FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Beveiliging van netwerken en computers

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 10 november 2018

UNIVERSITEIT
GENT

Inhoudsopgave

Hoofdstuk 1

Basisconcepten

Computerbeveiliging

Hoofdstuk 2

Netwerk en communicatiebeveiliging

Voorlopig hebben we enkel de basisconcepten gezien zoals: symmetrische encryptie, asymmetrische encryptie, hashfuncties en message authentication codes. Dit hoofdstuk zal deze concepten toepassen om beveiligingsprotocollen te ontwikkelen voor netwerken.

2.1 SSH

SSH (Secure Shell) is een **applicatielaagprotocol**. Ondanks deze naamgeving bevat SSH ook een **transportlaagprotocol**, met als bedoeling een veilige connectie te maken tussen andere OSI applicatielaagprotocollen (bv HTTP) en de werkelijke OSI transportlaag. SSH can draaien op zowel workstations als routers en switches. Routers en switches volgen de OSI laag namelijk niet strict, zodat zij ook een applicatielaag hebben. Vroeger werd het programma **telnet** gebruikt om remote configuratie toe te passen. Deze manier van werken is **onveilig** aangezien verkeer tussen de local host en remote host niet geëncrypteerd wordt, zodat deze door eender wie kunnen bekeken worden. SSH verhelpt dit probleem door de informatie te encrypteren. Vrijwel alle UNIX en Linux distributies komen met een versie van SSH geinstalleerd. SSH laat onder andere toe om:

- een veilige remote verbinding op te zetten vanuit eender welke local host naar eender welke remote host. Het protocol maakt gebruik van erkende algoritme voor zowel encryptie (sleutels van ten minste 128 bits), data-integriteit, sleuteluitwisselingen en public key management. Tijdens het leggen van een connectie worden algoritmen tussen de local host en remote host afgesproken, op basis van welke ze al dan niet ondersteunen,
- TCP te tunnelen via een SSH connectie,
- bestanden te verplaatsen, gebruik makend van bijhorende protocols zoals SCP (Secure Copy) of SFTP (SSH File Transfer Protocol),
- zowel X-sessions als poorten te forwarden.

2.1.1 Architectuur

SSH kan opgesplitst worden in drie elementen: Transport Layer Protocol, User Authentication Protocol en het Connection Protocol.

Transport Layer Protocol

Dit protocol heeft onder andere de verantwoordelijkheid om: authenticeren van servers, sleuteluitwisselingen en perfect forward privacy implementeren. Perfect forward privacy wil zeggen dat, indien een sleutel gecomprimeerd wordt tijdens een sessie, deze geen invloed kan hebben op de beveiliging van vorige sessies. Dit protocol loopt uiteraard op de transportlaag, en in de meeste gevallen zal dit TCP zijn. Vooraleer de cliënt kan connecteren moet hij de **public host key** van de server bezitten. Hiervoor kunnen er twee modellen gebruikt worden (cfr. RFC 4521) waarbij:

1. de cliënt een lokale databank bevat die de mapping beschrijft van elke hostnaam naar de correspondeerde public host key. Op deze manier is er geen centrale entiteit nodig, maar het beheer van deze databank kan, indien deze groot genoeg wordt, lastig zijn om te onderhouden.
2. de mapping bevestigd wordt door een **CA (Certification Authority)**. De cliënt kent in dit geval enkel de CA root key waarmee de geldigheid van elke host key kan nagaan die getekend zijn door deze CA. In dit geval moet de cliënt slechts één of enkele CA root keys bevatten.

Wanneer er een connectie kan gelegd worden van een cliënt tussen een server, is het eerste proces altijd het **onderhandelen van de algoritmen**. Deze stap zal algoritmen selecteren die compatibel zijn met zowel de cliënt als de server. Wanneer de cliënt een TCP connectie heeft met de server worden volgende pakketten verstuurd:

- **Identification string exchange.** Dit is een string dat zowel het protocolversie als de softwareversie van SSH bevat. Deze string wordt ten eerste van de cliënt naar de server verstuurd, waarop de server dan antwoord met zijn protocol en softwareversie. Indien hieruit blijkt dat de machines niet compatibel zijn met elkaar, wordt de connectie onderbroken (SSH2 is bv niet compatibel met SSH) en worden volgend stappen bijgevolg niet meer uitgevoerd.
- **Algorithm Negotiation.** In deze fase sturen zowel de server als de cliënt een SSH_MSG_KEXINIT bericht, die een lijst van alle ondersteunde algoritmen bevat, in volgorde van voorkeur. Elk type van algoritme zoals sleuteluitwisseling, encryptie, MAC algoritme en compressiealgoritme heeft zo zijn eigen lijst. Elk type moet dan ook een algoritme toegekend krijgen en wordt bepaald door het eerste algoritme dat de cliënt goed vind dat ook beschikbaar is op de server.
- **Key Exchange.** Indien de vorige fase goed gelukt is, start nu de sleuteluitwisseling. Voor de sleuteluitwisseling worden er momenteel slechts twee versie van **Diffie-Helman** ondersteund (cfr. RFC 2409). Het einde van de sleuteluitwisseling wordt gesignaleerd door een SSH_MSG_NEWKEYS pakket, met als gevolg dat zowel de cliënt als de server de gegenereerde sleutels mag gebruiken.
- **Service Request.** De laatste stap, dat eigenlijk het begin is van een volgend proces, wordt gesignaleerd door een SSH_MSH_SERVICE_REQUEST pakket. Dit pakket vraagt ofwel de start van het User Authentication Protocol of van het Connection Protocol. Alle verkeer

tussen server en cliënt wordt op dit moment getransporteerd als de payload van een SSH Transport Layer pakket, beveiligd met encryptie en een MAC.

Een SSH Transport Layer pakket heeft de volgende vorm, waarbij elementen met een δ-symbool geëncrypteerd en geauthenticeerd zijn en elementen met een Δ-symbool ook optioneel gecompresseerd kunnen zijn:

δ **Pakketlengte (4 bytes).** De lengte van het pakket in bytes, zonder de lengte van de pakketlengte zelf en het MAC veld in beschouwing te nemen.

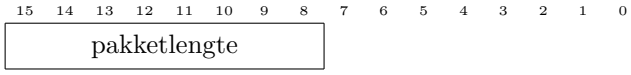
δ **Paddinglengte (1 byte).** Dit is de lengte van het random padding veld.

Δ **Payload**. De eigenlijke informatie van het pakket.

δ **Random padding.** Dit veld dient om cryptoanalyse moeilijk te maken. Kleine pakketten zijn op deze manier minder veel minder voorspelbaar en algoritmen, die vaak een kenmerkende vaste lengte hebben, worden ook moeilijker te achterhalen.

/ **MAC veld.** Dit veld wordt enkel gegenereerd indien dit zo in de onderhandeling besproken werd. Dit veld wordt berekent over het hele pakket en krijgt ook nog een bijhorend sequentienummer. Het sequentienummer start op 0, en wordt telkens met 1 geïncrementeerd voor elk pakket. Een aanvaller kan dit sequentienummer niet achterhalen aangezien een MAC een onomkeerbaar proces is.

packet



_ToDo: juiste afbeelding

Het **sleuteluitwisselingsproces** vraagt wat enige uitleg, er is echter nog niet nagegaan of deze sleutels op een **veilige** manier uitgewisseld worden. Op het moment van sleuteluitwisseling is er helemaal nog geen SSH connectie. Het uitwisselingsproces verloopt in twee fasen: eerst wordt er een gedeelde sleutel gegenereerd met Diffie-Helman, daarna wordt deze gedeelde sleutel gesigneerd met de publieke sleutel van de cliënt voor authenticatie. Sleutels kunnen ook heruitwisseld worden, hierbij gelden volgende regels:

- Mogelijk op elk moment, behalve tijdens het sleuteluitwisselingsproces.
- Kan aangevraagd worden door beide partijen.
- Sessie identificaties blijven ongewijzigd.
- Cryptografische algoritmen kunnen gewijzigd worden.
- Sessiesleutels worden vervangen.

Meestal worden sleutels vervangen na het behalen van een bepaalde quota zoals een tijdslimiet of het aantal totaal verstuurde bytes.

User Authentication Protocol

Dit protocol specifieerd **hoe** een cliënt zich moet authenticeren aan een server. Meerdere methoden zijn mogelijk waaronder de drie belangrijkste ervan: Public Key Authentication, Password Authentication en Host Based Authentication:

- **Public Key Authentication.** De implementatie van deze methode is afhankelijk van het gebruikte public-key algoritme. Een cliënt verstuurt berichten, dat gesigneerd is door de cliënt zijn private sleutel, naar de server, die de publieke sleutel van de cliënt bevat. De server gaat na of deze publieke sleutel nog geldig is en zoja, of dat de signatuur correct is.
- **Password Based Authentication.** De cliënt verstuurt zijn paswoord, dat geëncrypteerd wordt door het SSH Transport Layer Protocol naar de server, die nagaat of het paswoord correct is.
- **Host Based Authentication.** ToDo:

Connection Protocol

ToDo: vanaf slide 26

Hoofdstuk 3

Encryptiealgoritmen

3.1 Geschiedenis

Encryptiemethoden worden al eeuwenlang gebruikt om informatie onleesbaar te maken voor partijen die deze informatie niet mogen achterhalen. Deze inleiding bespreekt de basismethoden om encryptie toe te passen.

Zogenaamde **substitution ciphers** zijn de meest eenvoudigste vorm van encryptie. De originele versie, ook wel Caesar cipher genoemd, vervangt elke letter van het alfabet met een voorgedefinieerde shift in positie. Een shift van 3 zal het originele alfabet (in kleine letters) afbeelden op het verschoven alfabet (in grote letters): Het duidelijke nadeel is dat er slechts 25 verschillende encryp-

a b c d e f g h i j k l m n o p q r s t u v w x y z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

tiemogelijkheden zijn. Een verbeterde versie mapt elke letter met een willekeurige andere (nog niet gebruikte) letter: waardoor er nu 26! $\approx 4 \cdot 10^{26}$ mogelijkheden zijn. Deze manier is daarom niet meer

a b c d e f g h i j k l m n o p q r s t u v w x y z
D K V Q F I B J W P E S C X H T M Y A U O L R G Z N

secuur, aangezien de onderliggende frequentie van letters nog steeds dezelfde is. Een eenvoudige dencryptiemethode is om de relatieve frequenties van de ciphertekst te tellen, waardoor er statistisch kan achterhaald worden met welke gedecrypteerde letter elke geëncrypteerde letter overeenkomt.

Een encryptiealgoritme wil ook de relatieve frequenties verbergen. Een methode dat dit implementeert is de Vigenère Cipher. Deze bevat een sleutel $K = k_1 k_2 ... k_d$ waarbij k_i het i-de alfabet specifieert dat gebruikt moet worden. Na d letters start ment terug vanaf k_i.

Een laatste voorbeeld van een substitutiecipher is de Playfair cipher. Hier wordt er een $5x5$ matrix opgesteld, waarbij eerst het sleutelwoord ingevuld wordt, gevolgd door de niet gebruikte letters in alfabetische volgorde. Dit wordt uitgewerkt met de sleutel `MONARCHY` in Tabel 3.1. Een bericht wordt in paren van letters geëncrypteerd. Stel dat we het woord ballon willen encrypteren, de paren letters

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Tabel 3.1: Playfair cipher.

worden: ba lx lo nx. Er wordt een x toegevoegd indien er twee dezelfde letters achter elkaar komen, en als het eindpaar uit slechts één letter bestaat. Er kunnen zich drie gevallen voordoen bij elk paar:

1. Als beide letters in dezelfde rij voorkomen, wordt elke letter vervangen door de letter die er rechts van ligt: (on → na).
2. Als beide letters in dezelfde kolom voorkomen, wordt elke letter vervangen door de letter die er onder van ligt: (ba → ib).
3. Anders wordt elke letter vervangen door de letter van de kolom van de andere letter: (lx → su).

Er zijn $26 * 26 = 676$ mogelijke diagrammen die gemaakt kunnen worden.

Na de **substitution ciphers** zijn er ook **transposition ciphers**. Zulke ciphers gaan letters niet vervangen, maar gaan ze echter verplaatsen. Dit heeft natuurlijk als nadeel dat de relatieve frequentie van de letters behouden wordt. Twee eenvoudige voorbeelden zijn:

1. Keer elke letter om:
   `A SIMPLE EXAMPLE` → `ELPMAXE ELPMIS A`
2. Keer de woordvolgorde om, en elk woord keert ook de lettervolgorde om:
   `A SIMPLE EXAMPLE` → `A ELPMIS ELPMAXE`

Een iets beter voorbeeld is de Rail Fence cipher. Deze heeft als private sleutel het aantal rijen dat gebruikt wordt. Elke letter zal diagonaal geschreven worden, uitgewerkt in Tabel 3.2 op de zin `DEFEND THE EAST WALL`: Geëncrypteerd is dit dus `DNETLEEDHESWLXFTAAX`.

D				N				E				T				L		
	E		E		D		H		E		S		W		L		X	
		F				T				A				A				X

Tabel 3.2: Rail Fence cipher.

De laatste methode die besproken wordt is de columnar transposition cipher. Een bericht wordt in rijen geschreven over een bepaald aantal kolommen. Hierna worden de kolommen gesorteerd op basis van de geheime sleutel $K = k_i k_j ... k_n$. Stel $K = k_3 k_4 k_2 k_1 k_5 k_6 k_7$ en plaintext `ATTACK POSTPONED UNTIL TWO AM`: Op Tabel 3.3 wordt deze plaintext uitgeschreven in rijen en kolommen. De ciphertext wordt dan `TTNAAPTMTSUOAODWCOIXKNLYPETZ`.

A	T	T	A	C	K	P
O	S	T	P	O	N	E
D	U	N	T	I	L	T
W	O	A	M	X	X	X

Tabel 3.3: Rail Fence cipher.

Uiteindelijk kan men ook de combinatie maken van **substitution** en **transposition** ciphers, wat de basis vormt van moderne cryptografie.

3.2 Symmetrische algoritmen

Vooraleer er in detail kan ingegaan worden op symmetrische algoritmen, moet eerst de term **block cipher** besproken worden. Een block cipher wil zeggen dat informatie in blokken zullen verstuurd worden en kan best vergeleken worden met een substitutiecipher, maar dan toegepast op blokken. Een typische blokgrootte varieert van 8 tot 128 bytes en wordt door de meeste algoritmen gebruikt. De **Feistel Cipher encryptie** vormt de basis van moderne blockciphers en maakt gebruik van twee primitieve encryptieoperaties:

- Substitutie (de S-box)
- Permutatie (de P-box)

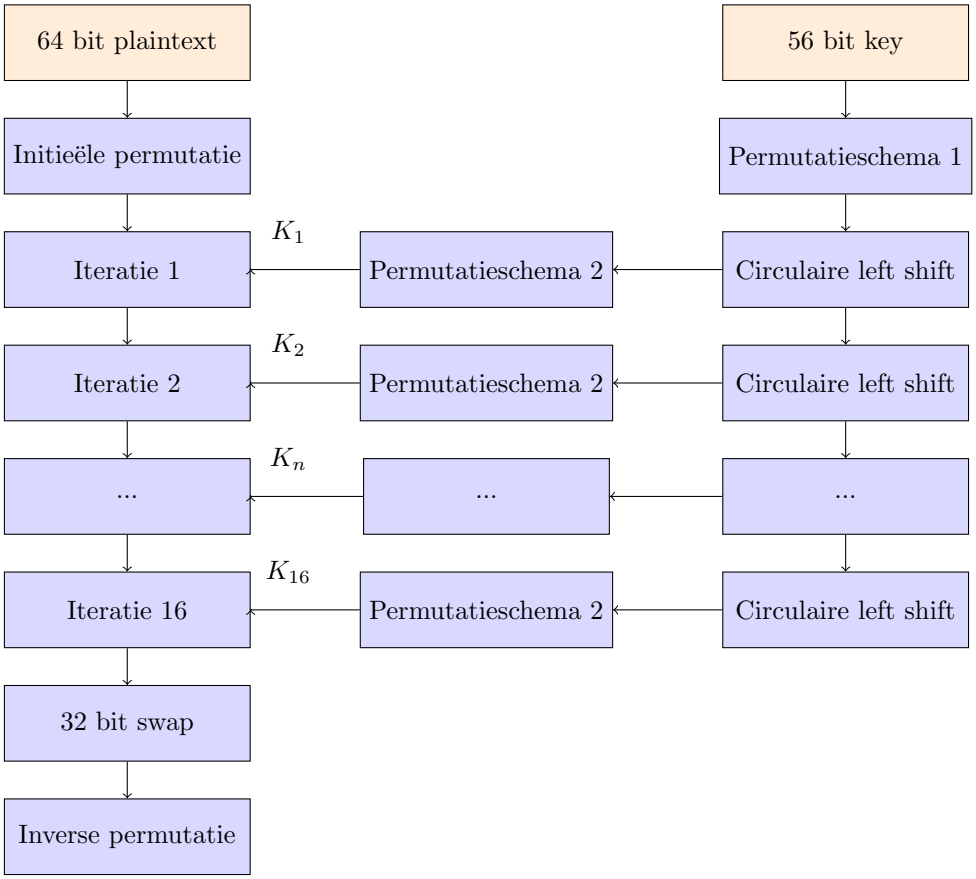
Het Feistel encryptieschema maakt gebruik van n ronden, een functie F en de XOR operator. ToDo: slide 36 uitleggen

3.2.1 DES & 3-DES

Het **DES (Data Encryption Standard)** maakt gebruik van het reeds vermelde Feistel schema waarbij de **blokgrootte 64 bits** en **sleutelgrootte 56 bits** bedraagt. Deze methode is redelijk traag, ondanks de kleine blokgrootte, en wordt door de kleine sleutelgrootte ook als onveilig beschouwd. Een 56 bit sleutel kan namelijk binnen de 10 uur gekraakt worden indien het aantal operatiers per **micro**seconde 10^6 bedraagt.

Het algoritme maakt gebruik van 16 ronden in het Feistel schema. De 56 bit sleutel wordt gebruikt om rondesleutels $K_1, K_2, ...K_{16}$ te bepalen, die elk 48 bits groot zijn. Voor de eerste ronde en na de laatste ronde wordt er een extra permutatie uitgevoerd, waarbij de laatste permutatie het inverse is van de eerste permutatie. Het resultaat van dit algoritme is één van de 2^{64} uitvoermogelijkheden van de 64 bits.

De 16 vereiste rondesleutels worden in het begin van het algoritme aangemaakt. Eerst en vooral wordt de 56 bit sleutel gepermuteerd, afhankelijk van het gekozen permutatieschema, op de afbeelding permutatieschema 1 genoemd. Een permutatieschema mapt een bit van de oorspronkelijke sleutel op een bit van de gepermuteerde sleutel, zodat er 2^{64} mogelijke schemas bestaan. De gepermuteerde sleutel wordt hierna beschouwd als twee resultaatsleutels C_n en D_n, met n het huidig


64 bit plaintext
56 bit key
Initieële permutatie
Permutatieschema 1
K_1
Iteratie 1
Permutatieschema 2
Circulaire left shift
K_2
Iteratie 2
Permutatieschema 2
Circulaire left shift
K_n
...
...
...
K_{16}
Iteratie 16
Permutatieschema 2
Circulaire left shift
32 bit swap
Inverse permutatie

rondenummer, van elk 28 bits groot. Beide resultaatsleutels worden onafhankelijk van elkaar verwerkt door een circulaire left shift van 1 of 2 bits, afhankelijk van de huidige ronde, dat eenvoudig door de conditionele operator kan beschreven worden: $n \,\%\, == 0 \;?\; 2 \;:\; 1$. Indien C_2 en D_2 de resultaatsleutels zijn van ronde 2, dan zijn C_3 en D_3 de resultaatsleutels van ronde 3 door respectievelijk C_2 en D_2 2 bits naar links te permuteren. Voor elke ronde wordt er een rondesleutel K_n gegenereerd, door een tweede permutatieschema, op de afbeelding permutatieschema 2 genoemd, toe te passen op $C_n D_n$, waarbij de 8 meest significante bits van C_n genegeert worden. Het eindresultaat zijn 16 verschillende rondesleutels, die zullen toegepast worden bij elke DES ronde.

Bij elke ronde wordt het 64-bit blok opgedeeld in twee 32-bit blokken L_n en R_n waarbij de 32 meest significante bits in L_n komen, en de 32 minst significante bits in R_n. Vooraleer dit proces plaatsvindt is er eerst een initiele permutatie van het hele 64-bit blok. Deze permutatie wordt uitgevoerd met behulp van een permutatieschema, zoals bij de sleutelgeneratie. Het gepermuteerde blok wordt hierna opgedeeld in de blokken L_n en R_n, en kan ronde 1 van start gaan. Bij elke iteratie wordt de functie F gebruikt, die twee blokken manipuleert: R_{n-1} en de sleutel van 48-bit groot, die op voorhand al gegenereerd werd. Deze functie F genereert een blok van 32 bits. L en R kan voor een volgende iteratie berekent worden als:

$$L_{n+1} = R_n$$

$$R_{n+1} = L_n + F(R_n, K_{n+1})$$

Hoe werkt de functie F? Eerst moet het blok R_n uitgebreidt worden via een selectietabel. Het gebruik van deze selectietabel wordt hier voorgesteld als de functie E. De funtie E heeft als input een blok van 32-bit en zal als output een blok van 48-bit genereren.

3.2.2 Andere

Hoofdstuk 4

Software- en systeembeveiliging