Utvärdering av Autoclass C

Martin Karresand, Dan Nordqvist

FOI-R--1484--SE
ISSN-1650-1942

Ledningssystem
Teknisk rapport

December 2005

FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1350 anställda varav ungefär 950 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömningen av olika typer av hot, system för ledning och hantering av kriser, skydd mot och hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Ledningssystem
Box 1165
581 11 LINKÖPING

Tel: 013-37 80 00
Fax: 013-37 81 00

www.foi.se

Martin Karresand, Dan Nordqvist

Utvärdering av Autoclass C

FOI-R--1484--SE
ISSN-1650-1942

Ledningssystem
Teknisk rapport

December 2005

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets Forskningsinstitut – FOI Ledningssystem Box 1165 581 11 LINKÖPING	FOI-R--1484--SE	Teknisk rapport
	Månad år December 2005	**Projektnummer** E 790133
	Forskningsområde Ledning, informationsteknik och sensorer	
	Delområde Ledning med samband, telekom och IT-system	
	Delområde 2	

Författare	Projektledare
Martin Karresand, Dan Nordqvist	Gunnar Wenngren
	Godkänd av Johan Allgurén IC, Institutionen för Systemutveckling och IT-säkerhet
	Tekniskt och/eller vetenskapligt ansvarig Johan Allgurén IC, Institutionen för Systemutveckling och IT-säkerhet
	Uppdragsgivare/kundbeteckning FMV

Rapporttitel

Utvärdering av Autoclass C

Sammanfattning

Den här rapporten beskriver en utvärdering av algoritmen Autoclass C. Algoritmen är baserad på Bayes sats vilken bland annat framgångsrikt används för att detektera spam mail. För utvärderingen användes datamängder från DARPA 98 för träning och test, samt egna data insamlade i ett laborationsnät. Algoritmen visade sig vara långsam vid träning, det vill säga framtagning av klasser. Den var däremot snabb när klasserna sedan användes för klassificering av okända data. För att kunna jämföra resultaten från användningen av de olika datamängderna användes Snort version 2.0.5 som facit. Sedan räknades hur många paket som klassificerats rätt enligt Snort. När DARPA 98-datamängden användes blev resultatet godtagbart, men den sex år yngre egeninsamlade datamängden kunde däremot inte klassificeras tillräckligt bra.

Nyckelord

algoritm, Autoclass C, Bayes, IDS, informationssäkerhet, intrångsdetektering

Övriga bibliografiska uppgifter

ISSN	Antal sidor	Språk
ISSN-1650-1942	40	Svenska
Distribution Enligt missiv	**Pris** Enligt prislista	
	Sekretess Öppen	

Issuing organisation	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 LINKÖPING	FOI-R--1484--SE	Technical report
	Month year December 2005	**Project number** E 790133
	Research area code C^4ISTAR	
	Sub area code C^4I	
	Sub area code 2	

Author(s)	Project manager
Martin Karresand, Dan Nordqvist	Gunnar Wenngren
	Approved by Johan Allgurén Head, Systems Developement and IT Security
	Scientifically and technically responsible Johan Allgurén Head, Systems Developement and IT Security
	Sponsoring agency FMV

Report title

Evaluation of Autoclass C

Abstract

This report presents an evaluation of Autoclass C, which is based on Bayes' theorem. The theorem has, among other things, been used to successfully detect spam mail. For the evaluation some of the data sets from the DARPA 98 collection were used to train and test Autoclass C. Also some locally produced data sets were used for the test phase. These were collected in the Information Warfare laboratory at FOI, Linköping. The algorithm turned out to be rather slow when being trained, but compensated with being fast when performing the actual classification of new and unknown data. To be able to compare the results from the different data sets Snort version 2.0.5 was used as a key. The number of packets correctly classified by Autoclass C were recorded. When the DARPA 98 data sets were used both for training and testing the result was satisfying, but when the locally collected data sets from 2004 were used, the result was unsatisfactory.

Keywords

algorithm, Autoclass C, Bayes, IDS, information security, intrusion detection

Further bibliographic information

ISSN	Pages	Language
ISSN-1650-1942	40	Swedish
Distribution By sendlist	**Price** According to price list	
	Security classification Unclassified	

Innehåll

1 Inledning

Det går att dra många paralleller mellan konstvärlden och datorvärlden, en av dem rör det brottsliga området. Konststölder från museer har förekommit i alla tider och så länge museerna är öppna för besökare kommer det även i framtiden att stjälas tavlor från dem. För att minska risken för stölder installeras olika former av inbrottslarm. De ökar risken för upptäckt och fungerar därför avskräckande på potentiella tjuvar. Larmen kan också aktivera olika former av fysiska skydd för att tillfälligt försvåra åtkomst av konsten.

Ovanstående scenario går i stora delar att överföra till datorvärlden, museernas öppenhet motsvaras av samhällets ökande beroende av Internet. Satsningen på det nätverksbaserade försvaret (NBF) och 24-timmarsmyndigheter är två exempel på denna utveckling. Den har många fördelar, men som samtidigt ökar risken för intrång och informationsstöld. Det går att göra systemen säkrare, men något helt säkert system är i praktiken omöjligt att uppnå om det fortfarande ska gå att använda. Därför behövs larmanordningar även i datorvärlden, larm i form av intrångsdetekteringssystem.

Det som skiljer exemplet med museer från datorexemplet är att i datorvärlden kan en kunnig angripare förbli anonym, trots att dennes intrång har detekterats. Det gör att inte alla angripare avskräcks av ett intrångsdetekteringssystem.

Hur bra ett intrångsdetekteringssystem är avgörs i mångt och mycket av hur bra den algoritm är som utgör kärnan i systemet. Därför är prestandan och effektiviteten hos dylika algoritmer mycket intressanta att studera. Den här rapporten presenterar av den anledningen en utvärdering Autoclass C, som bygger på Bayes sats.

Den princip för intrångsdetektering som den här undersökningen bygger på är anomalidetektering, vilket innebär att algoritmen som används ska lära sig normalläget för det bevakade systemet. Den detekterar sedan avvikelser mot normalläget. Problemet med metoden är att den ofta genererar en stor mängd falsklarm. Den har däremot potential att detektera nya typer av angrepp, vilket de algoritmer som bygger på signaturigenkänning inte klarar eftersom ett angrepp måste vara känt för att det ska ha en signatur. Därför är det mycket intressant att studera anomalidetekterande algoritmer för att om möjligt hitta metoder att minska antalet falsklarm och ytterligare förbättra detektionsförmågan av nya, hittills okända angreppstyper.

Det spelar inte någon roll vilken typ av nätverk som intrångsdetekteringen sker i. En anomalidetekterande algoritm upptäcker avvikelser från normalläget i de data som den matas med. Således kan den detektera intrång i både fasta nätverk och mobila nätverk. Det kan till och med vara nätverk av ad hoc-typ. Det går även att använda algoritmen för att detektera intrång i enskilda dato-

rer, till exempel tillfälligt autonoma noder i ett trådlöst, mobilt ad hoc-nätverk. Då används den som grund i ett systembaserat intrångsdetekteringssystem.

1.1 Bakgrund

Som ett del i Security Management-projektet 2003 fick FOI i uppdrag att studera en algoritm för intrångsdetektering. Arbetet omfattade 400 timmar, vilka fördelades på två personer. Det har bedrivits som en förstudie för att förbereda för en bredare och djupare undersökning av fler algoritmer och i slutänden färdiga intrångsdetekteringssystem.

Den algoritm som valdes var Autoclass C. Orsakerna till att valet föll på Autoclass C var bland annat att den är väl beprövad och därmed relativt fri från programmeringsfel. Vidare är källkoden och tillhörande dokumentation är fri och kan enkelt laddas ner från Internet. Dessutom bygger algoritmen på en matematisk teori etablerad redan på 1700-talet och som visat lovande resultat vid användning inom andra områden.

1.2 Syfte

Syftet med rapporten är att med utgångspunkt i verkliga loggdata studera Autoclass C och då främst dess prestanda vad gäller korrekt detektering av intrång. Detta görs för att i ett längre perspektiv få en bättre kunskap om olika algoritmtypers för och nackdelar, så att färdiga intrångsdetekteringssystems förmågor sedan kan värderas.

Studien ger också implicit en ökad förståelse och inte minst praktisk erfarenhet av intrångsdetekteringsalgoritmer i allmänhet samt specifikt av Autoclass C. Likaså erhålls ökad kunskap om testning och utvärdering av dylika algoritmer. Detta kommer att ge FOI och därmed också totalförsvaret en ökad kompetens inom intrångsdetekteringsområdet, något som kan komma väl tillpass i det fortsatta arbetet inom NBF.

1.3 Avgränsning

Studien är begränsad till en specifik implementation av Bayes sats i form av Autoclass C. Det är den färdiga produkten som studerats ur ett användarperspektiv.

De datamängder som använts för träning är hämtade från tcpdumpdelarna av data insamlade av DARPA 1998. Dessa har sedan behandlats med egenhändigt skrivna program för att formatera dem så att de passat Autoclass C.

De datamängder som används för testning och utvärdering kommer från ett nät utan några egentliga användare. De enda användarna är medlemmarna i IT-säkerhetsgruppen på vår institution, med andra ord är den trafik som förekommer till största delen angrepp eller spaningsåtgärder från Internet. Den icke angreppsrelaterade trafiken härrör från vanlig http- och ftptrafik i form av webbsurfning, filöverföring och andra laboratorierelaterade aktiviteter.

1.4 Förväntat resultat

Vi förväntar oss inte att Autoclass C ska kunna detektera lika mycket som till exempel Snort [5], eftersom den inte är specifikt anpassad för intrångsdetektering. Däremot förväntar vi oss att få en djupare insikt i de problem och möjligheter som konstruktörer av anomalidetekterande intrångsdetekteringssystem möter.

2 Teori

I den här delen presenteras den teoretiska bakgrunden till undersökningen. Först en generell del om Bayes sats och sedan en redogörelse för Autoclass C och dess egenskaper.

2.1 Bayes sats

Bayes sats formulerades redan under 1700-talet av en brittisk matematiker och tillika präst vid namn Thomas Bayes [11]. Satsen används för att beräkna sannolikheten att en hypotes är sann givet kända (observerade) data. Något förenklat innebär det att beräkna sannolikheten för en händelse i efterhand, ett exempel är hur stor risken är att ett positivt cancertest innebär att patienten också har cancer.

För att kunna beskriva Bayes sats behövs vissa notationer förklaras. Mängden av alla hypoteser rörande en viss händelse benämns $A = A_1, A_2, \ldots, A_N$, det finns alltid minst två hypoteser eftersom den hypotes som formulerats har ett komplement, en negation av ursprungshypotesen. Därför använder vi ett index för att beteckna vilken hypotes det gäller.

Mängden av hypoteser fyller tillsammans upp alla möjliga kombinationer som kan göras i varje specifikt fall. Det inses lätt när vi tänker på cancerfallet ovan, antingen har patienten cancer, eller så har den det inte.

De data som observeras avser ett objekt av en viss typ, till exempel cancertest. Det som observeras är mätbara egenskaper hos ett objekt, vi kan även kalla dem händelser. Vi låter $X = x_1, x_2, \ldots, x_n$ vara mängden av mätbara egenskaper, eller händelser.

Mätningen av egenskaperna behöver inte vara helt tillförlitliga, om vi återvänder till cancerfallet så säger provsvaren bara att det tagna provet med en viss sannolikhet innehåller cancerceller. Om vi låter G var händelsen att provet innehöll cancerceller och H vara händelsen att patienten också hade cancer, så skrivs sannolikheten att provsvaret också visade det som $P(H|G)$.

Skrivsättet $P(H|G)$ kallas betingad sannolikhet och betyder *sannolikheten för en händelse H förutsatt att G inträffat*. Ett specialfall är om H och G är oberoende. Då blir $P(H|G) = P(H)$. [4]

Om ovanstående benämningar och skrivsätt används får Bayes sats följande utseende:

$$P(A_i|X) = \frac{P(X|A_i)P(A_i)}{\sum_{j=1}^{N} P(X|A_j)P(A_j)} = \frac{P(X|A_i)P(A_i)}{P(X)} \tag{2.1}$$

2.1.1 Ett praktiskt exempel

För att lättare kunna förklara konceptet kan vi tänka oss att vi letar efter massförstörelsevapen i ett land någonstans. De underrättelsedata vi har tillgång till pekar på att var femte granat i det förråd vi undersöker innehåller kemiska stridsmedel. Detta är vår hypotes A_1 som vi skriver $P(A_1) = 0,20$. Komplementet, som behövs i ett senare skede, blir $P(A_2) = 1 - P(A_1) = 0,80$.

Till vår hjälp har vi en C-stridsmedelsdetektor som med 90 % tillförlitlighet kan upptäcka att det är C-stridsmedel i en granat. Tyvärr ger den också upphov till falsklarm i 5 % av fallen där det inte finns några C-stridsmedel. Detta blir således mängden av våra observerade data X (i det här fallet är det bara en mätbar faktor och $X = x_1$). Vi kan alltså beskriva sannolikheterna $P(X|A_1) = 0,90$ och $P(X|A_2) = 0,05$.

Vår detektor indikerar nu att det finns kemiska stridsmedel i en granat. Hur stor är då sannolikheten för vår hypotes om att så också är fallet? Svaret ges av Bayes sats och ser ut så här:

$$\begin{aligned} P(A_1|X) &= \frac{P(A_1)\cdot P(X|A_1)}{P(A_1)\cdot P(X|A_1)+P(A_2)\cdot P(X|A_2)} \\ &= \frac{0,20\cdot 0,90}{0,20\cdot 0,90+0,80\cdot 0,05} \\ &= 0,82 \end{aligned}$$

2.1.2 Användningsområden

Ekvation (2.1) är tillämpbar på valfritt antal antaganden och kan till exempel användas för klassificering av objekt eller händelser. Varje objekt har en eller flera egenskaper som kan mätas på något sätt. Dessa egenskaper ligger till grund för de klasser som formulerats på valfritt sätt och som sedan ett objekt ska fås att tillhöra.

Ett problem i sammanhanget är att veta hur sannolik varje klass är. Det är vanligt att alla klasser antas vara lika sannolika vilket kan vara ett felaktigt antagande och därmed göra klassificeringen oanvändbar. Många gånger går det dock att utifrån de träningsdata som använts för att hitta klasserna sluta sig till deras respektive sannolikhet. Förutsatt att de data som ska klassificeras är mycket lika de träningsdata som använts, är metoden tillämplig.

Det finns två grundprinciper för klassificeringen, den ena kallas för naiv bayesklassificering och den andra för bayesiska nät. Den största skillnaden mellan metoderna är att vid naiv bayesklassificering antas alla egenskaper hos en klass vara oberoende, i bayesiska nät är minst två egenskaper beroende av varandra. [10]

Antagandet att alla ingående egenskaper är oberoende underlättar beräkningarna jämfört med om de vore beroende. Själva klassificeringen innebär i korthet att hitta den klass, bland m stycken klasser, det är mest sannolikt att ett objekt tillhör. Det kan också uttryckas som att sannolikheten för att ett objekt X tillhör en viss klass C_i ska maximeras. Klasserna utgör hypoteser och alltså ersätter C_i helt enkelt A_i i Ekvation 2.1. I matematisk form får klassificeringen följande utseende:

$$P(C_i|X) > P(C_j|X) \qquad \text{för } 1 \leq j \leq m, j \neq i.$$

Här används sedan Bayes sats för att beräkna sannolikheten för varje klass och objekt. Uttrycket i nämnaren i Ekvation 2.1 innehåller den betingade sannolikheten för att ett objekt tillhör en viss klass summerat över alla klasser. Eftersom summan omfattar alla klasser blir nämnaren densamma oavsett objekt och det räcker med att beräkna täljaren. Det förkortade uttrycket kommer därmed att se ut så här:

$$P(C_i|X) = P(X|C_i)P(C_i) \qquad \text{för } i = 1, 2, \ldots, m \tag{2.2}$$

I kapitel 2.1 betecknas de n stycken egenskaper som vi mäter på objekt X med $x_1, x_2, \ldots, x_n$. När dessa egenskaper betraktas som oberoende kan den betingade sannolikheten beräknas på följande sätt:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \tag{2.3}$$

Om egenskaperna har diskreta mätvärden beräknas den betingade sannolikheten i högerledet i Ekvation (2.2) som $P(x_k|C_i) = \frac{s_{ik}}{s_i}$ där s_{ik} betecknar antal objekt i klass C_i som har värde x för egenskap k. Uttrycket s_i i nämnaren står för det totala antal objekt i klass C_i.

Har egenskaperna däremot kontinuerliga mätvärden, det vill säga antar vilket värde som helst inom ett intervall, brukar mätvärdena antas vara normalfördelade inom intervallet. En normalfördelning kan som bekant beskrivas med hjälp av dess medelvärde, μ, och dess varians, σ. Sannolikheten för att mätvärdet blir x_k för klass C_i, alltså $P(x_k|C_i)$, beräknas då så här:

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) = \frac{1}{\sqrt{2\pi\sigma_{C_i}}} e^{-\frac{(x_k-\mu_{C_i})^2}{2\sigma^2_{C_i}}} \tag{2.4}$$

När det gäller bayesiska nätverk så består de av en graf som beskriver de beroenden som finns i datamängden. Dessutom behövs en tabell som specificerar den betingade sannolikheten för respektive beroende.

Om alla beroenden är kända och alla variabler möjliga att mäta, beräknas de betingade sannolikhetsvärdena i tabellen på ungefär samma sätt som vid naiv bayesklassificering (se Ekvation (2.2)).

Ibland kan det dock vara så att det inte går att mäta en del egenskaper för vissa objekt. I det fallet används en optimeringsmetod som stegvis närmar sig en *lokalt* optimal lösning. På så sätt tränas algoritmen att känna igen, eller snarare hitta, de klasser som finns. En sak som är viktig att komma ihåg är att den uppsättning av klasser som algoritmen hittar inte nödvändigtvis är den optimala lösningen på problemet.

2.2 Autoclass C

Autoclass är en specifik implementation av Bayes sats för generell klassificering. Det finns flera versioner av programmet, den vi valt att studera är Autoclass C som är "public domain", gratis och skrivet i C. Bakom programmet återfinns Bayes-gruppen på Ames Research Center, NASA. [3]

Eftersom programmet är skrivet för att vara generellt och därmed kunna klassificera objekt från vitt skilda områden har det ett flertal inställningsmöjligheter. Programmets nyckelegenskaper enligt upphovsmännen är att det:

- bestämmer antal klasser automatiskt,
- kan hantera data innehållande både diskreta och reella värden blandat,
- kan hantera avsaknad av värden,
- exekveringstiden är ungefär linjär mot mängden data,
- olika fall har sannolikhetsbaserad klasstillhörighet,
- tillåter korrelation mellan egenskaper i en klass,
- genererar rapporter över de klasser som hittats,
- bestämmer ett testfalls klasstillhörighet utifrån klasser funna i träningsdatamängd.

Det har ingen specifik gräns för hur stora datamängder som kan hanteras, vare sig för träning eller klassificering i kända klasser. Eftersom exekveringstiden är ungefär linjär sker ingen dramatisk prestandaförlust vid något tröskelvärde. Exekveringstiden kan uppskattas som $N_{\text{data}} \cdot N_{\text{attribut}} \cdot N_{\text{klasser}} \cdot N_{\text{försök}} \cdot N_{\text{iterationer för konvergens}}$ där N betecknar antal. Den osäkra faktorn i beräkningen är $N_{\text{iterationer för konvergens}}$. Beroende på vilken typ av konvergensmodell som används kan antalet bli $10 < N_{\text{iterationer för konvergens}} < 200+$. Det finns en användarstyrd maximal gräns för antalet försök, grundinställningen är 200. [2]

För reella värden finns möjligheten att ange ett relativt mätfel som algoritmen sedan tar hänsyn till vid klassificeringen. Det går också att ange ett fast mätfel för varje egenskap.

De värdetyper med tillhörande attribut som finns att tillgå visas i Tabell 2.1. Real/location kan anta värden mellan $-\infty \leq x \leq +\infty$. Mätfelet anges

Tabell 2.1: Värdetyper med tillhörande attribut

Typ	Deltyp	Attribut
dummy	none/nil	—
discrete	nominal	range
real	location	error
real	scalar	zero point & rel error

specifikt för varje objekt. För real/scalar gäller att värdena har en undre gräns motsvarande *zero point*, det vill säga *zero point* $\leq x \leq \infty$. Dessutom anges mätfelet som ett relativt värde, vilket alltså är beroende av aktuellt mätvärde.

Fördelen med att dela upp de reella värdena i två typer är att för de skalära värdena undviks att algoritmen tar hänsyn till värden som aldrig kan erhållas. Ett praktiskt exempel är längd, vikt, volym och andra fysikaliska egenskaper som ett objekt kan ha. De kan aldrig bli mindre än noll och dylika värden kan alltså negligeras.

För att lättare kunna hantera de skalära värdena använder Autoclass C en logaritmfunktion, $\ln(x - \textit{zero point})$. Den ersätter x i normalfördelningsmodellen (Ekvation (2.4)) och ger en bättre modellering av värden på x som ligger nära *zero point*.

Autoclass använder fyra sannolikhetsmodeller för att kunna hantera olika typer av värden på egenskaperna. Det går att blanda flera modeller i samma körning av algoritmen.

De tre första modellerna motsvarar naiv bayesklassificering, det vill säga de förutsätter att alla egenskaper är oberoende. Det som skiljer modellerna åt är att en av dem, SINGLE_MULTINOMIAL, hanterar diskreta värden, även i form av text. De två andra, SINGLE_NORMAL_CN och SINGLE_NORMAL_CM, hanterar reella värden där den ena dessutom klarar av att en delmängd värden saknas.

Den fjärde modellen, MULTI_NORMAL_CN, hanterar egenskaper som är beroende av varandra. Den klarar dock inte av att hantera avsaknad av värden, utan datamängden måste vara komplett.

3 Metod

Den metod som användes för utvärdering av Autoclass C baserade sig på data tagna från ett aktivt, skarpt nät. Resultatet från körningen med dessa jämfördes sedan med en ofta använd datamängd, som dock är artificiell. För att kunna använda datamängderna krävdes en del arbete med formatering för att anpassa dem till algoritmen. Detta gjordes med hjälp av egna specialskrivna program.

I det här kapitlet beskrivs först de datamängder som användes. Sedan följer en presentation av datorkrigslaboratoriet (IDS-labbet) vid FOI i Linköping och den hårdvara som användes vid utvärderingen. Därefter görs en kortfattad introduktion till de program som skrevs för formateringen av datamängderna.

3.1 Datakälla

MIT (DARPA '98) [14] och KDD CUP '99 [1] har tidigare publicerat datamängder. Dessa data representerar artificiell eller simulerad nätverkstrafik och angrepp eftersom insamling skett i slutna nätverk utan tillgång till internet. I stället har trafikgeneratorer använts och angrepp har i vissa fall lagts till i efterhand i datamängden. [8, 12]

Bristen på verkliga, skarpa data att testa med, i kombination med det faktum att FOI har ett eget laborationsnätverk med internetförbindelse, gjorde att en egen insamling av nätverkstrafik blev aktuell. Tyvärr används inte laborationsnätverket i vardagslag och det gjorde att de data som samlades in inte heller kunde användas för att representera en skarp miljö med, eftersom datamängden till största delen innehöll angrepp och då i form av portskanningar och maskar.

För att Autoclass ska kunna skapa en användbar klassificering måste den tränas med data som innehåller exempel på så många olika varianter som möjligt av trafik som kan förekomma i nätet. Vi var därför tvungna att använda DARPA:s data för att träna algoritmen. DARPA har märkt upp sina datamängder med angreppstyper för att underlätta utvärdering av algoritmer och annan användning. Tyvärr var deras textuella version av träningsdata som extraherats från tcpdump fattig på attribut. Träningen av algoritmen kunde därför bli lidande eftersom de attribut som fanns inte riktigt passade att använda för intrångsdetektering. Till exempel var tidsstämplar, källadress och destination för kommunikationen inte särskilt intressanta ur ett framtidsperspektiv. Sådana attribut är ofta dynamiska i och med användningen av DHCP. De går även att förfalska. I vårt fall hade dessutom vår egen datamängd av förklarliga skäl inte samma adressrymd och tidsstämpling som DARPA:s.

KDD CUP 99:s träningsdata såg betydligt mer intressanta ut med sina 41

attribut. Vid en närmare granskning var de tyvärr alltför applikationsberoende. Det medförde att vi inte kunde formatera vår egen datamängd på samma sätt vilket gjorde att någon utvärdering inte var möjlig. KDD hade heller inte publicerat några tcpdumpar med rådata som annars kunde använts för omformatering till passande format.

Lösningen blev att använda DARPA 98:s tcpdumpar med rådata. Utifrån dessa kunde vi sedan extrahera träningsdata med rätt formatering, anpassad till en jämförelse med våra egna data.

3.1.1 Dataattribut

Varje rad i tränings och testdatafilerna innehåller en komplett nätverkssession från tcpdump. I TCP fallet rör det sig om paket från uppkoppling till och med nedkoppling av en session. För UDP, ICMP, IGRP och så vidare saknas dylika begrepp på nätverksnivån och de får alltså en ny unik session för varje enskild överföring, även om de egentligen hör ihop.

Följande attribut valdes som intressanta för algoritmen:

duration - tid i sekunder för sessionen

proto - protokoll för överföring, det vill säga TCP, UDP, ICMP osv

service - service som överföringen är riktad mot. (För närvarande implementerat som destinationsportens nummer mellan 1 och 65535. Detta kan eventuellt bytas till strängnamn på servicen istället. Ett namn skulle i så fall kunna omfatta fler portar, till exempel ftp som använder både port 20 och 21.)

srcport - klientens källport (1-65535)

destport - serverns destinationsport (1-65535)

initFlags - initiala flaggor för första paket i en session

curFlags - flaggor i sista paketet i sessionen

ttl - IP time-to-live

len - IP längd på sista paketet

fwdPayload - antal bitar data som gått från klient till server

rewPayload - antal bitar data som gått från server till klient

Följande attribut användes inte i klassificeringen, men skulle eventuellt kunna vara av värde för senare steg:

caseno - ordningsnummer på sessionen. Används som uppslagning och korsreferens av algoritmen.

startTime - tidsstämpel för sessionen

incoming - om förbindelsen är riktad från externt nätverk till internt (topologi)

srcIp - IP-nummer på klient

destIp - IP-nummer på server

id - TCP-paketidentifikation

label - angreppsmärkning alternativt normaltrafik, fotnot från Snort vid angrepp

3.2 Datorkrigslabbet

Här följer en kort beskrivning av den laborationsuppställning som användes för insamling av data i en skarp miljö.

3.2.1 Algoritmdator

Autoclass kräver att hela datamängden får plats i internminnet, det vill säga den kan inte arbeta sekvensiellt med delar av datamängden. För att kunna bearbeta stora mängder data och även samla in data i realtid i höghastighetsnät införskaffades en kraftfull dator med snabb processor, mycket internminne och snabba hårddiskar. Resultatet blev en dator (algoritmdatorn) med följande prestanda:

- Processor: Intel P4 3.0 Ghz
- Minne: 2 GB RAM DDR
- Hårddisk: SCSI Ultra 320 146 GB

Datorn installerades med operativsystemet RedHat 9 som hade version 2.4.20-8 av kärnan.

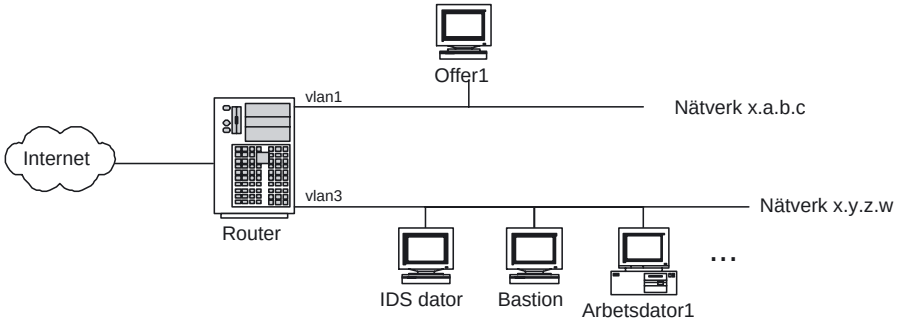
3.2.2 FOI laborationsnät

Vårt laborationsnätverk har två subnät, se Figur 3.1. All trafik till dessa går som synes genom routern. En lämplig punkt för datainsamling är sålunda just där. Ett problem är dock att det inte går att samla in data från båda nätverken samtidigt med en sådan topologi. Detta är möjligt att lösa genom att slå samman datamängderna från de båda subnäten, men en sådan lösning medför att det kan bli problem med tidsstämplar, löpnummer och andra attribut med sekvensiell ordning.

Routerns nätverksgränssnitt *vlan3* är kopplat till det subnät som har mest trafikaktivitet. På detta finns ett flertal datorer, bland annat algoritmdatorn, en bastiondator[1] med DNS och ett antal arbetsstationer som används sporadiskt för åtkomst av webbsidor på internet. Angreppsintensiteten mot detta nät är mycket hög, troligtvis på grund av att det exponeras utåt vid surfning på webben. Datamängden Test2a har samlats in i det här nätet.

Det andra nätet med gränssnitt *vlan1* innehöll vid tidpunkten för den här utvärderingen bara en dator, Offer1. I det här nätet samlades Test2b in. Nätet har en mycket lägre trafikintensitet och även färre angrepp än *vlan3*.

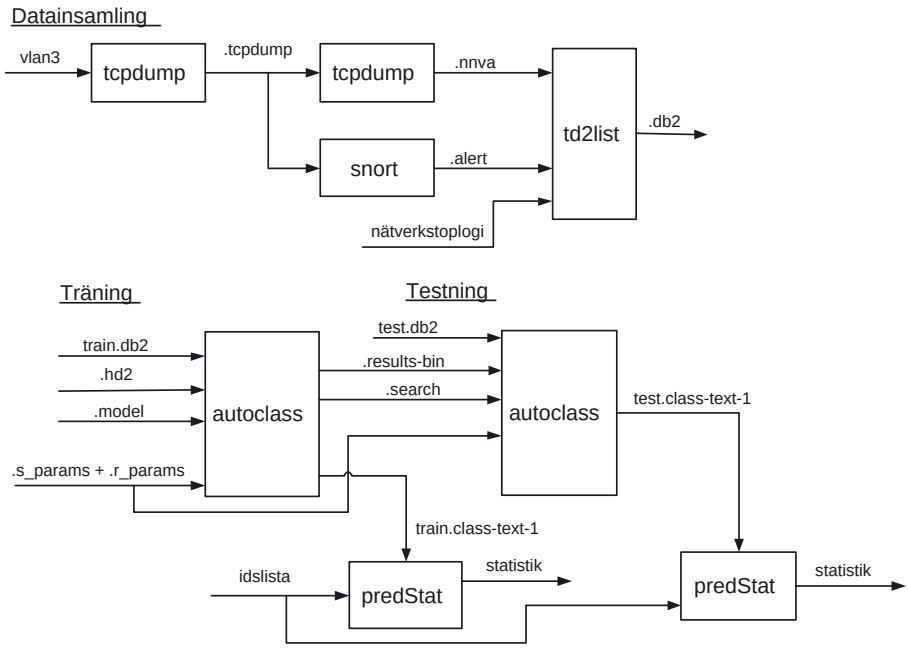
[1] En dator med förstärkt skydd som ofta ingår i eller finns utanför gränsskyddet av ett datorsystem.



Figur 3.1: FOI laborationsnät

3.3 Programvara

Här följer en förteckning över de olika datorprogram som använts för bland annat formatering av datamängderna. I Figur 3.2 visas en översikt av programmen.



Figur 3.2: Dataflöde och program

3.3.1 Tcpdump

Tcpdump användes i två steg. För parametrar se mansida för tcpdump [6]. Först loggade den all trafik till en binär datafil.

```
tcpdump -i vlan1 -s 65535 -tttt -n -F options -w data.tcpdump
```

Filen *options* användes för att exkludera trafik som ansågs störa mätningen. Under mätningarna överfördes mycket data via SSH (Secure Shell) med hjälp

av SCP (Secure Copy) på port 22 och att logga denna trafik skulle bara fyllt upp hårddisken med onödiga data. Filen hade följande utseende:

```
not (host x.y.z.w and port 22) and
not (net a.b.c.d and port 22)
```

Det andra steget med tcpdump var att läsa den binära datafilen och konvertera den till ascii-format med det här kommandot:

```
tcpdump -r data.tcpdump -nn -v > data.nnva
```

Alternativet hade varit att läsa den binära filen direkt, se Kapitel 3.3.4 nedan. En sådan lösning hade dock blivit för omfattande för att rymmas inom ramen för det här projektet.

3.3.2 Tcpslice

Vissa binära tcpdump-filer blev alldeles för stora för att kunna bearbetas inom rimlig tid. Därför användes tcpslice för att med hjälp av tidsstämplar beskära tcpdump-filerna till lämplig storlek. Se mansida för tcpslice [7] för mer information.

3.3.3 Snort

Ett facit (uppmärkning av angrepp respektive normal trafik) behövdes för att bedöma kvalitén på klassificeringen av testdatamängden. Det skulle användas som facit av oss (alltså ej av algoritmen) under träning och testning. För att kunna skapa ett sådant facit valde vi Snort eftersom det är ett mycket välkänt intrångsdetekteringssystem, det kan dessutom läsa tcpdump-filer och generera alarm utifrån dessa. DARPA-datamängden var klassificerad i angrepp och normal trafik, men processen som använts för att göra denna märkning är inte offentlig och kunde därför inte användas för att göra en ekvivalent uppmärkning av våra egna datamängder.

Facit framställdes med Snort version 2.0.5 från år 2004 och den medlevererade uppsättningen av regler användes i sin helhet utan förändring. Snort klassificerade inte alla händelser på exakt samma sätt som i gjorts DARPA-datamängden från 1998. Den största skillnaden var att en SYN-portskanning i datamängden från 1998 inte detekterades. Troligen var filtret för denna signatur avslagen i Snort:s grundinställning. En möjlig anledning kan vara att detta angrepp inte betraktas som särskilt intressant eller allvarligt nu för tiden och därför ignoreras.

Vid framställningen av facit startades Snort med följande kommando:

```
snort -A fast -c snort.conf -b -l . -r data.tcpdump > data.alert
```

I resultatet från Snort finns bland annat tidsstämpel för varje angrepp, samt angreppets namn. Allt som Snort larmade på togs med utan förbehåll, även till synes ofarliga telnet-förbindelser. 1998 användes fortfarande telnet som ett verktyg för vanlig, seriös trafik, inte bara för angrepp. Idag är telnet inte så intressant att använda som verktyg för seriös trafik eftersom trafik och lösenord är okrypterade och därmed syns i klartext. Telnet kan dock med fördel användas för intrång eftersom det är lätt att emulera (härma) andra protokoll

med det. På så sätt kan en angripare koppla upp sig mot en godtycklig port, till exempel smtp mail server och sedan utnyttja svagheter i den tjänsten med full kontroll av vad som sker. Det är därför mycket troligt att Snort därför numera är konfigurerat för att reagera på dylik trafik.

3.3.4 Generering av dataset - td2list

Den kanske största utmaningen för oss rent tekniskt sett var att omvandla en tcpdump-fil till ett lämpligt listformat (.db2). Vår första idé var att använda ett verktyg som kan läsa binära tcpdump-filer, såsom Ethereal. Ethereal har insticksmoduler för olika applikationer, så den skulle kunna användas för att ta fram attribut gällande dessa. Källkoden är dock mycket omfattande, vilket gjorde att vi tvingades söka oss andra vägar runt problemet. Lösningen blev att använda ett program som gjorde om tcpdumps alternativa utdataformat till vanlig ascii. Vi kunde tyvärr inte hitta något verktyg som gjorde precis detta, så vi fick utveckla ett eget istället.

Verktyget vi konstruerade fick namnet *td2list*, DARPA använde beteckningen list på sina asciidatafiler. Initialt blev namnet därför det längre namnet tcpdump2list, vilket sedan förkortades till td2list. Vi kunde dock lika gärna kallat det för det kanske mer korrekta td2db2.

Från början omvandlade bara programmet en .nnva-fil, men sedan lades alarm från Snort till samt en möjlighet att specificera den lokala nätverkstopologin. Tanken var att använda topologin för att hitta mönster i kommunikationsströmmarna, så som att hemmanätverket kommunicerar med en viss typ av paket lokalt, medan externa maskiner använder andra typer. Detta kunde ha varit intressant och ökat precisionen vid detektering, men i både våra data och DARPA 98:s var nästan all kommunikation initierad utifrån, vilket gjorde att vi valde att ignorera nätverkstopologin under träningsfasen.

Programmet td2list behandlar nätverkssessioner på olika sätt beroende på vilket protokoll det gäller. En TCP-session bearbetas som en enhet från början till slut. Den börjar oftast med SYN-handskakning och slutar med RESET eller en FIN-handskakning. Varje paket innehåller också ett sekvensnummer, vilket gör att de ingående paketen är lätta att känna igen (se RFC för TCP för mer om detta [9]). En UDP, ICMP eller annan överföring har inte några sekvensnummer på nätverksnivå. Det skulle därför vara nödvändigt att granska innehållet i datadelen i varje paket för att hålla reda på en session. Det gjorde att vi var tvungna att behandla varje överföring som en ny session.

Träningsdata - Train

Träningsdatamängden hämtades från DARPA 98, vecka 3, fredag och innehöll 60000 rader eller sessioner av de ursprungliga 77389. Anledningen till det var att det inte gick att träna algoritmen med 100 % av den ursprungliga datamängden för det uppstod minnesfel i malloc(). Därför användes bara 75 % av datamängden, vilket motsvarade 8 timmars nätverkstrafik. Med den mängden data tog det cirka 10 timmar att träna Autoclass.

Datamängden innehöll 777 angrepp fördelade på 11 angreppstyper varav "ICMP Ping Nmap" och "BAD-Traffic TCP port traffic" var de vanligaste (se [13] för en förklaring av dessa signaturer). Träningsdatamängden var arti-

ficiellt framställd och kunde därför eventuellt skilja sig från hur verkliga angreppsmönster egentligen såg ut vid tidpunkten för framställningen. Trots det använde vi den eftersom det var den enda datamängd vi hade tillgång till som innehöll både normal trafik och angrepp.

Testdata - Test1

Den första testdatamängden hämtades från DARPA 98, vecka 2, fredag. Den innehöll endast 15000 rader från DARPA 98:s datamängd eftersom vi ville hålla proportionen 4:1 mellan träningsdata och testdata, vilket är brukligt i IDS-världen. Med det antalet rader blev det ungefär 2 timmars nätverkstrafik att testa med. Datamängden innehöll fler angrepp än träningsdatamängden, 3079 angrepp fördelat på 20 olika typer. "SNMP public access UDP" följt av "POP3 PASS overflow attempt" var de två vanligaste angreppen.

Testdata - Test2a

Test2a är den ena av våra egna datamängder och samlades in 2004 från vlan3 nätverket. Datamängden innehöll cirka 15000 sessioner insamlade under en tidsperiod på 30 sekunder och detta i ett nätverk med få och passiva användare. Det mesta var följaktligen angrepp, 14629 st fördelat på 3 angreppstyper: "IMCP Ping CyberKit 2.2 Windows", "IMCP ping speedera" och "MS-SQL Worm propagation attempt" (se [13] för en närmare förklaring av dessa signaturer). Antalet angrepp och fördelningen av dem var förvånande och vi funderade på om de skulle gå att använda. Vi beslöt dock trots det att använda dem, eftersom de trots allt representerade ett verkligt fall av trafikmönster i nätverket.

En möjlig förklaring till den mycket höga angreppsfrekvensen kan vara att det just då fanns gott om datorer som var smittade med maskar och att dessa var i en skanningsfas vid insamlingstillfället.

Testdata - Test2b

Test2b är också data hämtade från FOI under 2004, men från vlan1 nätverket. Dessa data innehöll en betydligt lägre angreppsfrekvens, en timmes trafik innehöll 1254 angrepp. Angreppstyperna var dock desamma som för Test2a. Detta minskade naturligtvis värdet för den här datamängden ur evalueringssynpunkt, men eftersom den dock innehöll normal trafik som skilde sig från den i test2a använde vi den i alla fall.

3.3.5 Autoclass C

Algoritmen som valdes till utvärderingen finns tillgänglig som öppen källkod och är en implementation av Bayes algoritm i programspråket C. Den heter Autoclass C version 3.3.4. och är välkänd och beprövad inom en rad områden.

Användningen av Autoclass C kan kortfattat beskrivas i två steg, först träning på en lite större datamängd och därefter klassificering på en ny och för algoritmen okänd datamängd. För att resultatet av klassificeringen ska bli bra bör datamängd nummer två vara mindre än träningsmängden. Dess innehåll

bör inte heller avvika alltför mycket från posterna i träningsdatamängden. I vårt fall innebar steg två själva testet eller utvärderingen.

En förteckning över de viktigaste filerna i Autoclassimplementationen visas nedan. Den första delen av filnamnet har ibland utelämnats då ändelsen är det viktiga.

Träning (search) indata:

train.db2 - träningsdata med sessioner

.hd2 - beskrivning av attribut hos träningsdata

.model - beräkningsmodeller som ska appliceras på attributen

.s-params - sökberoende parametrar, hur många gånger algoritmen ska slumpmässigt skapa nya klasser innan den ger upp

.r-params - rapportparametrar, attribut som ska listas

Träning utdata:

.results-bin - binärt resultat av träning

.search - lista med iterationer över slumpmässigt skapade klasser

train.class-text-1l - korsreferens med funna klasser mot sessionerna i testdatamängden i korsreferensen

Testning (predict) indata:

test.db2 - testdata med sessioner

.results-bin - se ovan, Kapitel 3.3.5

.search - se ovan, Kapitel 3.3.5

.r-params - rapportparametrar, attribut som ska listas

Testning utdata:

test.class-text-1 - den gamla klassningen nu tillämpad på testdatamängden redovisad som korsreferens från klasser till sessioner

3.3.6 Tolkning av Autoclass C:s klassificering - predStat

Korsreferensfilen .class-text-1 från Autoklass C var mycket stor och därför skrevs ett verktyg i C för att snabbt sammanställa datamängden och räkna ut hur många procent rätt och fel en klass hade i sin skattning.

Eftersom Autoclass skickar med märkningen av varje session gjord av Snort i utskriften så kan detta lätt korreleras mot klassen vid korsjämförelser. En tom sträng angav normal trafik utan anmärkning, och fick anta värdet "-" i resultatfilen. Alla andra strängar som "godtyckligt angrepp" fick sålunda indikera ett intrång eller intrångsförsök som detekterats av Autoclass.

Programmet användes för att generera statistik så att klassindelningen kunde bestämmas efter träningen av algoritmen. Alla klasser måste i träningsfasen delas in i de två typerna alarm respektive normal. Denna indelning specificerades i en fil *idslist* som indata till predStat. Om det var något fel på idslist var det enkelt att ändra den och göra om statistiken.

Samma fil, idslist, användes sedan som indata för att göra statistik för testresultatet. Den här gången var den låst och fick inte ändras vid träningen.

3.4 Mätning av klassificeringen

För att kunna mäta hur pass bra de klasser var som Autoclass C levererade var valde vi att använda Snort som mall. Snort matades med samma träningsdata som Autoclass C och resultaten jämfördes rad för rad i varje klass. Om Snort detekterat en rad som intrång markerades motsvarande rad i klassfilen.

Sedan räknades antalet markeringar och det totala antalet rader i klassen. Klassen som sådan kategoriserade vi sedan som larm eller normal beroende på om $markeringar \leq ej\ markeringar$. För att kunna få ett begrepp om klassificeringens kvalitet beräknades ett effektivitetsvärde på följande sätt, där M = totalt antal objekt i en klass och N = antal klasser.

$$K_{eff} = \frac{1}{N} \sum_{i=1}^{N} \text{abs}(\frac{markeringar}{M} - 0,5) + 0,5 \tag{3.1}$$

Ekvationen ger medelvärde mellan 0,5 och 1. Ett värde nära 0,5 innebär att varje klass har ungefär lika många markerade rader, som omarkerade. En klass med nära 50 % markeringar kan inte användas för klassificering av okända objekt eftersom den spänner över lika stora delar av båda grupperna.

Ju högre medelvärdet är desto bättre klara den av att klassificera nya data med liknande egenskaper. Värdet ska ses som att i K_{eff} % av fallen väljer Autoclass C rätt klass för ett okänt objekt. K_{eff} kan vara lägre än 0,5 om det ingår klasser utan några objekt i medelvärdet. Det är dessutom ett tecken på att klassen inte är särskilt väl anpassat till de data som använts.

Effektivitetsvärdet hade kunnat beräknas på många olika sätt. Vi valde att använda Ekvation (3.1) för att den visar förhållandet mellan gruppen av larm och normal trafik, oberoende om det är larm eller normal trafik som är den större. Likaså ville vi att ekvationen skulle ge värdet 1 för grupper med bara en typ av objekt (larm eller normal trafik).

4 Resultat

I det här kapitlet redovisas resultaten av undersökningen. Det kommer från de fyra utvärderingsomgångar vi genomförde. De har fått följande benämningar (se även Kapitel 3.3.4):

Träning kallas även "train", Snort kördes på samma data som träningen gjordes på.

Test1 är gjord med data från DARPA 98 datamängden, dock ej träningsdata.

Test2a gjordes på de egna data vi samlat in från det ena av våra nät. Insamlingen avslutades efter 30 sekunder eftersom angreppsfrekvensen var hög.

Test2b utfördes med egna data insamlade under en längre tid från ett nät med lägre angreppsfrekvens.

Kommentarer och diskussion av resultaten har vi placerat i Kapitel 5.

4.1 Detektering av intrång

Efter att ha använt ekvation (3.1) på de klassfiler som Autoclass C genererat vid träningen respektive testningen visade det sig att 24 av 55 klasser i träningsklassificeringen inte innehöll några data. Fördelningen av klasser med träffar blev 22 av normaltyp och 9 av larmtyp. Samma klasstruktur användes vid testningen, men då blev antalet klasser med träffar alltid färre än vid träningsklassificeringen.

I Tabell 4.1 visas klassificeringseffektiviteten för tränings- och testdata. Medelvärdena har vi beräknat för de ursprungliga 22 respektive 9 klasserna för jämförelse skull.

Tabell 4.1 visar att klasserna från träningsdatamängden gav högre värden för normalklasserna än för larmklasserna. Likaså syns en tydligt avvikelse i form av att testdatamängderna där vi använde våra egna data från 2004 inte gav några som helst träffar i någon larmklass. De möjliga orsakerna till resultatet kommer att diskuteras i Kapitel 5.

4.1.1 Träning

Träningsdatautvärderingen visar hur pass bra Autoclass C klarade av att hitta klasser som stämde överens med vad Snort kunde hitta i datamängden. Som vi skrev ovan saknades det värden för 24 av de totalt 55 klasser som Autoclass

Tabell 4.1: Klassificeringseffektiviteten för träningsdata och testdata

Typ	Träning	Test1	Test2a	Test2b
Antal larm	9	7	0	0
Antal normal	22	14	2	4
Larm; medel [%]	*65,8*	*46,5*	*0*	*0*
Larm; max [%]	80,7	80,7	0	0
Larm; min> 0 [%]	51,0	50,1	–	–
Normal; medel [%]	*83,7*	*50,4*	*9,5*	*15,8*
Normal; max [%]	100	100	97,6	100
Normal; min> 0 [%]	58,8	52,0	50,0	69,6
Total medel	76,6	49,3	6,8	11,2

C numrerat i stigande ordning. Klassificeringsfilen listade bara de klasser som sedan fått en sannolikhet större än noll.

Värdena för klasstillhörighet för de objekt som ingick i respektive klass fanns alla i området från cirka 40 % till 100 %.

4.1.2 Test1

Den första testkörningen utfördes som vi förklarat tidigare på data från samma datamängd som träningen utfördes på. Här lyckades Autoclass C hitta objekt tillhörande 14 av 22 normalklasser och 7 av 9 larmklasser. Snort detekterade intrång i ungefär samma utsträckning som för träningsdatamängden.

Värdena för klasstillhörighet för de objekt som ingick i respektive klass fanns även här alla i området från cirka 40 % till 100 %. Dock fanns det en förskjutning av tyngdpunkten mot högre värden än för träningsutvärderingen.

4.1.3 Test2a

Den här testkörningen utfördes på egna data insamlade på ett nät med hög angreppsfrekvens. På 30 sekunder hade 15000 loggposter samlats in. Det skiljde ungefär fem år mellan datamängden i test1 och de i den här datamängden. 1998, då test1 samlades in fanns inte MS-SQL maskar, så åtminstone den attacken blev ny och okänd för Autoclass C.

Autoclass C hittade bara objekt som tillhörde 2 normalklasser. Värdena för dessa två låg dock på mellan 80 och 100 %.

4.1.4 Test2b

Testet utfördes på data från ett av våra egna nät med en lägre angreppsfrekvens än det i test2a. Autoclass C hittade objekt som tillhörde fyra normalklasser, dock inte några larmklasser. Det senare beroende på att angreppstyperna i den här datamängden var desamma som för test2a. Även dessa data var nya och liksom för test2a detekterade Snort därför flera angrepp som inte fanns i DARPA 98 datamängden.

Värdena för de klasser som hittades låg i området mellan 65 % och 100 %.

5 Diskussion

Detta projekt har varit mycket lärorikt. Det har handlat om ämnen som IDS (intrångsdetekteringssystem), datafusion, analys av nätverkstrafik samt granskning av diverse användbara nätverksverktyg. Framför allt har projektet gett oss en god inblick i de problem och möjligheter som konstruktion av ett intrångsdetekteringssystem medför.

Vi har även fått ökad kunskap om Bayes sats och dess tillämpningar. Kunskapen kan användas inom flera områden, men det är främst inom intrångsdetekteringsområdet vi tycker oss se möjligheter till konkret användning för vår del.

5.1 Resultat av undersökningen

De resultat av undersökningen som presenterades i föregående kapitel diskuteras närmare i det här kapitlet. För att underlätta läsningen har vi separerat diskussionen i underrubriker på samma sätt som i resultatdelen.

Anledningen till att 24 av de totalt 55 klasser som Autoclass C angav i klassfilen var tomma misstänker vi beror på den princip som används för klassificeringen. Algoritmen slumpar fram en klassfördelning som sedan testas för att se hur den passar de objekt som används för träningen. Det kan då bli så att det någon gång under körningen funnits så mycket som 55 klasser. Dessa har sedan testats och 24 av dem har inte passat de data som använts.

Att antalet klasser vid testerna sedan alltid var lägre än vid klassframställningen bör ha berott på att klasserna som hittats inte var optimalt anpassade till de nya datamängderna. Det är dessutom så att antalet klasser inte kan öka eftersom de fastställts vid första körningen. Det här gör att nya angrepp kan klassificeras som angrepp men inte nödvändigtvis i rätt klass.

Processen att framställa en klassificering av träningsdatamängden tog så lång tid att vi bara han utföra ett fåtal tester med andra data än de från DARPA 98 uppsättningen. Det gör att underlaget för en mer kvalificerad utvärdering av prestanda hos klassificeringssteget inte hann göras. Dock märkte vi att det gick avsevärt mycket fortare, flera magnituder, än att ta fram klasserna. Allting tyder på att algoritmen kan passa för anomalidetektering i realtid eller nära realtid när väl klasserna har specificerats. Om däremot en kontinuerlig uppdatering av klassernas sammansättning ska ingå i intrångsdetekteringen blir det troligtvis svårare att uppfylla dylika krav. Detta måste dock testas mer ordentligt innan det går att avgöra definitivt.

En anledning till att körningarna tog tid var att vi hade så mycket mer data än vad Autoclass C förmodligen är tänkt att hantera. Konstruktörerna nämner i dokumentationen att algoritmen kan börja gå långsamt om antalet

rader data gånger antalet attribut att klassificera efter närmar sig 10000. Vi hade ibland värden på närmare 700000.

Så höga värden kan mycket väl förekomma inom intrångsdetekteringsområdet. Det ställer stora krav på de förbearbetningsfunktioner som används, de måste kunna komprimera datamängden utan att information går förlorad i alltför hög grad. Vi har inte haft möjlighet att studera problemet närmare utan har gjort våra undersökningar på okomprimerad data.

5.1.1 Mätmetod

För att mäta klasseffektiviteten använde vi Ekvation (3.1). Det finns andra sätt att mäta informationsinnehåll på, men vi valde den metoden för att den var enkel och visade på hur pass mycket överlapp mellan larm och normaltrafik varje klass innehöll. Ett värde på 75 % betyder att klassen har 75 % av sin vikt på ena sidan (larm eller normal). Som vi beskrev i Kapitel 3.4 innebär värdet också att Autoclass C gissar rätt i 75 % av fallen.

Valet av Snort för att hitta angrepp i den redan färdigklassade datamängden gjorde vi för att enkelt kunna jämföra de data från helt olika insamlingstillfällen och tidpunkter. En nackdel med metoden var dock att Snort med en uppdaterad angreppsdatabas detekterar till exempel telnettrafik som angrepp. När DARPA 98 datamängden framställdes klassades dylik trafik som helt legitim eftersom telnet var ett av nätverksadministratörernas favoritredskap.

Telnettrafik i sig utgör i och för sig inte ett intrång, men i och med att all trafik sker i klarttext med telnet används inte applikationen av administratörer i samma utsträckning längre. I stället har SSH och liknande applikationer med krypterad kommunikation tagit över. Det är förmodligen därför Snort klassificerar telnettrafik som intrång nu för tiden. Med stor sannolikhet finns det fler exempel på hur olika protokoll och program gått från att vara administratörernas favorit till att bli paria.

Alternativet till att använda en modern version av Snort hade varit att hitta en version från 1998. Den hade dock inte kunnat hitta alla nya angrepp i våra egna data. Resultatet hade därför blivit missvisande och förmodligen innehållit en minst sagt stor mängd falska negativ, alltså missad angrepp.

Det tredje alternativet, att använda en version av Snort anpassad till respektive datamängds insamlingstidpunkt hade gjort det omöjligt att jämföra resultaten från de olika körningarna, måttstocken hade inte varit densamma. Eftersom vi var intresserade av att se hur pass bra en gammal träningsdatamängd kunde ge klasser som fungerade även med nya data var vi tvungna att använda en modern version av Snort på båda datamängderna.

Anledningen till att vi använde så helt olika tränings- och testdatamängder var att vi ville se hur Autoclass C kunde hantera verkliga data och inte bara tillrättalagda data. Som resultatet visade var tyvärr datamängderna alltför olika för att vi skulle kunna dra några mer djupgående slutsatser. Det enda vi kan säga är att träningsdatamängden måste vara mer lik testdatamängden än den vi använde.

5.1.2 Träning

Träningsdatamängden passade av förklarliga skäl bra att köra en provklassificering på. På så sätt kunde vi få en bild av Autoclass C:s förmåga att hitta klasser i en så stor datamängd. Klasserna för normaltrafik hade effektivitetsvärden som varierade mellan 58,8 % och 100 % med ett medelvärde på 83,7 %, vilket får anses som bra. Medelvärdet för normal trafik och angrepp tillsammans hamnade på 76,6 %.

Siffrorna kanske inte ser mycket ut för världen, i vart fjärde fall placerar algoritmen objektet i fel klass. Det här gäller dock för en icke optimerad uppsättning dataattribut. Eftersom vi använde samma attribut för alla datamängder spelade inte valet av dem någon roll för det inbördes resultatet, alla datamängder hade samma förutsättningar.

Ser vi till hur pass bra algoritmen själv beräknade att klasserna var anpassade till de data som användes indikerade värdena en mycket hög överensstämmelse. Det här innebär att de egenskaper eller dataattribut vi valt inte var optimala, vilka som gett bättre resultat vet vi tyvärr inte i dagsläget. För att kunna bygga ett användbart intrångsdetekteringssystem måste de optimala egenskaperna för en korrekt klassificering hittas.

5.1.3 Test1

Test1 nådde nästan upp till 50 % för den totala klassificeringseffektiviteten i medeltal. Det är inte något bra resultat, men de bästa respektive sämsta klasserna med objekt i hade ungefär samma effektivitetsvärde som vid träningstillfället. Det var bara färre klasser som innehöll objekt och därmed fick något värde.

Med bättre anpassade klasser hade detektionsgraden stigit. Dessutom användes en modern version av Snort, så minst en av angreppsklasserna var klassad som normaltrafik vid datainsamlingstillfället 1998. Hade vi använt en version av Snort som var anpassad för datamängden hade förmodligen resultatet sett bättre ut.

5.1.4 Test2a

Det här var den datamängd som bestod av data insamlade under 30 sekunder i ett av våra nät. Resultatet var inte direkt lysande. En av orsakerna till det dåliga resultatet var att det skiljde ungefär fem år mellan träningsdatamängden och testdatamängden. Med andra ord innehöll testdatamängden helt nya sorters angrepp som inte passade de klasser som träningsdatamängden gett upphov till. Det här är naturligvis ett problem, klasserna måste uppdateras med jämna mellanrum. Exakt hur en sådan uppdatering ska gå till och hur ofta den måste göras behöver utredas vidare.

En annan orsak till det dåliga resultatet var det mycket korta insamlingsperioden. Vi fick ungefär 15000 angrepp under den tiden och det gör att den knappast var en normal fördelning av angrepp och normal trafik. De angreppstyper det handlade om var två pingskannar och en mask som försökte sprida sig. Masken som försökte sprida sig ingick definitivt inte i DARPA 98-datamängden. Detta i kombination med den stora tidsskillnaden mellan

träningsdatamängden och testdatamängden gjorde att resultatet blev undermåligt.

5.1.5 Test2b

Här användes data insamlade i våra nät under längre tid än för test2a. Det gjorde att resultatet blev något bättre än för det tidigare testet. Medelvärdet för den totala klassificeringseffektiviteten var nära dubbelt så bra som för det tidigare testet. Det var dock inte i närheten av av resultatet för de data som var från samma år. Här märktes alltså återigen hur skillnaden i tid mellan de två datamängderna försämrade resultatet märkbart.

Dock var den sämsta klassificeringseffektiviteten för en klass 69,6 % för de fyra klasser som innehöll data. De klasser som faktiskt användes var alltså riktigt bra anpassade till datamängden, trots åldersskillnaden.

5.2 Praktiska erfarenheter

För att kunna utföra den praktiska delen av algoritmanalysen köptes en specialbyggd dator in som fick namnet *algoritmdatorn*. Den optimerades för relativt tunga beräkningar och snabb lagring av stora datamängder. Datorn blev också snabb, men med facit i hand hade det krävts ännu större beräkningskapacitet, kanske i form av ett Linuxkluster eller i alla fall en flerprocessorlösning. När datorn sattes att träna på 60000 rader data och 11 attribut belastades processorn till 99.9 % i 10 timmar. Det var å andra sidan väntat med tanke på vad dokumentationen sa om tidskomplexiteten för algoritmen.

Datorn algoritmen kördes på hade gott om diskutrymme så där uppstod inte några problem. På routern däremot blev diskutrymmet fort fullt och därför planerade vi att sätta upp ett SSH baserat NFS (Network File System) mellan routern och algoritmdatorn. Det lades dock på is efter att det visat sig att Autoclass C oförklarligt kraschade efter ett par timmars körning. Alltså kunde vi inte använda alltför stora datamängder, i alla fall inte i närheten av de veckor av data som vi från början planerat att samla in.

Lyckligtvis visade sig Autoclass C inte behöva några större mängder data att träna på för att kunna göra godtagbara klassificeringar av okända data. Det gällde dock bara när träningsdata och testdata var från samma tidsperiod (datamängd). När data från den fem år nyare datamängden användes blev resultatet sämre. Det märktes tydligt att trafikmönster, angreppstyper och systemanvändning skilde sig mellan datamängderna.

Helst hade vi velat utvärdera flera algoritmer och testa dem mot varandra. Det hade gett oss mer kunskap om hur pass bra de resultat vi nådde var. I vissa fall bestod de klasser Autoclass C hade hittat av i stort sett lika stora delar larm och normal trafik. Frågan är om någon annan algoritm kunnat göra en bättre klassificering. Likaså hade det varit önskvärt att kunna testa med fler och mer lika datamängder.

En provisorisk lösning för att öka detektionsgraden kan vara att filtrera bort data som klassificeras i en klass med sämre effektivitetsvärde än 0,75 och skicka dessa till en annan algoritm. Värdet 0,75 kommer från ett snabbtest av den föreslagna lösningen där olika värden testades och värdet 0,75 gav den bästa kompromissen mellan falska positiv (falsklarm) och falska negativ

(missade riktiga intrång). När värdet användes för filtrering av DARPA 98 datamängden klarade algoritmen av att korrekt klassificera 49,3 % av trafiken.

Den här studien har handlat om post-mortem-testning, det vill säga detekteringen har utförts på loggdata i efterhand. Det vore intressant att modifiera Autoclass C att lyssna på nätverkstrafiken i realtid istället. Träningsprocessen kan utföras på en kraftfull dator i förhand och sedan sker själva detekteringen i till exempel en router eller en brandvägg. En gång i månaden eller kvartalet utförs sedan en ny träningsomgång och en ny uppsättning klasser kan distribueras till de detekterande processerna.

Den svåraste delen av projektet var att hitta relevanta och lämpliga attribut att utföra träningen och sedan detekteringen på. Attributen var tvungna att tydligt skilja på normal trafik och intrång, samtidigt som de inte skulle variera för mycket inom sitt intervall. Det var inte heller helt lätt att formatera de olika datamängderna så att de gick att träna på eller att över huvud taget jämföra. För att kunna utföra formateringen fick ny programvara skrivas och det adderade naturligtvis ett extra element av osäkerhet till utvärderingsresultaten. De program som skrevs var dock aldrig större än att vi kunde ha god kontroll över utvecklingen och resultatet av algoritmstudien ska inte har påverkats.

6 Framtida arbete

Eftersom den här undersökningen bara tilldelades 400 timmar totalt, var det mycket som inte kunde studeras i tillräckligt hög grad. Några av dessa saker kommer vi att presentera närmare i det här kapitlet.

6.1 Prestanda

Våra mätningar skulle behöva göras med mer data för att riktigt kunna utvärdera Autoclass C:s prestanda. Den datamängd som användes för testning av algoritmen skulle behöva utökas för att ge en mer korrekt bild av ett system i normalbruk. Våra egna data innehöll dessutom bara en bråkdel av vanlig, icke angreppsrelaterad trafik, vilket inte gav en riktig bild av verkligheten. Helst av allt skulle vi vilja använda data samlade från en större nätverk i skarp miljö under flera månader. På så sätt skulle vi få med förändringar i användningsprofilen på över tiden och kunna anpassa klasserna efter periodiciteten i användningstrenderna. Det kan till exempel handla om att vissa arbetsuppgifter bara utförs en viss veckodag, eller en vid en viss tid på året. Även ledigheter och andra perioder av återkommande låg eller hög arbetsintensitet kommer att synas.

Det vore också intressant att testa algoritmen i realtidsmiljö. Dock kräver det att en komplett IDS utvecklas från algoritmen. Testet skulle kunna utföras i laboratoriemiljö där det inte gör någonting om det sker intrång. Det viktiga är att testa algoritmen på riktiga data. I och för sig skulle det kunna ersättas av den datainsamling från ett skarpt nät som vi beskrev ovan, men den kan inte ge den tidsmässiga strukturen som en installation i skarp miljö kan ge. Vid en dylik installation får vi automatisk också ett belastningstest av algoritmen.

Ett annat problem som måste lösas är hur inlärningen kan ske kontinuerligt under drift. Användningsprofilen för ett system förändras över tiden på grund av förändrade arbetsuppgifter, uppdateringar av programvara, nyinstallation av hårdvara med mera. Allt eftersom den normala användarprofilen ändras kommer den ursprungliga klassificeringen som gjorts att ge upphov till ett ökande antal falsklarm eller missade angrepp. Därför måste olika sätt att underlätta eller automatisera processen med att anpassa algoritmen till nya förhållanden tas fram.

En mycket viktig egenskap hos detektionsalgoritmer är deras förmåga att klara av hög belastning. Autoclass C:s förmåga till att kunna skalas upp måste också utredas. I dagsläget har den bara testats på data från ett litet nät med låg trafikintensitet. Det är därför nödvändigt att utföra belastningstest både i kontrollerad miljö och i skarp miljö med högre belastning.

Eftersom datamängden för träning var från 1998 och de data som använ-

des för test var från 2004 fanns det med stor sannolikhet angrepp i testdatamängden som inte fanns träningsdatamängden. Undersökningen skulle behöva utökas så att algoritmen kan tränas med nyare typer av angrepp. Det skulle ge möjlighet att studera hur väl den kan hantera nya och för den okända angreppstyper. Som det är nu har algoritmen inte kunnat utvärderas ordentligt.

Det som gäller för nya angreppstyper i datamängden gäller också för rent felaktiga data. Autoclass C:s förmåga att kunna hantera saknade eller korrupta data behöver utredas mer. Detta är viktigt eftersom det finns en möjlighet att en angripare försöker förblinda ett intrångsdetekteringssystem genom att göra ett överlastningsangrepp med felaktiga data.

Ytterligare en sak som vore intressant att studera är om det finns några skillnader mellan att använda Autoclass C som en nätverksbaserad intrångsdetektor och att använda den som en värddatorbaserad detektor. I praktiken innebär det att utföra samma tester som beskrivits i den här rapporten, med ovanstående tillägg, men i stället använda systemloggar, till exempel Syslog eller Eventlog. Det är mycket möjligt att de två typerna skulle skilja sig åt, eftersom systemloggar i allmänhet har en mer regelbunden struktur och färre variationsmöjligheter än nätverksloggar.

6.2 Implementation

Vad gäller implementationen av ett intrångsdetekteringssystem byggt kring Autoclass C så måste framför allt förbehandlingsstegen utredas mer. Det gäller sådana saker som vilka egenskaper eller attribut som bäst lämpar sig för detektering av intrång, hur de ska formateras för att passa algoritmen och om de behöver sammanställas på något sätt.

Det behövs även funktionalitet för att ta hand om och tolka det resultat som Autoclass C levererar. Hur dessa funktioner ska konstrueras för att göra detektionen så effektiv som möjligt behöver också studeras närmare. Likaså behövs kanske funktioner för att kunna koppla samman flera detektorer, både sådana baserade på Autoclass C och sådana som bygger på någon annan algoritm. Det skulle i så fall även behövas någon passande standard för kommunikation mellan dessa detektorer.

Sammankopplingen av detektorer leder vidare till en djupare studie av krav och riktlinjer för distribuerad intrångsdetektering. Här skulle det vara intressant att se på möjligheten att distribuera klassificeringen eller träningen av Autoclass C för att bättre utnyttja den hårdvara som finns i systemet. I nuläget har Autoclass C krav på minnesutrymme och processorkraft som gör att enklare och därmed billigare hårdvara inte kan användas. Om den kunde distribueras skulle det eventuellt gå att åtgärda detta.

Ytterligare en sak som behöver studeras är om det redan finns färdiga intrångsdetekteringssystem som använder Autoclass C. Eftersom det för närvarande forskas intensivt inom intrångsdetekteringsområdet är det mycket möjligt att någon annan forskningsgrupp har planerat att använda eller redan använder Autoclass C. Det vore därför värdefullt att få kontakt med dem för att kunna utbyta erfarenheter.

Ett sak som togs upp tidigare (se Kapitel 6.1) var hur problemet med en automatisk anpassning av Autoclass C till rådande användningsprofil skulle gå

till. Denna funktion skulle behöva byggas in i de funktioner som skulle stödja Autoclass C vid användning i ett intrångsdetekteringssystem. En tänkbar lösning är att använda data som omfattar en fast tidsperiod av tidigare händelser i systemet. Vad som skulle behöva utredas är om det är en möjlig lösning och i så fall vad för parametrar som ska styra urvalet av data. Likaså måste en lösning hittas för hur algoritmen ska kunna tränas utan att dess detekterande funktion måste avbrytas.

6.3 Testbänk

För att kunna genomföra den utökning av testerna som föreslås ovan på ett så effektivt sätt som möjligt vore det önskvärt med någon slags testbänk för intrångsdetekteringssystem och algoritmer. Den skulle vara modulärt uppbyggd så att den gick enkelt att uppdatera eller anpassa till olika förhållanden och algoritmer. På så sätt skulle den tillåta testning av både färdiga intrångsdetekteringssystem, såväl som delar av dylika.

Om en testbänk enligt ovan beskrivna modell ska utvecklas behöver också dess exakta funktionalitet utredas i detalj. För närvarande har vi bara en övergripande idé om vilka funktioner som skulle vara bra att ha. Eftersom testbänken ska vara generell för alla slags typer av intrångsdetekterande algoritmer är det viktigt att den är genomtänkt redan från början. Utvecklingen av den behöver därför drivas som ett eget projekt.

Inte bara funktionaliteten behöver specificeras i detalj, även hårdvaran måste väljas med omsorg med tanke på den modulära uppbyggnaden av systemet. Tanken är att det ska gå att lägga till enheter för att simulera stora heterogena system när till exempel belastningsprov och skalningstester ska utföras. Det ställer krav på att hårdvaran ska vara kompatibel med en stor mängd olika typer och märken av hårdvara.

Naturligtvis måste det också göras en studie för att utröna om det redan finns några testbänkar som motsvarar de krav som ställs. Eftersom forskningen inom IDS-området är intensiv är det troligt att det redan existerar någon slags testbänk som kanske kan modifieras för att passa våra behov.

6.4 Andra algoritmer

Det vore ur flera synvinklar intressant att studera andra koncept och algoritmer inom intrångsdetekteringsområdet. För det första måste Autoclass C jämföras med andra algoritmer för att vi ska kunna bilda oss en uppfattning om hur de ska rangordnas inbördes. För det andra kan det uppstå synergieffekter när principer från en algoritm kombineras med principer från en annan algoritm. För det tredje skulle en sådan studie leda till att försvarsmakten fick tillgång till en ytterligare ökad kompetens inom området. Dessutom skulle undersökningen i sig naturligtvis ge en bild av utvecklingen av de produkter som finns på marknaden idag, vilket vore bra med tanke på planerna på att bygga NBF med COTS-produkter.

Vidare så behöver de krav som en heterogen, dynamisk och distribuerad miljö i stil med NBF ställer på ett intrångsdetekteringssystem utredas i grunden. En sådan undersökning skulle sedan leda vidare till en studie av hur ett

intrångsdetekteringssystem bäst anpassas för att möta dessa krav. Undersökningen skulle också omfatta utvärderingar av de idag befintliga systemen och deras förmåga att uppfylla de ovan nämnda kraven.

Något som ligger längre fram i tiden är att faktiskt utveckla ett intrångsdetekteringssystem som är helt anpassat för NBF:s behov. Ett sådant arbete skulle dock behöva initieras snarast för att det ska kunna nå resultat i tid innan införandet av NBF.

Litteraturförteckning

[1] Kdd cup 1999 data, 1999. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html` läst 2003-11-19.

[2] *SEARCHING FOR GOOD CLASSIFICATIONS*, januari 2002. Ingår som en del i Autoclass C:s källkodspaket. Filen heter *search-c.text*.

[3] Bayes-gruppen. Autoclass c - general information, 2003. `http://ic.arc.nasa.gov/projects/bayes-group/autoclass/autoclass-c-program.html`, läst 2004-01-23.

[4] Gunnar Blom. *Sannoliksteori med tillämpningar – A*. Studentlitteratur, 1984.

[5] Brian Caswell och Marty Roesch. Snort^{TM} – the open source network intrusion detection system, 2004. `http://www.snort.org/` läst 2004-01-15.

[6] die.net. tcpdump(8) - linux man page, 2004. `http://www.die.net/doc/linux/man/man8/tcpdump.8.html` läst 2004-03-03.

[7] die.net. tcpslice(8) - linux man page, 2004. `http://www.die.net/doc/linux/man/man8/tcpslice.8.html` läst 2004-03-03.

[8] Robert Durst, Terrence Champion, Brian Witten, Eric Miller och Luigi Spagnuolo. Testing and evaluating computer intrusion detection systems. *Communications of the ACM*, 42(7):53–61, juli 1999.

[9] faqs.org. Rfc 793 - transmission control protocol, 2004. `http://www.faqs.org/rfcs/rfc793.html` läst 2004-03-03.

[10] Jiawei Han och Micheline Kamber. *Data Mining – Concepts and Techniques*. Academic Press, 2001.

[11] Kari Marklund, redaktör. *Nationalencyclopedin*. Bokförlaget Bra Böcker, 1990.

[12] John McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, november 2000.

[13] snort.org. Snort.org signature search, 2004. `http://www.snort.org/cgi-bin/sigs-search.cgi` läst 2004-03-04.

[14] Marc Zissman. 1998 darpa intrusion detection evaluation data set overview, 1998. `http://www.ll.mit.edu/IST/ideval/data/1998/1998_data_index.html` läst 2003-11-19.