tabelle 4 : Mount-Point – Testsystem 1

Die Audit-Daten wurden im Verzeichnis `/tmp` und die Daten vom IDS `snort` wurden im Verzeichnis `/var/log/snort` abgelegt. Die Datei `foo` zum Testen der Übertragungsleistung lag ebenfalls im Verzeichnis `/tmp`.

B.1.2 Testsystem 2 – Notebook

Hardware	System
Mainboard	Toshiba Portege 7020CT
CPU	Intel Mobile Pentium II (Dixon) 366 MHz
Arbeitsspeicher	192 MB SDRam (66 MHz)
Disk-Controller	Onboard IDE-Controller
Festplatten	Toshiba Mk 6411 MAT
Netzwerk-Interface	Intel 8255x-basierter PCI Fast-Ethernet Adapter
	Xircom CreditCard PCMCIA Fast Ethernet Adapter
Software	Linux SuSE 6.3
	Kernel 2.4.1

Tabelle 5 : Testsystem 2 – Notebook

Das System verfügt nur über eine Systemfestplatte, so dass der Audit-Trail und die Testdatei auch auf dieser abgelegt wurden.

B.1.3 Testsystem 3 – Ultra Sparc 1

Hardware	System
Mainboard	Sun UltraSparc 1/170E
CPU	Texas Instruments UltraSparc I 167 MHz
Arbeitsspeicher	384 MB
Disk-Controller	ESP UltraWide SCSI-Controller
Festplatten	Fujitsu MAB 3091S UltraWideSCSI
Netzwerk-Interface	10/100 MBit Fast Ethernet onboard
Software	Linux RedHat 6.2 SparcEdition
	Kernel 2.2.14-sparc64

Tabelle 6 : Testsystem 3 – UltraSparc1

Wie das Notebook verfügt das Sparc-System nur über eine Systemfestplatte, so dass die Testdatei auch auf dieser abgelegt werden musste.

B.2 Testnetz

Da in dem lokalen Netzwerk noch andere Rechner vorhanden waren, mussten die Tests in verschiedenen Netzwerkkonfigurationen durchgeführt werden. Im Folgenden werden die einzelnen Konfigurationen kurz beschrieben.

B.2.1 Performancetests

Das Testnetz zur Messung der Übertragungsleistung bestand im Wesentlichen aus den drei Rechnern aus Anhang B.1. Sie waren für die Performance-Tests über einen 100 MBit Switch miteinander verbunden. Jeder Rechner wurde mit einem seiner 100 MBit Schnittstellen an den Switch angeschlossen.

Neben diesen drei Rechnern wurde noch ein File-Server und ein Firewall an diesem Switch betrieben. Der Firewall war jedoch während der Test abgeschaltet und der File-Server erzeugte keine nennenswerte Last (unter 6 kBytes pro Sekunden). Dies wurde mittels des Programms `ethereal` zuvor überprüft.

B.2.2 DoS-Angriffe

Die DoS-Angriffe wurden in einem separatem Netz durchgeführt. Als Angreifer diente immer das Testsystem 2 und als angegriffenes System der Rechner 1. Auf diesem wurden auch die zu testenden Applikationen (LOSA, `snort` oder `ethereal`) betrieben.

Neben diesen Rechnern war ein LINUX-Firewall zur Anbindung des Netzes an das Internet im Testnetz vorhanden.

B.2.3 Spoofing

Für die Spoofing-Angriffe, insbesondere mittels `mendax`, wurde ein dritter Rechner benötigt. Hierzu wurde ein Vobis LeBook Notebook verwendet. Da die Spoofing-Attacken unabhängig von der Hardware der Systeme sind, wurde auf eine Beschreibung der Hardware des Notebooks im Anhang B.1 verzichtet.

Das Netz selbst war unabhängig und bestand nur aus dem Notebook und den Systemen 1 und 2. Eine Anbindung zum Internet war nicht vorhanden.

B.3 Software-Releases

Dieser Abschnitt enthält eine kurze Beschreibung der verwendeten Software. Die zugrundeliegenden Distributionen und Kernel Versionen wurden bereits im Anhang B.1 erläutert.

B.3.1 ethereal

Das Programm `ethereal` wurde der RedHat und der SuSE Distribution entnommen. Bei beiden Programmpaketen handelte es sich um die Original-RPM-Dateien der Version 0.8.1.

Die `libpcap` wurde ebenfalls aus den Distributionen genommen und hatte die Versionsnummer 0.4.

B.3.2 snort

Das Programm `snort` wurde ebenfalls mittels eines RPM installiert. Die Dateien wurden von der Homepage heruntergeladen und unverändert verwendet. Da für SuSE kein RPM verfügbar war, wurden die Dateien des RedHat-RPM mittels `tar` auf das SuSE-System übertragen. Die verwendete Versionsnummer war 1.7.

Wie für `ethereal` wurde auch für `snort` die `libpcap` der Version 0.4 verwendet.

Anhang C

Installation und Konfigurierung

In diesem Abschnitt erfolgt die Erläuterung der manuellen Installation. Die Installation von HoNA unterteilt sich in mehrere Schritte. Das System sollte erst dann wieder neu gestartet werden, wenn alle Schritte erfolgreich abgearbeitet wurden.

Patchen des Kernels

Das Patchen des Kernels erfolgt mittels des im Verzeichnis `patches/linux-[Version]-patch/` vorhandenen Skriptes. Das Skript erstellt im Kernel-Verzeichnis ein Verzeichnis namens `audit`, in welches alle Dateien des Audit-Moduls kopiert werden. Zudem werden die notwendigen Header im Verzeichnis `include/linux` abgelegt. Anschließend wird der Kernel-Patch eingespielt.

Wurde in den Kernel bereits ein anderer Patch eingespielt oder entspricht die verwendete Kernel-Version nicht der Patch-Version, kann das Installieren des Patches bei einigen Dateien fehlschlagen. In diesem Fall müssen die betroffenen Dateien per Hand nacheditiert werden. Das Programm **`patch`** hinterlegt für jede Datei, die nicht aktualisiert werden konnte, zwei Dateien namens `[orig_name].orig` und `[orig_name].rej`. Die Datei `*.rej` enthält die nicht eingefügten Zeilen des Patches. Diese Zeilen müssen dann nachträglich per Hand übertragen werden. Hierzu empfiehlt es sich, einen unveränderten Kernel bzw. einen Kernel der richtigen Version zu patchen und diesen als Referenz zu verwenden.

Damit das Audit-Modul beim Übersetzen des Kernels eingebunden wird, muss dieses in der Kernel-Konfiguration aktiviert werden. Nach der Installation des Patches ist dies nicht der Fall. Standardmäßig ist die Option `‘Audit support‘` im Menü `‘Processor type and features‘` auf `no` gesetzt.

Bevor die nächsten Schritte der Installation ausgeführt werden können, muss der Kernel mittels:

```
make dep clean bzImage modules modules_install
```

konfiguriert und übersetzt werden.

Nachdem die Datei `/etc/lilo.conf` angepasst wurde, kann der Kernel neu installiert werden. Eine genaue Beschreibung hierzu findet man auch in [13].

Installieren der Audit-Bibliothek

Die Installation der Audit-Bibliothek erfolgt mittels `make` und `make install` innerhalb des Verzeichnisses `libaudit`. Hierbei ist unbedingt darauf zu achten, dass die Version des verwendeten Kernel-Patches mit der Version der Audit-Bibliothek übereinstimmt.

In der Datei `Makefile.global` können die Pfade und Besitzer der Dateien der Bibliothek festgelegt werden. Normalerweise wird die Bibliothek in das Verzeichnis `/usr/lib` installiert, sollte dies nicht gewünscht sein, kann im Makefile ein anderes Verzeichnis angegeben werden. Da alle Applikationen von LOSA auf die Audit-Bibliothek zugreifen, ist darauf zu achten, dass das neue

Verzeichnis in den Environment-Variablen `LD_LIBRARY_PATH` und `LD_RUN_PATH` enthalten und in den Makefiles der LOSA-Applikationen zusätzlich angegeben ist.

Schlägt das Übersetzen der Bibliothek fehl, so liegt dies meist daran, dass der Kernel noch nicht übersetzt bzw. konfiguriert wurde und damit der Link für das `include`-Verzeichnis noch nicht gesetzt wurde. Ebenso sollte überprüft werden, ob der Link `linux` im Verzeichnis `/usr/include` auf das korrekte Header-Verzeichnis des Kernels verweist.

Installieren der audit-Applikationen

Die einzelnen Applikationen befinden sich in den Unterverzeichnissen des Verzeichnisses `utils`. Für den Betrieb von LOSA sind die Applikationen:

- **`aca`**: Audit Control Application – Steuerungsprogramm und
- **`auditd`**: Audit-Dämon

zwingend erforderlich. Das Programm:

- **`atv`**: Audit Trail-Viewer

ist lediglich für die Betrachtung der Audit-Daten notwendig. Es sollte aber, so kein anderes Programm vorhanden ist, installiert werden.

Die Installation kann entweder für jedes Programm separat aus dem jeweiligen Verzeichnis oder direkt aus dem Verzeichnis `utils` heraus erfolgen. Wie bei der Audit-Bibliothek werden die Pfade und die Nutzerrechte in der Datei `Makefile.global` festgelegt. Dies gilt sowohl für die separate als auch für die gemeinsame Installation der Programme.

Installation der man-Pages

Für die Installation der man-Pages befindet sich im Verzeichnis `man` das Skript `install_man.sh`. Hierbei handelt es sich um ein Shell-Skript, welches die Installation der einzelnen man-Pages in die jeweiligen Unterverzeichnisse übernimmt. Die im Skript vorhandenen Variablen

- `USR`: Besitzer der man-Pages (default: `root`),
- `GRP`: Gruppe des Besitzers (default: `man`) und
- `MANDIR`: Installationsverzeichnis (default: `/usr/man`)

können, so dies notwendig ist, entsprechend der Richtlinien des eigenen Systems angepasst werden.

Installation der Konfigurationsdateien

Im Verzeichnis `config` befinden sich die Konfigurationsdateien des Audit-Moduls. Standardmäßig werden die Dateien in das Verzeichnis `/etc/security` kopiert. Sollte dies nicht den allgemeinen Richtlinien des Systems entsprechen und ein anderes Zielverzeichnis festgelegt werden, ist darauf zu achten, dass alle Applikationen ihre Konfigurationsdateien in diesem Verzeichnis suchen. Um die Programme des Paketes nutzen zu können, muss das neue Verzeichnis jedem Programm als Parameter übergeben werden.

In der Datei `auditd.conf` wird unter anderem das Verzeichnis der Audit-Trails und das Verzeichnis für die `connect`-Datei festgelegt. Bevor das Audit-Modul aktiviert werden kann,

müssen diese Verzeichnisse erstellt werden. In der vorgegebenen Konfiguration sind dies die Verzeichnisse `/audit` und `/var/run`. Für den Einsatz auf einem produktiven System ist es ratsam, dass das Verzeichnis der Audit-Dateien auf eine separate Partition verweist. Dies verhindert, dass wichtige Partitionen vollgeschrieben werden.

Installation der Skript-Dateien

Für den Betrieb des Audit-Moduls sind die vier Skripte

- `audit`: Audit-Init-Skript
- `audit_shutdown`: Shutdown-Skript
- `audit_warn`: Warn-Skript
- `awinit`: Applikation-Wrapper Skript

notwendig.

Das Skript `audit` sollte in das Verzeichnis der init-Skripte des Systems kopiert werden. Bei RedHat und bei fast allen anderen LINUX-Distributionen ist dies das Verzeichnis `/etc/rc.d/init.d`. Anschließend muss noch ein Link im gewünschten Runlevel-Verzeichnis erzeugt werden. Das aktuelle Runlevel sollte in der Datei `/etc/inittab` zufinden sein. Werden mehrere Runlevels verwendet, sollte in den jeweiligen Verzeichnissen ebenfalls ein Link erzeugt werden. Das Erzeugen des Links erfolgt mittels

```
ln –s ../init.d/audit S01audit
```

aus dem Zielverzeichnis heraus. Es ist ratsam dem Audit-Skript den Präfix S01 zugeben, dadurch wird der Audit-Dämon vor allen anderen Programmen gestartet und kann diese damit von Anfang an überwachen. In jedem Fall muss das init-Skript vor den Skripten, die mittels des Steuerungsprogramms verändert wurden, gestartet werden. Anderenfalls können wichtige Einstellungen und damit Audit-Daten verloren gehen.

Die Skripte `audit_shutdown` und `audit_warn` sollten entsprechend der Konfiguration in die jeweiligen Verzeichnisse kopiert werden. Bei der vorgegebenen Konfiguration ist dies das Verzeichnis `/etc/security`.

Das Wrapper-Init-Skript wird standardmäßig in das Verzeichnis `/usr/sbin` installiert. Es wird entweder vom Administrator oder vom Audit-init-Skript zum Erzeugen der Wrapper-Skripte benötigt. Sollte in der Datei `audit.conf` die Variable `app_wr` auf `script` gesetzt sein, ist darauf zu achten, dass das init-Skript das Wrapper-Skript findet.

Wie bereits mehrfach erwähnt wurde, kann das Audit-Modul ohne root-Privilegien betrieben werden. In diesem Fall müssen die Skripte und alle von den Skripten aufgerufenen Programme vom Audit-Administrator ausführbar sein.

Erstellen der ACL-Datenbank

Das Erstellen der ACL-Datenbank erfolgt mittels des Steuerungsprogramms, welches zu diesem Zeitpunkt bereits installiert sein sollte. Bevor man das Programm startet, sollte die Konfigurationsdatei `audit_acl.conf` angepasst werden. Anschließend kann durch folgenden Aufruf:

```
aca –-create-acl /etc/security/audit_acl.conf /etc/audit_acl.db
```

eine neue ACL-Datenbank erstellt werden. Sollte bereits eine Datenbank vorhanden sein, kann auch folgendes Kommando zum Aktualisieren der Datei verwendet werden:

```
aca –-update-acl /etc/security/audit_acl.conf
```

Es ist in jedem Fall empfehlenswert, wenn HoNA neu installiert wird, eine vorhandene Datenbank zu aktualisieren.

Patchen der Systemapplikationen

Für den korrekten Betrieb des Audit-Moduls ist es zwingend notwendig, dass beim Anmelden eines Nutzers das Audit-Environment des Nutzer in den Kernel übertragen wird. Hierzu sind zwei Patches vorhanden. Zum einen für das Programm **`login`**, welches beim Anmelden auf der Console und vom inet-Dämon verwendet wird und ein Patch für das Programm **`kdm`**, welches einen graphischen Login bereitstellt.

Beide Programme müssen aus den Source-Paketen erstellt werden. Es ist empfehlenswert, zunächst die Programme ohne die Audit-Patches zu übersetzen und zu testen. Beide Pakete müssen vor dem Übersetzen mittels des Skriptes `configure` konfiguriert werden und können anschließend, mittels **`make`**, übersetzt werden. Eine Installation mittels `make install` ist nicht erforderlich, da die Pakete eine Vielzahl von Programmen enthalten, die nicht notwendig sind. Es reicht, die beiden Dateien `kdm` und `login` in die entsprechenden Verzeichnisse zu kopieren.

Ist das Anmelden an das System mit den neuen Programmen möglich, können diese mittels

```
cd /usr/src/redhat/SOURCES

patch –p0 < [losa-package]/patches/[package]-losa.patch

cd [package]

make
```

gepatcht und neu übersetzt werden.

Außerdem existiert ein Patch für das Programm **`su`**. Beim Übersetzen des Programms kann wie bei den Programmen **`kdm`** und **`login`** vorgegangen werden. Bei der Installation von **`su`** ist darauf zu achten, dass beim Kopieren des Programms das `setuid`-Recht verloren geht. Dies ist jedoch zwingend erforderlich und kann mittels

```
chmod u+s `which su`
```

neu gesetzt werden.

Anhang D

Literaturverzeichnis

[1] Patrick R. Gallagher, Jr., A Guide to Understanding Audit in Trusted Systems, Fort George G. Meade 1987

[2] Richter, Birk; Sobirey, Michael; König, Hartmut: „*Host-orientiertes Netz-Audit, Ein neuer Ansatz zur Protokollierung von Netzaktivitäten*“, Tagungsband KIVS'97, Springer-Verlag, Braunschweig, Februar 1997.

[3] Martius, Kai: „*Sicherheitsmanagement in TCP/IP-Netzwerken*“, Vieweg-Verlag, Braunschweig/Wiesbaden 2000.

[4] Sobirey Michael: „*Datenschutzorientiertes Intrusion Detection*“, Vieweg Verlag, Braunschweig/Wiesbaden 2000.

[5] Kalinowski, Dariusz: „*Host-basierte Protokollierung sicherheitrelevanter Netzwerkaktivitäten unter Microsoft Windows NT/2000*“, Diplomarbeit, Brandenburgische Technische Universität Cottbus, Oktober 2000.

[6] Stecklina, Oliver: „*Betriebssystem-Audit für Linux*“, Praktikumsbericht, Brandenburgische Technische Universität Cottbus, Januar 2001.

[7] Tannenbaum, Andrew S.: „*Computernetzwerke*“, 3. Auflage, Prentice Hall, München 1997.

[8] Hildebrandt, Ralf: „*Kain und Abel; snort und nmap – zwei Seiten der selben Münze*“, Linux Magazin, 12, 2000, Seite 102.

[9] Dittler, Hans Peter: „*IPv6 - das neue Internet-Protokoll: Technik, Anwendung, Migration*“, dpunkt-Verlag, Heidelberg 1998.

[10] Anonymous: „*Linux Hacker's Guide, Sicherheit für Linux-Server und –Netze*“, Markt&Technik, München 2000.

[11] Bovet, Daniel P.; Cesati, Marco: „*Understanding der Linux-Kernel*“, O'Reilly, Sebastopol 2001.

[12] Ousterhout, John K.: „*Tcl und Tk, Entwicklung grafischer Benutzeroberflächen für das X Window System*“, Addison-Wesley, Bonn 1995.

[13] Beck, Michael; Böhme, Harald; Dziadzka, Mirko; Kunitz, Ulrich; Magnus, Robert; Schröter, Claus; Verworrner, Dirk*: „Linux-Kernelprogrammierung, Algorithmen und Strukturen der Version 2.2*“, 5.Auflage, Addison-Wesley, Bonn 1999.

[14] Schubert, Jan; Leitner, Achim: „*Firewalls: Technologie und Architektur - Einführung in die Brandvorsorge*“, Linux Magazin, 6, 2001, Seite 36.

[15] Schöning, Uwe: „*Theoretische Informatik – kurzgefasst*“, 3. Auflage, Spektrum Akademischer Verlag, Heidelberg 1997.

[16] Linux Kernel Homepage: http://www.kernel.org.

[17] FreeS/WAN-Homepage: http://www.freeswan.org

[18] Ethereal-Homepage: http://ethereal.zing.org.

[19] Snort-Homepage: www.snort.org.

[20] Netfilter-Homepage: http://netfilter.filewatcher.org.

[21] H.E.R.T Audit Dämon: http://plan9.hert.org/projects/linux/auditd/

[22] *Linux Basic Security Modul* Projekt: http://linux.bsm.sourceforge.net

[23] nmap Netzwerkscanner: http://www.insecure.org/nmap