CODEN:LUTEDX/(TEIE-5258)/1-39/(2008)

Industrial Electrical Engineering and Automation

En säkerhetsanalys av E1000 terminal och dr-250 adsl router

Axel Villamo

Dept. of Industrial Electrical Engineering and Automation
Lund University

En säkerhetsanalys av E1000 terminal och dr-250 adsl router.

Axel Villamo

16 oktober 2008

Abstract

The need of communication between automation systems is growing rapidly. Since a lot of systems are using Internet, which is a open network, security is in focus. Customers of the company Beijer Electronics AB had troubles with reliability of the operator terminals when they were connected to Internet. This master thesis is a security analysis of the operating terminals and the dr-250 adsl router from the company Westermo which is in the Beijer Electonics group. Regarding the operator terminal, the focus has been on the ftp and http servers with debugging and code corrections. For the router, the IPsec protocol and VPN functionality have been in focus.

Sammanfattning

Det finns ett ständigt ökande behov av kommunikation inom distribuerade automationssystem. I många fall används här Internet som sammanbindande länk. Detta sätter fokus på säkerheten hos de anslutna produkterna på ett sätt som inte provas vid anslutning inom företagsinterna skyddade nätverk. Kunder som använder Beijer Electronics operatörsterminaler har i vissa fall rapporterat problem vid anslutning till Internet.

Inom detta examensarbete har gjorts en säkerhetsanalys av operatörsterminaler från Beijer Electronics samt av ADSL-routern dr-250 från Westermo. För terminalerna har ftp- och http-servrarna felsökts och korrigerats medan implementationen av IPsec och VPN hos routern har granskats.

Förord

Det är med stor glädje jag presenterar detta examensarbete som är den sista delen i min civilingenjörsutbildning i elektroteknik. Det är väldigt kul för mig att kunna presentera en lösning som bidrager till en bättre produkt.

Jag skulle vilja tacka alla inblandade i arbetet, mina handledare Jonas Lindgren och Håkan Jeppson på Beijer Electronics, Gunnar Lindstedt min handledare på IEA Institutionen samt min examinator Henriette Weibull. Jag vill också tacka samtliga på teknikavdelningen på Beijer Electronics för en trevlig tid.

Lund den 16 oktober 2008

Axel Villamo

Innehåll

1 Inledning **1**
- 1.1 Bakgrund . . . 1
- 1.2 Problematik och syfte . . . 1
- 1.3 Fokus och avgränsningar . . . 1
- 1.4 Målgrupp . . . 2

2 Teori **3**
- 2.1 E1000 operatörsterminal . . . 3
  - 2.1.1 Hårdvara . . . 4
  - 2.1.2 Mjukvarustruktur . . . 4
  - 2.1.3 Kodens uppbyggnad . . . 4
  - 2.1.4 Systemprogrammet . . . 5
  - 2.1.5 E-Designer . . . 6
- 2.2 DR-250 ADSL router . . . 6
  - 2.2.1 DSL tekniker . . . 7
- 2.3 FX3U-PLC . . . 8
  - 2.3.1 GX IEC Developer . . . 9
- 2.4 VPN nätverk . . . 9
- 2.5 TheGreenBowVPN client . . . 9
- 2.6 IPsec . . . 10
  - 2.6.1 IKE - Internet Key Exchange . . . 10
  - 2.6.2 SA - Security Association . . . 11
  - 2.6.3 SPI - Security Parameter Index . . . 11
  - 2.6.4 ESP - Encapsulation Security PayLoad . . . 11
  - 2.6.5 AH - Authentication Header . . . 13
- 2.7 DNS - Domain Name System . . . 14
  - 2.7.1 Dynamisk DNS . . . 16

3 Säkerhetsproblem i terminalen **17**
- 3.1 Autentisering vid ftp login . . . 17
- 3.2 Hantering av oönskade data . . . 17

4 Utförda tester **18**
4.1 Belastningstest av http och ftpserver i E1000 terminal 18
4.2 IIS exploits 19
4.3 Aggressive Mode VPN med DR-250 20

5 Felsökning och kodändringar **22**
5.1 Funktionen xn_line_get() i IPstacken 22
5.1.1 Beskrivning av funktionen 22
5.1.2 Hur problemet uppstår 22
5.1.3 Ursprunglig kod 22
5.1.4 Ändringar 25
5.1.5 Modifierad kod 25

6 Slutsats och diskussion **28**

7 Referenser **29**

8 Appendix A **31**
8.1 Exploitkod 31
8.1.1 Exempel 1 31
8.1.2 Exempel 2 31

Tabeller

2.1 OSI Modellen 10
2.2 ESP header 12
2.3 Transport Mode före ESP 12
2.4 Transport Mode efter ESP 12
2.5 Tunnel Mode före ESP 12
2.6 Tunnel Mode efter ESP 13
2.7 Authentication header 13
2.8 Transport Mode före AH 14
2.9 Transport Mode efter AH 14
2.10 Tunnel Mode efter AH 14

4.1 Resultat av test1 och test2 19

Figurer

2.1 E1060 Terminal på en storlek av 6 tum, 320x240 pixlar med funktionstangenter. 3
2.2 Vy över BlockManager i E-Designer 7
2.3 DR-250 ADSL router från Westermo 7
2.4 Bandbreddsuppdelning för ADSL2 8
2.5 DNS trädstuktur 15

1 Inledning

1.1 Bakgrund

Beijer Electronics AB[1] är ett ledande företag i Norden som tillhandahåller system för industriautomation. Företaget är uppdelat i Beijer Electronics Automation AB, Beijer Electronics Products AB och IDC (Industrial Data Communications). HMI Products utvecklar operatörsterminaler som säljs över hela världen, men tack vare ett nära vägg i vägg samarbete med Beijer Automation AB kan produkterna skräddarsys efter kundernas krav och målsättningar. I slutet av 2007 köpte Beijer Electronics AB upp Westermo Electronics AB[2], som nu är en del av Beijerkoncernen och går under benämningen IDC. Även externa sammarbetspartners finns. Beijer Electronics har distributionsrättigheterna för Mitsubishi Electric i Norden och Baltikum.

I examensarbetet ingår operatörspanel och PLC system från Mitsubishi samt kommunikationsutrustning från Westermo.

1.2 Problematik och syfte

Kunder med operatörspaneler från Beijer Electronics AB har problem med icke tillfredsställande funktion när de är uppkopplade mot Internet. Ett problem är att terminalerna hänger sig när de är uppkopplade mot Internet. Examensarbetet ska ta reda på var i problemet består, hur det uppkommer och lösa det. Syftet är att Beijer Electronics terminaler skall bli säkra vid kommunikation över Internet.

1.3 Fokus och avgränsningar

Fokus ligger i säkerhetsaspekten då en E1000 operatörsterminal kopplas upp mot Internet med en DR-250 ADSL-router från Westermo. Det har gjorts

[1] www.beijer.se
[2] www.westermo.com

ett 'Kom igång dokument' för uppkopplingen som används vid nyinstallation. Tester på terminaler har utförts och källkoden till terminalerna har felsökts.

1.4 Målgrupp

Målgruppen för denna rapport är de som har teknisk bakgrund i automationssystem och datorkommunikation. Läsaren bör också känna till datorsäkerhetsbegrepp som exploits, bufferoverflows och DoS-attacker.

2 Teori

2.1 E1000 operatörsterminal

E1000 terminalerna[1] är en serie som finns i olika utföranden. Av siffrorna i namnet, E1XXY, står XX för terminalens storlek i tum. Om Y är ett jämnt tal är terminalen utrustad med funktionstangenter, om Y är 1 har terminalen touchfunktion. De olika terminalerna har mycket gemensamt. De är uppbyggda på en plattform med Windows CE 4.2 och programmeras i samma utvecklingsmiljö, E-Designer. Programmet är utvecklat av Beijer Electronics Products men marknadsförs av Mitsubishi Electric.

Figur 2.1: E1060 Terminal på en storlek av 6 tum, 320x240 pixlar med funktionstangenter.

[1] www.beijerelectronics.com

2.1.1 Hårdvara

Terminalerna som tillhör serien E1000 finns i varierande storlekar och utföranden. Med funktionstangenter finns de i storlekar från 3 tum med 240x64 pixlar till 10 tum med 800x600 pixlar. De med touchfunktion finns i storlekar från 3.5 tum med 320x240 pixlar till 15 tum med 1024x768 pixlar. Processorn som driver terminalerna är Intel XScale PXA270. Det är en energisnål processor för implementering på kretskort. Den finns i hastigheterna 416 och 316 MHz. De större modellerna har den snabbare processorn.

Terminalen är robust uppbyggd med IP66 klassificering för fronten och IP20 för kåpan. Terminalen drivs av 24 VDC eller 20-30 VAC. Det finns jordskruv på baksidan och terminalen uppfyller EMC EN 61000-2-6 och EN 61000-6-4.

Till terminalerna finns följande anslutningar:

- RS232C och RS422/RS485
  Via de seriella portarna RS232 och RS422 ansluts terminalen till en PLC eller mottagare av seriell kommunikation exempelvis en dator. E1000 terminalerna har drivrutiner för att kommunicera med de flesta kontrollsystem från de stora tillverkarna Mitsubishi, Siemens, Allen-Bradley , Delta Tau etc.

- USB Host typ A och USB Device typ B
  Till USB kan anslutas HID interface, Printers och Storage Devices.

- Ethernet RJ45 10/100 Mbit
  Standard ethernet som uppfyller IEEE 802.3 standarden

- Compact flash typ I och II
  För överföring av projekt och lagring av trendkurvor.

2.1.2 Mjukvarustruktur

Mjukvaran i E1000 terminalserien är baserad på Windows CE 4.2. I högre lager ovanför CE finns en applikation som benämns Systemprogram. I Systemprogrammet finns ett projekt, detta utvecklas i E-Designer. För att projektet skall kunna kommunicera med styrsystemet körs en drivrutin för det specifika styrsystemet i projektet.

2.1.3 Kodens uppbyggnad

Koden som terminalen bygger på är ett realtidssystem framtaget för små integrerade system skrivet av EBSNet [2]. Det har filsystem support, en TCP/IP stack, FTP och HTTP server. Koden användes ursprungligen till föregående

[2] http://www.ebsnetinc.com

terminalserie, E-serien. Dessa terminaler har inte Windows CE i botten utan är helt uppbyggda kring koden från EBSNet. Vid framtagandet av E1000 serien konverterades koden till Windows CE miljö.

2.1.4 Systemprogrammet

För att användaren av terminalen inte skall se Windows CE startas ett egenutvecklat skal, beijershell.exe. Från detta startas sedan systemprogrammet, bepp.exe, som innehåller allt i terminalfunktionaliteten. Parallellt med bepp.exe körs ett program, watchdog.exe, som kontrollerar att bepp.exe fungerar som det skall. Skulle bepp.exe av någon anledning hänga sig eller avslutas så startar watchdog.exe om bepp.exe.

Kommunikationstjänster i systemprogrammet
De tjänster som finns i terminalen beskrivs nedan. De ligger alla i bepp.exe, inga externa program körs för någon tjänst.

- Application Transfer Server port 6000
  Via denna server laddas projekt över från E-Designer via ethernet. Är inte denna server igång måste projekt laddas ner seriellt vilket tar betydligt längre tid.

- BDTP Server och Client port 6002
  BeijerDataTransferProtol används för kommunikation mellan E1000-terminaler. Med hjälp av detta protokoll kan en server och en klient utbyta värden på ett sätt som är lätt att konfigurera via E-Designer.

- FTP Server port 21
  FileTransferProtocol är ett standard protokoll som används till filöverföring. Via FTP nås de filer som finns på terminalen. Det går att köra 5 stycken FTP instanser parallellt.

- Remote Access port 5800 web och 5900 vnc client.
  Remote access för terminalen. Anropas terminalen via port 5800 i webbläsaren ses en spegling i en java applikation, TightVNC Desktop[3]. Tyvärr går inte funktionstangenterna att använda i denna applikation. Via Port 5900 körs programmet Remote Access Viewer, som simulerar funktionstangenter så att även andra terminaler som saknar touchfunktion fungerar.

- SMTP Client port 25
  SimpleMailTransferProtocol är ett protokoll för att sända och ta emot email. E1000 terminalerna är endast klienter dvs. de kan skicka mail men inte ta emot.

[3] http://www.tightvnc.com/

- Terminal Controller port 6001
  Används för att byta mellan körläge och överföringsläge på terminalen.
- Transparent Mode port 6004
  Används för kommunikation genom terminalen i nätverk med paneler.
- Webbserver port 80
  Webbservern ger möjlighet att läsa och skriva värden på terminalen med CGI eller SSI skript. Upp till 5 stycken instanser fungerar parallellt.

2.1.5 E-Designer

Programmet E-Designer är utvecklat av Beijer Electronics AB, men marknadsförs genom Mitsubishi Electric. Designen torde vara bekant för programmerare då den följer standard utvecklingsmiljö. När ett projekt är skapat, väljs vilken terminal projektet skall användas i och vilket styrsystem som terminalen är kopplad till. Det går i efterhand att ändra till en annan terminalversion eller annat styrsystem, men det kan uppstå konverteringsproblem så det är inte att rekommendera.

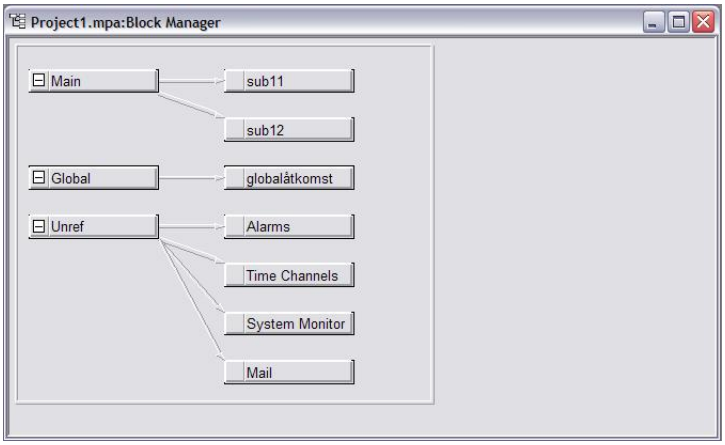
Programmeringen sker i block. Det finns ett antal fördefinierade block, Main, Alarms etc. Mainblocket är det som syns när terminalen startas. Under Mainblocket läggs sedan nya block i en trädstuktur som är åtkomliga nedåt och uppåt i strukturen från det aktiva blocket. Det finns också möjlighet för globala block som är åtkomliga oavsett vilket block som är aktivt. I blocket skapas symboler som textfält, knappar, indikatorer, värdefält, diagram etc. Insignaler kan avläsas i terminalprogrammet i ett värdefält. Utsignaler och dataregister kan ställas direkt i styrsystemet. Om en signal läses in i ett diagram från styrsystemet lagras detta lokalt i terminalen och kan hämtas ner därifrån via ftp.

2.2 DR-250 ADSL router

DR-250 är en ADSL router framtagen för industriellt bruk. Den stödjer både ADSL2 och ADSL2+, vilket ger en överföringshastighet nedströms på 24 Mbit/s och 3.5 MBit/s uppströms. Det finns också en inbyggd 3G modul med dubbla simkortplatser.

Routern har många funktioner

- IPSec VPN support för upp till 200 tunnlar.
- DES, 3DES, AES, SSL, SSH krypteringsalgoritmer.
- Avancerad brandvägg



Figur 2.2: Vy över BlockManager i E-Designer

- SNMP protokoll
- RS-232 terminal server port.

Figur 2.3: DR-250 ADSL router från Westermo

2.2.1 DSL tekniker

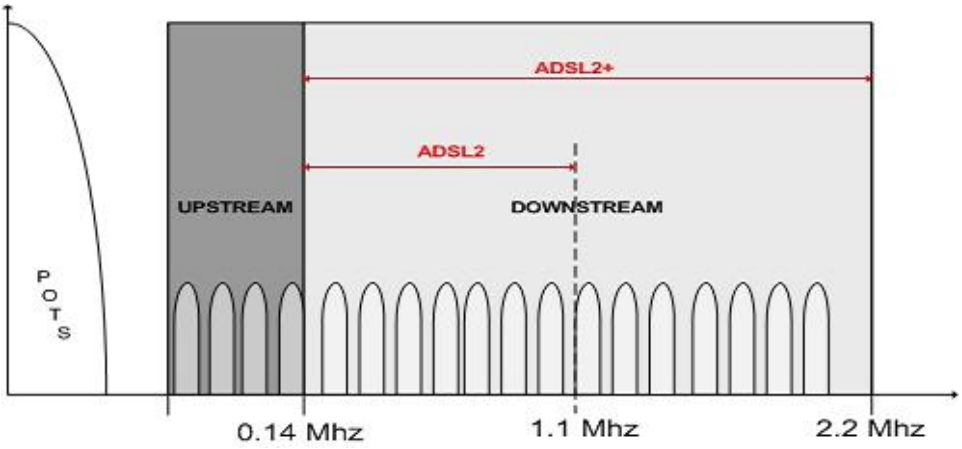
Digital Subscriber Line är ett samlingsnamn för de olika datakommunikationstekniker över telefonledningar som finns. Det finns två huvudgrupper: **S**ymmetrisk **DSL** och **A**symmetrisk **DSL**. Skillnaden mellan dessa är hur mycket bandbredd som finns tillgänglig i de olika riktningarna. SDSL har lika mycket bandbredd tillgänglig nedströms som uppströms. Den uppdelningen gör att vanlig telefonkommunikation inte kan ske parallellt, vilket är möjligt med ADSL.

Asymmetrin i ADSL ger en högre nedströms hastighet än uppströms. Tekniken utvecklades för Video-On-Demand till privatpersoner. Då de flesta kunder efterfrågar denna typ av teknik har den blivit totalt dominerande. Detta innebär att de kopplingsstationer för telefoni som finns runt om i landet har möjlighet för ADSL och inte SDSL. SDSL vore dock en mer fördelaktig

teknik att använda för den typ av datatrafik som exempelvis kommunikation med en operatörsterminal genererar.

Modulationen i ADSL är uppbyggd kring FDM (Frequency Division Multiplexing) vilket innebär att flera olika signaler skickas på samma ledning men på olika frekvenser. De varianter av FDM som används i ADSL är CAP (Carrierless Phase/Amplitude Modulation) och DMT (Discrete Multitone Technology) som båda bygger på QAM (Quadrature Amplitude Modulation). QAM är en kombination av fasmodulation och amplitudmodulation. Genom att variera både amplitud och fas skapas betydligt fler kombinationer än endast en modulationsmetod. Fyra amplitudlägen och fyra faslägen ger 16 olika kombinationer, 16-QAM, vilket ger 15 bitar/Hz. CAP är en icke-standard implementation av QAM som nu inte används längre.

Istället används **DMT** som också bygger på QAM men är mer förfinad. Istället för att använda hela bandbredden i ett stort fält delar DMT upp denna i mindre intervall. För vanlig ADSL, 0 – 1.1 MHz, är bandbredden uppdelad i 256 intervall, där de sex första intervallen är dedicerade till telefoni, 7 – 38 till uppströmstrafik och resten till nedströmstrafik.



Figur 2.4: Bandbreddsuppdelning för ADSL2

2.3 FX3U-PLC

FX3U är ett kompakt PLC system från Mitsubishi Electric. Med kompakt menas att det är mindre, ej utbyggbart och inte har samma funktionalitet och prestanda som de större systemen. Till FX3U går det att ansluta moduler för DA/AD omvandling, ethernetkommunikation etc.

2.3.1 GX IEC Developer

GX IEC Developer är ett utvecklingsprogram för Mitsubishis PLC produkter. Programmeringsspråken som stödjs är Ladder, Structured Text och Sequential Function Chart. De olika språken har sina för- och nackdelar. Då ladder är väldigt enkelt att programmera i och ger en god översikt är ladder det mest utbredda språket i industrin. I GX IEC Developer är Ladder samt Structured Text också de språk som har bäst stöd och har flest funktioner implementerade.

Funktionsblock är en användbar möjlighet att implementera funktioner, exempelvis en PID kontroller eller Profibuskommunikation, snabbt och enkelt. De vanligaste funktionerna som en programmerare kan tänkas använda finns fördefinierade i standardbibliotek. Beijer tillhandahåller bibliotek innehållande funktionsblock som rör deras produkter och andra stora aktörers.

2.4 VPN nätverk

VPN - Virtual Private Network är ett sätt att skapa en tunnel för trafik över ett nätverk. Med VPN kan ett internt nät skapas som kommunicerar över ett öppet nät exempelvis Internet. Tunnlarna i sig är varken krypterade eller autentiserade. För att åstadkomma det krävs det andra underliggande protokoll som IPSec eller SSL/TLS.

Den enklaste uppställningen av ett VPN är mellan två noder på ett nät, som sätts upp med var sin VPN programvara så att de kan nå varandra på andra adresser än de ursprungliga. Ett exempel är följande. Nod A har adress 84.55.116.24 och Nod B har adress 84.55.116.41 ut mot internet. Ett subnet på adress 10.4.0.0 skapas med VPN programvaran, där Nod B tilldelas 10.4.0.1 och Nod A 10.4.0.2. Då kan nod A och nod B nå varandra på de nya klass A adresserna.

Om exemplet utvecklas så noderna A och B också är routrar för interna nät i samma klass som VPN nätet kan alla klienterna bakom noderna också nå varandra. På så vis kan två olika interna nät kopplas samman över internet.

2.5 TheGreenBowVPN client

TheGreenBow VPN är en programvara för att ansluta en dator mot VPN baserade på IPSec. Programmet finns i en 30 dagars prova på licens annars kostar det 58 euro. Programmet har används för att ansluta mot DR-250 routern.

2.6 IPsec

IPSec (IP security) är ett protokoll för att kommunicera säkert på ett IP nät. Det sker genom att autentisera och/eller kryptera varje IPpaket. IPsec ligger i lager 3 (nätlagret) i OSI modellen, ett lager under TLS/SSL som ligger i lager 4 (transportlagret). Protokollen som ligger över lager 3 (TCP, UDP, applikationer mm.) blir således helt opåverkade av IPsec. Detta är en fördel gentemot lager 4 protokollen då applikationen inte behövs specialdesignas.

IPsec använder sig av två protokoll 'Authentication Header' (AH) och 'Encapsulation Security PayLoad' (ESP). ESP måste stödjas av IPsec implementationen, men AH är alternativt (det behövs inte stödjas). Detta är beroende på att AH har begränsningar som gör det svårimplementerat i generella fall och därför inte används i lika stor utbredning. Protokollen kan användas var för sig eller tillsammans. AH ger bekräftelse till mottagaren att headern är autentisk, dvs att paketet kommer från rätt avsändare. ESD krypterar datapaketet vilket ger en skyddad överföring men har dessutom autentiseringmöjlighet som AH.

Lager 7	Applikation
Lager 6	Presentation
Lager 5	Session
Lager 4	Transport
Lager 3	Nät
Lager 2	Datalänk
Lager 1	Fysiska skiktet

Tabell 2.1: OSI Modellen

2.6.1 IKE - Internet Key Exchange

Initialt när en IPSec förbindelse öppnas har noderna inga SA. För att kunna skapa dessa krävs ett utbyte av säkerhetsinformation. Här kommer IKE in i bilden genom att skapa delade säkra nycklar. Vid skapandet av nycklarna används alltid Diffie-Hellman algoritmen, med de för anslutningen specificerade parametrarna. I kommunikationen finns det två parter, en som startar *initiatorn* och en som svarar *respondern*. Det är initiatorn som bestämmer vilka parametrar som skall användas i överföringen, förutsatt att mottagaren stödjer dem.

När Diffie-Hellmanhandskakningen och en säker förbindelse mellan noderna är upprättad vet de fortfarande inte att de har kontakt med rätt nod.

Kontrollen sker efteråt genom autentisering av en nyckel. Det finns 5 olika alternativ för autentisering, de vanligaste är pre-shared keys. När autentiseringen är klar, fas ett, kan SAs skapas, fas två.

Fas ett kan ske på två olika sätt, Main Mode eller Aggressive Mode. Main Mode använder sig av sex paket för att skapa en anslutning och Aggressive Mode använder sig av tre stycken. Det gör Main Mode mer flexibelt och Aggressive Mode snabbare. Med Aggressive Mode måste både parterna ha samma inställningar, men i Main Mode kan parterna komma överens själv. Dessutom kan parterna i Main mode neka att de haft informationsutbyte med varandra även om handskakningsförfarandet är sparat hos endera parten. Detta kan vara en bra funktion om anonymitet önskas. I RFC 2407[4] och RFC 4306[5] är IKE ytterligare beskrivet.

2.6.2 SA - Security Association

SA är basen i IPSec, de fungerar som kontrakt mellan de kommunicerande noderna. Genom dessa bestäms enliget vilket protokoll kommunikationen skall ske, ESP eller AH, och vilka algoritmer detta skall använda. Om två klienter A och B kommunicerar med IPSec så har noderna två SA, SA_{in} och SA_{out}, där SA_{in} hos A motsvarar SA_{out} hos B och vice versa. Om både ESP och AH används finns det en SA för varje protokoll. Lagringen av de olika SA sker hos respektive klient i en databas, kallad SADB.

2.6.3 SPI - Security Parameter Index

För att SA skall fungera krävs att sändaren vet vilken SA den skall använda och att mottagaren använder den samma, SPI löser detta. SPI är ett 32-bitars tal som skickas med varje paket och som mottagaren använder för att lokalisera rätt SA i sin SADB.

2.6.4 ESP - Encapsulation Security PayLoad

ESP pakethuvudet används för att ge möjlighet till både kryptering och autentisering av paketen. De tre möjliga kombinationerna är beskrivna nedan med förklaring.

- Kryptering
  Stöds inte av alla produkter då det är en osäker lösning som ger användaren en falsk trygghet. Tillsammans med ett överliggande verifieringsprotokoll kan dock en säker överföring skapas, men det är ovanligt att det används.

[4] http://tools.ietf.org/html/rfc2407
[5] http://tools.ietf.org/html/rfc4306

- autentisering
  Alla produkter på marknaden stödjer autentisering vilket är ett bra alternativ till AH då dataöverföringen är snabbare och har bredare funktionalitet. NAT är möjligt bland annat.

- Kryptering och autentisering
  Alla produkter på marknaden stödjer kryptering och autentisering tillsammans. Oberoende av val av algoritm och krypteringslängd är detta det vanligaste sättet man ser ESP användas.

I tabell 2.2 ses ESP pakethuvudet med dess ingående komponenter. I pakethuvudet före ESP måste protokollnummret 50 anges, vilket är numret för ESP.

0 -7 bit	8-15 bit	16-23 bit	24-31 bit
Security Parameters Index (SPI)			
Sequence Number			
Payload Data (Variable length)			
	Padding (0-255 bytes)		
		Pad Length	Next Header
Integriy Check Value-ICV (variable length)			

Tabell 2.2: ESP header

IP header	TCP	Data

Tabell 2.3: Transport Mode före ESP

IP header	ESP header	TCP	Data	ESP Trailer	ESP ICV
		<– Krypterat –>			
	<– Autentiserat –>				

Tabell 2.4: Transport Mode efter ESP

IP header	TCP	Data

Tabell 2.5: Tunnel Mode före ESP

New IP hdr	ESP hdr	Org IP hdr	TCP	Data	ESP Trailer	ESP ICV
		<– Krypterat –>				
	<– Autentiserat –>					

Tabell 2.6: Tunnel Mode efter ESP

2.6.5 AH - Authentication Header

AH används för att få integritet hos dataöverföringen. Det innebär att mottagaren kan vara säker på att paketet kommer från rätt avsändare och innehållet inte är ändrat. Vilken sorts kodning som används är den som stödjs av båda parter, detta sköts av IKE. Autentiseringen sker genom att sändaren räknar fram en hashsumma av de beständiga fälten i IPheadern, vilket kan göras med MD5 eller SHA-1 etc. En hemlig nyckel som är känd av både sändare och mottagare sammanfogas med hashnycklen och paketet skickas. Proceduren för mottagaren är då att räkna fram samma hashsumma av paketdatan och jämföra. Stämmer värdet med det som sändaren skickade är paketet autentiskt.

Ett problem är att AH inte fungerar med NAT då fälten i IPheadern som hashsumman beräknas på ändras. Skall AH användas genom en router måste routern känna till den hemliga nyckeln och räkna om hashsumman vilket är direkt olämpligt. För att systemet skall anses säkert skall routern inte känna till nyckeln utan bara de två parterna. Detta ses i tabell 2.8 och 2.9 för AH då de beständiga fälten, vilka inkluderar käll och måladress, i IP headern är hashsummerade och därför inte kan ändras.

Headern direkt utanför AHn måste innehålla värdet 51 så att AH är specificerat. Vid konfiguration av brandväggar är det viktigt att inte glömma det som villkor så att datatrafiken kommer genom.

0 -7 bit	8-15 bit	16-23 bit	24-31 bit
Next header	Payload length	Reserved	
Sequence number			
Security parameters index (SPI)			
Integrity Check Value - ICV			

Tabell 2.7: Authentication header

Förklaring till hur AH headern är uppbyggd:

- Next header
  Beskriver pakethuvudet efterföljande AH, dvs specificerar om det rör sig om ett TCP, UDP eller något annat protokoll.

- Payload Length

Här specifieras hur långt AH pakethuvudet är.

- Reserved
  Utrymmet är reserverad för framtida bruk.
- Security Parameters Index (SPI)
  Ett tal specifieras här för att mottagaren skall kunna identifiera den SA (Security Association) som det inkommande paketet är bundet till.
- Sequence Number
  Är ett tal som räknas upp för varje paket som är sänt.
- Integrity Check Value
  Är checksumman (hashnyckeln) som används för att verifiera paketet.

Beroende om transportmode eller tunnelmode används ser de totala paketen olika ut. I tabellerna 2.8, 2.9 och 2.10 ses uppbyggnaden av paketstrukturen och vad i paketet som är autentiserat.

Org IP header	TCP	Data

Tabell 2.8: Transport Mode före AH

Org IP header	AH	TCP	Data
<– Autentiserat förutom vissa fält i IP headern –>			

Tabell 2.9: Transport Mode efter AH

New Ip header	AH	Org IP header	TCP	Data
<– Autentiserat förutom vissa fält i nya IP headern –>				

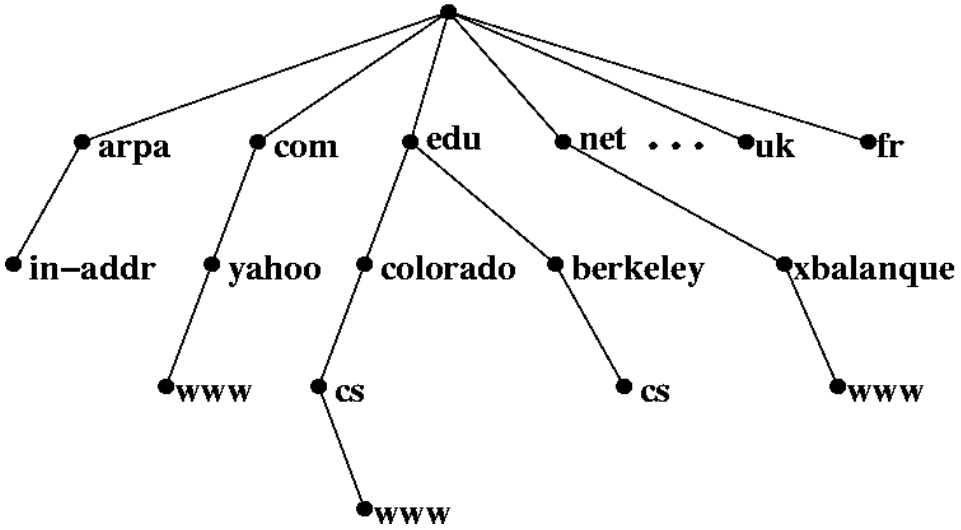
Tabell 2.10: Tunnel Mode efter AH

2.7 DNS - Domain Name System

DNS är ett system för att koppla namnadresser till IPnummer. Initialt tillhandahöll NIC (Network Information Center) filen HOSTS.TXT som alla värdar hämtade hem och använde som referens. I takt med att det tillkom fler värdar växte filen snabbt och värdarna blev tvungna att uppdatera den oftare. Denna enorma tillväxt av kommunikationen mot NICs HOSTS.TXT blev ohållbar. Dessutom uppkom ett behov av att lokala administratörer kunde lägga till värdar utan att kontakta NIC.

En flexiblare lösning som inte överbelastade en specifik server togs fram och används fortfarande. Huvudkomponenterna i DNS är tre stycken.

- Domain Name Space
  Domain Name Space är namnet på den trädstruktur av domännamn som finns. Varje nod är en namnserver som innehåller information om domänen och dess värdar. Det är möjligt att det finns underdomäner som innehåller information som inte den övre domänen har lokalt utan hänvisar till den undre om den informationen efterfrågas.



Figur 2.5: DNS trädstuktur

- Namn Servrar
  Namn servrar är serverprogram som innehåller information om domänens struktur och värdar. En namnserver kan temporärt lagra information om vilken del som helst av domänträdet, men i praktiken så har en namnserver endast information om de underliggande värdarna och hänvisar till andra namnservrar för information som ligger utanför dess område. Området som namnservern har ansvar för är indelat i zoner. Det är för att ge systemet redundans. Om en namnserver går ner så skall en annan alltid ha zonens information tillgänglig.

- Resolvrar
  Resolvrar är de program som utläser DNS information från namnservrar och svarar klienterna. En resolver måste ha kontakt med minst en namnserver för att kunna göra förfrågningar som sedan vidarebefordras. Generellt är en resolver ett program som körs lokalt på en klient.

2.7.1 Dynamisk DNS

De uppkopplingar som erbjuds idag har ofta dynamisk tilldelning av ipnummer. När ipaddressen ändras blir det omöjligt att komma åt den datorn utifrån om den inte själv skapar kontakt. Med dynamisk dns ges möjligheten att låta uppdatera namnet hos namnservern enkelt när ipnumret ändras. Det sker genom att ett program på klienten med dynamisk adress ofta skickar sitt ipnummer till namnservern.

Det finns flera olika företag som erbjuder denna typ av tjänster. Ett av dem är dyndns[6] som erbjuder gratis dynamisk dns tillgång. Dyndns är en erkänt bra och utbredd tjänst som många routrar har kommunikations stöd för. Så även Westermos DR-250.

[6] www.dyndns.org

3 Säkerhetsproblem i terminalen

3.1 Autentisering vid ftp login

Vid anslutning till ftpservern finns en säkerhetsrisk i autentiseringen. Anges korrekt användarnamn men felaktigt lösenord påtalar servern att lösenordet är felaktigt. Anges ett användarnamn som inte finns (vilket automatiskt gör lösenordet felaktigt) påtalar servern att användarnamnet eller lösenordet är felaktigt. Praxis vid inloggning är att alltid använda samma svar oavsett om användarnamnet finns eller ej, vilket ger en mycket högre säkerhet. Om en attack för att försöka få tillgång till servern genom att prova olika användarnamn och lösenord med en ordlista med längden x, är attackhastigheten $O(x)$ så som servern är implementerad i nuläget. Om servern följt praxis hade attackhastigheten varit $O(x^2)$. Tiden för att genomföra attacken ändras alltså till kvadraten av ursprungstiden, vilket är högst rekommendabelt att ändra till. Detta är inte ett nytt problem, det fick uppmärksamhet redan 1979, men likaväl är det fortfarande aktuellt.

3.2 Hantering av oönskade data

En klient som är kopplad ut mot internet blir mottagare av oönskad trafik. Det kan röra sig om trafik som oavsiktligt skickas till fel mottagare. Det finns scannrar på internet som söker efter klienter att utnyttja. Typiska svagheter som scannrarna letar efter kan vara exploits i vanliga typer av serverapplikationer som FTP och HTTP. Den trafik som kommer in till terminalen misstänker man blir så pass stor att klienten inte kan hantera den, därför utförs ett belastningstest. Test mot förekommande exploits testats också.

4 Utförda tester

4.1 Belastningstest av http och ftpserver i E1000 terminal

FTP- och HTTPservern som finns i systemprogrammet Bepp.exe utsätts för ett stresstest. Det sker genom att öppna upp till 5 stycken tcp förbindelser till port 80 resp 21 med programmet netcat[1]. Belastningen på terminalen ökar desto fler förbindelser som öppnas samtidigt. Den information som skickas över tcpförbindelsen läses från en binärfil à 256kB innehållande endast nollor. För att starta en tcp förbindelse körs kommandot 'nc 192.168.1.1 80 $<$*binärfil*'.

För att testa hur funktionen i terminalerna påverkas laddas en minimal applikation in, vilken har 3 stycken block som kan alternera. När det uppstår en fördröjning större än 1 s vid växling mellan block anses terminalen vara blockerad och ej längre funktionell. Då testen görs på både tangentbaserade och touchbaserade terminaler har applikationerna olika utföranden vad gäller hur växlingen mellan blocken sker. Applikationen testas också med olika antal tjänster igång. Dessa benämns test1 och test2. Test1 applikationen har endast projektöverföring, HTTP och FTP, medan test2 har samtliga tjänster igång.

De modeller av terminalen som testats är följande. Touchbaserade: 1043, 1063, 1071, 1101, 1151. Tangentbaserade: 1032, 1060, 1070, 1100.

Resultat
Terminalerna reagerar likartat på testerna. Felen är deterministiska och lika för alla terminaltyperna. Den skillnad i hårdvara som finns mellan 106x och nedåt gentemot de större terminalerna gör att de mindre modellerna är mer känsliga, vad gäller antalet anslutningar som de tål utan att bli långsamma. När terminalen belastas med trafik på respektive port reagerar terminalerna långsamt på knapptryckningar. Vad som är anmärkningsvärt är att terminalerna inte kommer tillbaka till sitt normaltillstånd efter att anslutningarna avslutats. Efter att testerna har utförts har terminalerna oberverats i mer än

[1] http://www.vulnwatch.org/netcat/

15 timmar. Att döma av testerna som gjorts, så kommer de aldrig tillbaka utan förblir i ett långsamt stadium tills de startas om.

Terminal	Test1 HTTP	Test 1 FTP	Test 2 HTTP	Test 2 FTP
1043	misslyckat	misslyckat	misslyckat	misslyckat
1063	misslyckat	misslyckat	misslyckat	misslyckat
1071	misslyckat	misslyckat	misslyckat	misslyckat
1101	misslyckat	misslyckat	misslyckat	misslyckat
1151	misslyckat	misslyckat	misslyckat	misslyckat
1032				
1060	misslyckat	misslyckat	misslyckat	misslyckat
1070	misslyckat	misslyckat	misslyckat	misslyckat
1100	misslyckat	misslyckat	misslyckat	misslyckat

Tabell 4.1: Resultat av test1 och test2

4.2 IIS exploits

IIS är en vanligt förekommande webbserver som det funnits ett antal säkerhetshål i. Exploits som fångats upp i loggfiler har testas mot olika terminalers HTTPservrar och FTPservrar. Vissa säkerhetshål är väldigt gamla. I november 2003 annonserade Microsoft en buggfix mot sina IIS servrar [2], trots åldern på buggfixen finns det fortfarande infekterade servrar som sprider denna kod vidare. Säkerhetshålet i IIS servern är av bufferoverflow karaktär.

Ett antal olika exploitrader har testas, se appendix A för ytterligare information.

Ett utdrag från en av raderna är följande
"'SEARCH /\x90\xc9\xc9\xc9\xc9\xc9\xc9 ..."'
SEARCH används för att tillkalla den utsatta programvaran , \xc9 är NOP dvs no operation. Efter en mängd NOP följer binär hex kod som körs av servern. Antal tecken kan uppgå till 29000, dvs 29 kB.
En annan rad är

```
'GET /NULL.IDA?CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC\
%u0aeb\%ub890\%udacf\%u77ee\%u ..."'
```

Resultat
Dessa exploits borde inte direkt påverka terminalerna då de inte kör IIS

[2] http://www.microsoft.com/technet/security/bulletin/MS03-051.mspx

applikationer, men det gör de. När en terminal anropas med någon av nämnda exploits svarar inte terminalen på anropet. När fem stycken anrop har skickats till terminalen svarar terminalen inte heller längre på vanliga anrop. Detta gäller för både FTP- och HTTP-servern. Då terminalen inte kör IIS och således inte borde vara drabbad, undersöks felet noggrannare genom felsökning av IPstacken i terminalen.

4.3 Aggressive Mode VPN med DR-250

Då VPN används är inte terminalen i fokus, då den är bakom VPN tunneln. Fokus ligger istället på säkerheten på VPN anslutningen i routern, som i detta fallet är konfigurerad med aggressive mode och pre-shared keys, ESP med autensiering (MD5) och kryptering (DES) används. En beskrivning av uppställningen finns att se i Kom-igång-dokument (se www.beijer.se). Brandväggen i DR-250 är konfigurerad med följande rader

pass break end on eth 4 from any to any port=4500 (VPN)
pass break end on eth 4 from any to any port=500 (IKE)
pass break end on eth 4 proto 50 (ESP)

Brandväggen är då blockerad såtillvida att endast de nödvändiga VPN-protokollen släpps genom. Med programmet ike-scan[3] fås hashsumman av lösenordet tillhörande anropet KEY_ID. Programmet psk-crack[4] används sedan mot hashsumman för att få fram lösenordet.

Resultat
Det är verifierat genom test att det går att knäcka lösenordet i aggressive mode om KEY_ID är känt. Det är därför av största vikt att ett KEY_ID väljs med sådan omsorg att det inte går att gissa sig till och har sådan längd att det är svårt att knäcka.

När routern anropas med ett icke existerande KEY_ID svarar den:

```
#ike-scan -A -id finnsej e1000beijer.dyndns.org
Starting ike-scan 1.7 with 1 hosts (http://www.nta-monitor.com/ike-scan/)
90.227.106.218  Notify message 18 (INVALID-ID-INFORMATION)

Ending ike-scan 1.7: 1 hosts scanned in 0.035 seconds (28.35 hosts/sec).
0 returned handshake; 1 returned notify
```

När routern anropas med ett existerande KEY_ID svarar den:

```
#ike-scan -A -id finns e1000beijer.dyndns.org
Starting ike-scan 1.7 with 1 hosts (http://www.nta-monitor.com/ike-scan/)
90.227.106.218  Aggressive Mode Handshake returned SA=(Enc=3DES Hash=SHA1
```

[3] http://www.nta-monitor.com/ike-scan/
[4] http://www.nta-monitor.com/ike-scan/

```
Auth=PSK Group=2:modp1024 LifeType=Seconds LifeDuration(4)=0x00007080)
KeyExchange(128 bytes) Nonce(20 bytes) ID(Type=ID_IPV4_ADDR, Value=90.227.106.218)
Hash(20 bytes) VID=afcad71368a1f1c96b8696fc77570100 (Dead Peer Detection)
VID=12f5f28c457168a9702d9fe274cc0100 (Cisco Unity)

Ending ike-scan 1.7: 1 hosts scanned in 0.407 seconds (2.46 hosts/sec).
1 returned handshake; 0 returned notify
```

Alltså ger routern bekräftelse på om ett KEY_ID finns eller ej, vilket kan utnyttjas vid angrepp. Det vore önskvärt att routern skickade tillbaka en hashsumma även vid felaktigt KEY_ID för att öka säkerheten. Stöd för NAT-T som beskrivs i RFC3947 (NAT-T för IKE) borde också implementeras så att det är möjligt att köra Main Mode bakom NAT. Det skulle öppna möjligheter för bättre konfigureringar av VPN.

5 Felsökning och kodändringar

5.1 Funktionen xn_line_get() i IPstacken

5.1.1 Beskrivning av funktionen

Både FTP och HTTP servrarna använder sig av en funktion kallad xn_line_get() i IPstacken, filen rtipapi.cpp för att läsa in en kommandorad. Varje rad skall avslutas enligt standard, vilket är return följt av blanksteg som i C motsvarar \r\n.

5.1.2 Hur problemet uppstår

När en rad som innehåller fler tecken än konstanten CFG_ETHERSIZE, vilken är satt till 1513, matas in kommer funktionen i en loop som inte avslutas oberoende om förbindelsen bryts eller ej. Tråden som ligger i bakgunden upptar systemresurser i terminalen så att den reagerar långsamt. Om terminalen blir tillräckligt belastad så startas den automatiskt om av watchdog.exe.

I normalfallet skickas ej anrop som är så pass långa, men det sker. Däremot är långa anrop vanliga i exploits för webservrar, som testats tidigare. Grundproblemet är således inte exploiten i sig utan att IPstacken inte kan hantera långa strängar. Det är inte ett säkerhetsproblem i den meningen att obehöriga kan göra intrång, utan yttrar sig i att terminalen fryses.

5.1.3 Ursprunglig kod

```
/*****************************************************************          */
/* xn_line_get() - Read a line from a socket                                  */
/*                                                                            */
/* Summary:                                                                   */
/*   #include "rtip.h"                                                        */
/*                                                                            */
/*   int xn_line_get(PIO_CONTEXT io_context, PFCHAR buffer, int buflen, long wait) */
/*                                                                            */
/* Description:                                                               */
/*   Performs the following:                                                  */
/*     Reads data until LF or EOF (connection closed).                        */
/*     Sets buffer to the start of the data in io_context->pb_in.             */
```

```
/*                                                                          */
/*   NOTE:  It is recommended that xn_line_get always be called with        */
/*          tmeout = TRUE.                                                  */
/*                                                                          */
/* Returns:                                                                 */
/*   -1 on error, -2 if there was no data in the alloted time, -3 means     */
/*   the remote host is no longer connected, else returns number of         */
/*   bytes read.                                                            */
/*   Sets errno upon error                                                  */
/*                                                                          */
/*                                                                          */

int xn_line_get(PIO_CONTEXT io_context, PFCHAR *buffer, long wait, int type)
{
int n, len;
int n_left;
PFCHAR slash_r;

    if (type == GET_BUF)
    {
        return(line_get_buf(io_context, buffer, wait));
    }

    /* type must be GET_LINE    */
    for (;;)
    {
#if (!POLLOS)
        /* yield; if the remote host is blasting data across and we have   */
        /* a full input window we must yield to let the IP task run        */
        /* since this loop will not block for a period of time             */
        ks_yield();
#endif

        /* strings should end with \r\n                    */
        /* if no \r\n in what we have read, read more data */
        slash_r = tc_strstr((PFCHAR)(io_context->pb_in+io_context->begin_offset_in),
                            "\r\n");
        if (!slash_r)
        {
            /* calculate how much room until end of buffer; leave room   */
            /* for \0                                                    */
            n_left = CFG_ETHERSIZE - io_context->end_offset_in - 1;

            if (n_left)     /* if more room to read */
            {
                /* if remote host is not responding, avoid recv() hanging   */
                if (!do_read_select(io_context->sock, wait))
                {
                    return(-2);
                }

                /* use the socket API to get another buffer of data         */
                /* NOTE: there is no option here to use the packet API      */
                /*       since there is not guarentee that a line           */
                /*       will not cross packets and be in two packets;      */
                /*       to handle this case would be messy and would       */
                /*       involve copying anyway and input done by this      */
                /*       routine is not the bulk of the data transmissions  */
                /*       going on anyway                                    */
                n = recv(io_context->sock,
                         (PFCHAR)(io_context->pb_in +
                                  io_context->end_offset_in),
```

```
                    n_left, 0);
            if (n < 0)
            {
                return(-1);
            }

            io_context->end_offset_in += n;
            io_context->pb_in[io_context->end_offset_in] = '\0';

            /* if EOF (no data and otherside closed)   */
            if (n == 0)
            {
                if (io_context->end_offset_in >
                    io_context->begin_offset_in)
                {
                    /* return what we have   */
                    slash_r = (PFCHAR)(io_context->pb_in +
                                       io_context->end_offset_in);
                    break;
                }
                else
                    return(-2);
            }

            /* strings should end with \r\n    */
            slash_r = tc_strstr((PFCHAR)(io_context->pb_in+io_context->begin_offset_in),
                                "\r\n");
        }
    }

    /* we have filled the buffer from current postion to end,    */
    /* check if we now have a \r                                 */
    if (slash_r)
    {
        *slash_r = '\0';
        break;
    }
    else
    {
        /* if this is true then the buffer is empty so reset it   */
        if (io_context->end_offset_in <= io_context->begin_offset_in)
        {
            io_context->end_offset_in = io_context->begin_offset_in = 0;
            io_context->pb_in[0] = '\0';
        }
        else
        {
            /* move unread data to beginning of buffer and go try again   */
            /* NOTE: +1 in length to ensure moving \0 at end of string    */
            tc_movebytes(io_context->pb_in,
                         io_context->pb_in+io_context->begin_offset_in,
                         io_context->end_offset_in -
                         io_context->begin_offset_in + 1);
            io_context->end_offset_in -= io_context->begin_offset_in;
            io_context->begin_offset_in = 0;
        }
    }
}           /* end of while loop */

*buffer = (PFCHAR)(io_context->pb_in + io_context->begin_offset_in);
len = (int)(slash_r - *buffer);
io_context->begin_offset_in += (len + 2);
```

```
    return(len);
}
```

5.1.4 Ändringar

Problemet i koden löses genom att avsluta funktionen och returnera ett fel om bufferten är full och inget radavslut funnits. Inget svar kommer att ges till klienten som anropat terminalen, utan anslutningen avslutas endast.

5.1.5 Modifierad kod

```
/*************************************************************                */
/* xn_line_get() - Read a line from a socket                                  */
/*                                                                            */
/* Summary:                                                                   */
/*   #include "rtip.h"                                                        */
/*                                                                            */
/*   int xn_line_get(PIO_CONTEXT io_context, PFCHAR buffer, int buflen, long wait) */
/*                                                                            */
/* Description:                                                               */
/*   Performs the following:                                                  */
/*     Reads data until LF or EOF (connection closed).                        */
/*     Sets buffer to the start of the data in io_context->pb_in.             */
/*                                                                            */
/*   NOTE:  It is recommended that xn_line_get always be called with          */
/*          tmeout = TRUE.                                                    */
/*                                                                            */
/* Returns:                                                                   */
/*   -1 on error, -2 if there was no data in the alloted time, -3 means       */
/*   the remote host is no longer connected, else returns number of           */
/*   bytes read.                                                              */
/*   Sets errno upon error                                                    */
/*                                                                            */
/*                                                                            */

int xn_line_get(PIO_CONTEXT io_context, PFCHAR *buffer, long wait, int type)
{
int n, len;
int n_left;
PFCHAR slash_r;

    if (type == GET_BUF)
    {
        return(line_get_buf(io_context, buffer, wait));
    }

    /* type must be GET_LINE    */
    for (;;)
    {
#if (!POLLOS)
        /* yield; if the remote host is blasting data across and we have   */
        /* a full input window we must yield to let the IP task run        */
        /* since this loop will not block for a period of time             */
        ks_yield();
#endif

        /* strings should end with \r\n                    */
        /* if no \r\n in what we have read, read more data */
```

```
slash_r = tc_strstr((PFCHAR)(io_context->pb_in+io_context->begin_offset_in),
                    "\r\n");
if (!slash_r)
{
    /* calculate how much room until end of buffer; leave room   */
    /* for \0                                                    */
    n_left = CFG_ETHERSIZE - io_context->end_offset_in - 1;

    if (n_left)     /* if more room to read */
    {
        /* if remote host is not responding, avoid recv() hanging   */
        if (!do_read_select(io_context->sock, wait))
        {
            return(-2);
        }

        /* use the socket API to get another buffer of data         */
        /* NOTE: there is no option here to use the packet API      */
        /*       since there is not guarentee that a line           */
        /*       will not cross packets and be in two packets;      */
        /*       to handle this case would be messy and would       */
        /*       involve copying anyway and input done by this      */
        /*       routine is not the bulk of the data transmissions  */
        /*       going on anyway                                    */
        n = recv(io_context->sock,
                 (PFCHAR)(io_context->pb_in +
                          io_context->end_offset_in),
                 n_left, 0);
        if (n < 0)
        {
            return(-1);
        }

        io_context->end_offset_in += n;
        io_context->pb_in[io_context->end_offset_in] = '\0';

        /* if EOF (no data and otherside closed)   */
        if (n == 0)
        {
            if (io_context->end_offset_in >
                io_context->begin_offset_in)
            {
                /* return what we have   */
                slash_r = (PFCHAR)(io_context->pb_in +
                                   io_context->end_offset_in);
                break;
            }
            else
                return(-2);
        }

        /* strings should end with \r\n    */
        slash_r = tc_strstr((PFCHAR)(io_context->pb_in+io_context->begin_offset_in),
                            "\r\n");
    }
}

/* we have filled the buffer from current postion to end,    */
/* check if we now have a \r                                 */
if (slash_r)
{
    *slash_r = '\0';
```

```
            break;
        }
        else
        {
// Begin
         if ( n_left == 0 )
         {
         return(-1);
         }
// End
            /* if this is true then the buffer is empty so reset it   */
            if (io_context->end_offset_in <= io_context->begin_offset_in)
            {
                io_context->end_offset_in = io_context->begin_offset_in = 0;
                io_context->pb_in[0] = '\0';
            }
            else
            {
                /* move unread data to beginning of buffer and go try again   */
                /* NOTE: +1 in length to ensure moving \0 at end of string    */
                tc_movebytes(io_context->pb_in,
                             io_context->pb_in+io_context->begin_offset_in,
                             io_context->end_offset_in -
                             io_context->begin_offset_in + 1);
                io_context->end_offset_in -= io_context->begin_offset_in;
                io_context->begin_offset_in = 0;
            }
        }
    }           /* end of while loop */

    *buffer = (PFCHAR)(io_context->pb_in + io_context->begin_offset_in);
    len = (int)(slash_r - *buffer);
    io_context->begin_offset_in += (len + 2);

    return(len);
}
```

6 Slutsats och diskussion

Under examensarbetets gång har frågetecken kring terminalens säkerhet rätats ut. Genom att sätta mig in i utvecklingsmiljön för terminalprojektet (E-Designer) och GX-IEC har jag lärt mig grunderna för hur terminalerna och PLC används. Olika tester på terminaler har lett till upptäckt om vad som får terminalerna att frysa. Vid uppkoppling till Internet har problemet härletts till IIS exploits attacker. Detta stämmer mycket bra in på den felbeskrivning som fanns och förhoppningen är att problemet nu är borta. Naturligtvis kan det finnas andra fel som leder till problem vid uppkoppling mot internet utanför min kännedom, det går aldrig att frisäga sig från.

Om en säker kommunikation krävs, uppfyller inte terminalerna denna funktion själv. En bra lösning är en krypterad VPN tunnel. Med DR-250 routern kan man åtstadkomma detta. Det finns dock vissa synpunkter på implementationen av IPSec som kunde förbättras, implementation av RFC 3947, så att säkerheten och funktionaliteten på VPN nätet blir ännu bättre.

Ett problem som inte går att komma ifrån när kryptering av datortrafik sker är att mängden överförd data blir väldigt mycket större. Datorlänken måste ha större överföringskapacitet så att kommunikationshastigheten bibehålls. Därför måste i varje individuellt fall avgöras hur stark kryptering som krävs och väga detta mot den tillgängliga överföringshastigheten.

7 Referenser

Litteratur
Naganana Doraswamy and Dan Harkins, *IPSec The new Security Standard for the Internet, Intranets, and Virtual Private Networks*

Dokument och rapporter
RFC 2409, IKE
RFC 2411, IPSec RoadMap
RFC 3947, IKE NAT-T
RFC 4301, IPSec
RFC 4303, ESP
RFC 4302, AH
RFC 1034, DNS
RFC 1035, DNS

Cryptograpy in Theory and Practice: The Case of Encryption in IPsec, *Kenneth G. Paterson and Arnold K.l Yau, Information Security Gruop, Royal Rolloway, University of London.*

Attacking the IPsec Standardsin Encryption-only Configurations, *Jean Paul Degabriele and Kenneth G. Paterson, Hewlett-Packard Laboratorys Bristal and Royal Holloway University of London.*

Password Security: A Case History, Communications of the ACM, Vol.22, No.11, November, 1979, pp.594-597 *R. Morris and K. Thompson*

Manualer till DR-250 router

E-Designer help

Internet
http://www.nta-monitor.com/posts/2005/01/VPN-Flaws-Whitepaper.pdf

http://www.unixwiz.net/techtips/iguide-ipsec.html ,2008-07-16

http://www2.rad.com/networks/2005/adsl/main.htm ,2008-07-16

Övrigt
Vindmätare på Älfsborgsbron, Sting AB, Kontakt Dennis Karlsson

8 Appendix A

8.1 Exploitkod

Exempel på IIS exploitkod som fångats upp i loggfiler.

8.1.1 Exempel 1

```
SEARCH /\x90\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9
/.../
(2128 upprepningar av \x90)
/.../
\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\xc9\x90\x90\x90\x90\x90\x90\x90
/.../ 5133 upprepningar av \xc90
```

8.1.2 Exempel 2

```
GET /NULL.IDA?CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC%u0aeb%ub8
/.../
1513 varierande \????
/.../
xfa\xcd\xae\xed\xb0cmd.exe$ HTTP/1.1
```