Beveiliging van het SNMP Security Model

Bart Lutgens, Erik Maessen, Jeffrey Lemein

27 juni 2008

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

1 Inleiding

1.1 Introductie

Veel netwerkapparaten ondersteunen het gebruik van het SNMP protocol. SNMP definieert een client/server-relatie. Het programma van de client (network manager) maakt verbinding met het programma van de server (SNMP agent) op een ander netwerkapparaat. De agent stuurt vervolgens zijn status met eventueel extra informatie terug naar de manager.

Er zijn tot op de dag van vandaag drie versies van SNMP uitgebracht. De eerste twee versies boden geen enkele ondersteuning voor beveiliging. De derde versie kwam uiteindelijk met toepassingen van beveiliging. Dit laatste protocol, SNMP versie 3, gaan we dus nader onderzoeken op enkele beveiligingspunten.

1.2 Wanneer noemen we iets beveiligd?

Om over de verschillende aspecten van beveiliging te praten heeft [STALL] de beveiliging ingedeeld in vier gebieden

- *Confidentiality*, vereist dat de data in een computersysteem alleen gelezen kan worden door geauthoriseerde partijen,
- *Integrity*, vereist dat de data in een systeem alleen gewijzigd kan worden door geauthoriseerde partijen,
- *Availability*, vereist dat data in een computersysteem beschikbaar is voor geauthoriseerde partijen, helaas kan een netwerkprotocol niks doen tegen hardwarevernieling, sabotage, etcetera.
- *Authenticity*, vereist dat een computersysteem in staat moet zijn een gebruiker te identi?ceren.

1.3 De beloften van het SNMP-protocol

SNMP versie 3 biedt voor het eerst beveiligingsvoorzieningen. In dit onderzoek zullen we het User-Based Security Model oderzoeken of de voorschriften die het protocol stelt ook inderdaad nageleefd worden. [RFC3414] stelt dat de volgende punten ondersteund moeten worden:

- *Data Integrity*,
- *Data Origin Authentication*,
- Bescherming tegen *message delay* of *replay* (waarbij de vertraging in tijd hoger ligt dan via normale operaties bereikt kan worden),
- Bescherming tegen *disclosure* van de *message payload*.

1.4 Onderzoeksvraag

Biedt het *User-Based Security Model* voldoende bescherming tegen veelvoorkomende kraakpogingen op de gebieden: confidentiality, integrity en authenticity? Met "voldoende" bedoelen we dat een verzilverde kraakpoging gemiddeld minstens een paar maanden duurt. De aanvallen die we zullen onderzoeken zijn:

- Brute-force attack op sleutels
- Replaying Messages
- Modification of Information
- Masquerade
- Disclosure
- Message Stream Modification
- Denial of Service
- Traffic Analysis

We zullen elke aanvalspoging een cijfer geven: een 1, 2 of 3:

1. Er is geen ondersteuning voor deze beveiliging of de beveiliging is ronduit slecht. Na enkele pogingen is de kans vrij groot (>50%) dat er gevoelige informatie data is opgehaald.
2. Niet geweldig beveiligd, na enkele dagen inspanning kan het systeem al gekraakt zijn.
3. Goede beveiliging, praktisch kan het systeem (vrijwel) niet gekraakt worden.

Als het gemiddelde cijfer groter is dan een 2,00 dan vinden we dat het protocol voldoende bescherming biedt.

2 Methode

Om erachter te komen of SNMP tegen de door ons opgestelde aanvallen beschermd is, is eerst wat vooronderzoek vereist over het user-based security model van het protocol. Dit bestaat bij SNMP vooral uit twee grote onderdelen die de authenticiteit en de privacy van berichten beschermd.

2.1 Authenticatie Protocol

Voor de authenticatie gebruikt SNMP het HMAC-MD5-96 Authentication Protocol. Dit protocol gebruikt de MD5 hash-functie. MD5 gebruikt het zogenaamde message digest algorithm. Dit algoritme zorgt voor:

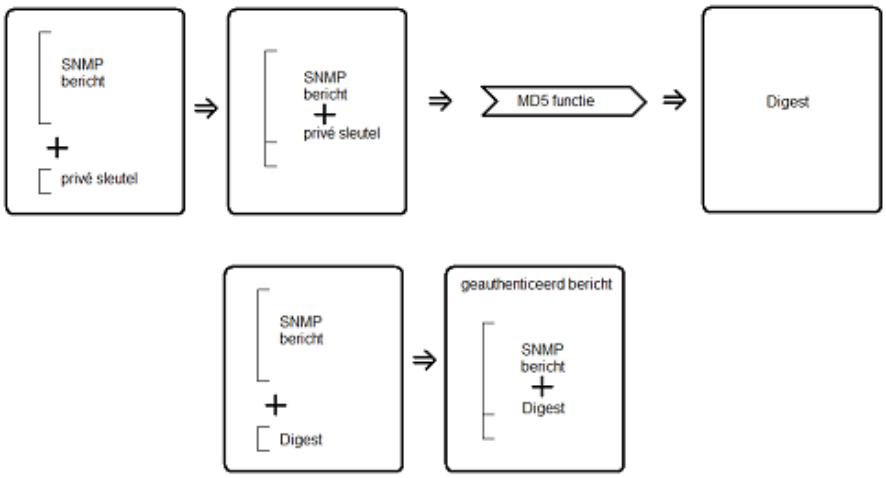
- Verificatie van de integriteit van een ontvangen bericht. (het ontvangen bestand is hetzelfde als het verstuurde bestand)
- Verificatie van de user die het bericht heeft verstuurd.

2.1.1 De globale werking van het Authenticatie protocol

Het algoritme gaat kort gezegd als volgt te werk:

1. Het te versturen bericht word naar de SNMP engine gestuurd. De engine gaat het bericht authenticeren voordat het bericht daadwerkelijk verstuurd wordt naar de ontvanger.
2. Een geheime waarde(de priv authenticatie sleutel) wordt aan het SNMP bericht gevoegd.
3. MD5 berekent een 128-bits digest over een bepaald gedeelte van het bericht. Een digest is een reeks nummers, berekent van een oorspronkelijk bericht. In dit geval dus van het SNMP bericht in combinatie met de geheime waarde. Hierdoor ontstaat een unieke digest.
4. De eerste 96 bits van de berekende digest worden samen met het bericht verstuurd. De geheime waarde niet.
5. Zodra de SNMP engine van de ontvanger het bericht ontvangt gaat hij de gebruiker en de integriteit van het bericht verifiren. Dit kan aan de hand van de digest. De engine berekent op dezelfde manier als de andere engine de digest en controleert of deze digest overeenstemt met de ontvangen digest. De geheime waarde van elke gebruiker is bekend bij de engine. Het geauthenticeerde bericht wordt dan naar de ontvanger doorgestuurd.

Een bericht dat dit algoritme gebruikt, heeft een <msgAuthenticationParameters> veld, als een deel van de <msgSecurityParameters>. In het <msgAthenticationParameters> veld worden de eerste 12 octetten (96 bits) van de digest meegestuurd. In figuur 1 is in een model schematisch weergegeven hoe een bericht geauthenticeerd word.



Figuur 1: Simple SNMP authentication model

2.1.2 Elementen van het Digest Authentication Protocol

User Een user heeft een <userName> en een <authKey>. De <userName> is een string en representeert de naam van de user. De <authKey> is altijd 16 octetten lang en wordt gebruikt bij het berekenen van de digest. Voor elke user die geauthenticeerd moet worden door een bepaalde SNMP engine, moet de desbetreffende engine kennis hebben van de user. Ook bij communicatie tussen over een user tussen twee engines moeten beide engines kennis hebben van die user.

msgAuthoritativeEngineID De msgAuthoritativeEngineID-waarde binnen een bericht specificeert de correcte SNMP engine voor dat specifieke bericht. De <authKey> van een user is normaal gesproken verschillend bij elke verschillende engine. Dus de engineID kiest de goede <authKey> voor het authenticatie proces.

Genereren van een uitgaand SNMP bericht *statusInformation = authenticateOutgoingMsg(IN authKey, IN wholeMsg, OUT authenticatedWholeMsg)*

De input bij het generen zijn dus de <wholeMsg> en de <authKey>. De <wholeMsg> is het gehele bericht dat geauthenticeerd moet worden en de <authKey> is de priv sleutel van de gebruiker. De output bij het genereren van een bericht is de <authenticatedWholeMsg>. Dit is het gehele geauthenticeerde bericht. De <statusInformation> is een indicatie of aan dit athenticatie proces is voldaan.

Proces bij een inkomend SNMP bericht *statusInformation = authenticateIncomingMsg(IN authKey, IN authParameters, IN wholeMsg, OUT authenticatedWholeMsg)*

Bij het proces van een inkomend SNMP bericht zijn er drie input attributen; <authKey>, <authParameters> en <wholeMsg>. De <authKey> is de priv sleutel van de user. De <authParameters> komen binnen via het bericht. De <wholeMsg> is het hele bericht dan geauthenticeerd moet worden. De output is <authenticatedWholeMsg> en representeerd het ingekomen bericht dat geauthenticeerd is. Wederom is <statusInformation> een indicatie of het proces succesvol is verlopen.

2.1.3 Procedures

Het volgende stuk beschrijft de procedures van de SNMP engine als het een uitgaand of inkomend bericht moet authenticeren.

Uitgaand bericht

1. Het <msgAuthenticationParameters> veld wordt zo aangemaakt, dat er 12 octetten in gezet kunnen worden.
2. Uit de priv sleutel worden twee sleutels afgeleid: K1 en K2.
   - De 16 octetten lange sleutel uitbreiden tot 64 octetten door er 48 nul-octetten achter te zetten en dit opslaan als <extendedAuthKey>.
   - Verkrijg IPAD door octet 0x36 64 keer te kopiren.
   - Verkrijg K1 door <extendedAuthKey> te XORen met IPAD.
   - Verkrijg OPAD door octet 0x5c 64 keer te kopiren.
   - Verkrijg K2 door <extendedAuthKey> te XORen met OPAD.
3. Voeg K1 toe aan <wholeMsg> en bereken de digest hiervan.
4. Voeg K2 toe aan het vorige resultaat en bereken de digest hiervan. De eerste 12 octetten van dit resultaat is de uiteindelijke digest ook wel MAC(Message Authentication Code) genoemd.
5. Zet de MAC in het <msgAuthenticationParameters> veld.
6. De <authenticatedWholeMsg> wordt dan teruggestuurd naar de user samen met de ¡statusInformation¿ die indiceert dat het succesvol is verlopen.

Inkomend bericht

1. Als de ontvangen digest in het <msgAuthenticationParameters> veld, niet 12 octetten lang is, wordt er een error indicatie teruggestuurd.
2. De ontvangen digest(MAC) wordt opgeslagen.
3. In het <msgAuthenticationParameters> veld wordt de digest vervangen door 12 nul-octetten.
4. Uit de priv sleutel worden twee sleutels afgeleid: K1 en K2.
   - De 16 octetten lange sleutel uitbreiden tot 64 octetten door er 48 nul-octetten achter te zetten en dit opslaan als <extendedAuthKey>.
   - Verkrijg IPAD door octet 0x36 64 keer te kopiren.
   - Verkrijg K1 door <extendedAuthKey> te XORen met IPAD.
   - Verkrijg OPAD door octet 0x5c 64 keer te kopiren.
   - Verkrijg K2 door <extendedAuthKey> te XORen met OPAD.
5. De MAC wordt berekend over de <wholeMsg>
   - Voeg K1 toe aan <wholeMsg> en bereken de digest hiervan.
   - Voeg K2 toe aan het vorige resultaat en bereken de digest hiervan.
   - De eerste 12 octetten van het vorige resultaat is de MAC
   - De MAC opgeslagen in stap 2 wordt in het <msgAuthenticationParameters> veld gezet.
6. De nieuw berekende MAC wordt vergeleken met de MAC opgeslagen in stap 2. Als ze niet overeenkomen wordt er een error indicatie teruggestuurd.
7. De <authenticatedWholeMsg> wordt dan teruggestuurd naar de user samen met de <statusInformation> die indiceert dat het succesvol is verlopen.

2.1.4 HMAC-SHA-96 protocol

Het HMAC-SHA-96 protocol wordt ook door SNMP gebruikt om berichten te authenticeren. Dit protocol verschilt op slechts twee plaatsen van het MD5 protocol. SHA gebruikt namelijk een priv sleutel van 20 octetten in plaats van de 16 octetten van MD5. Het tweede verschil is de manier waarop de digest wordt berekend. MD5 gebruikt hiervoor MD5 hash-functie en SHA de SHA hash-functie.

2.1.5 Conclusie

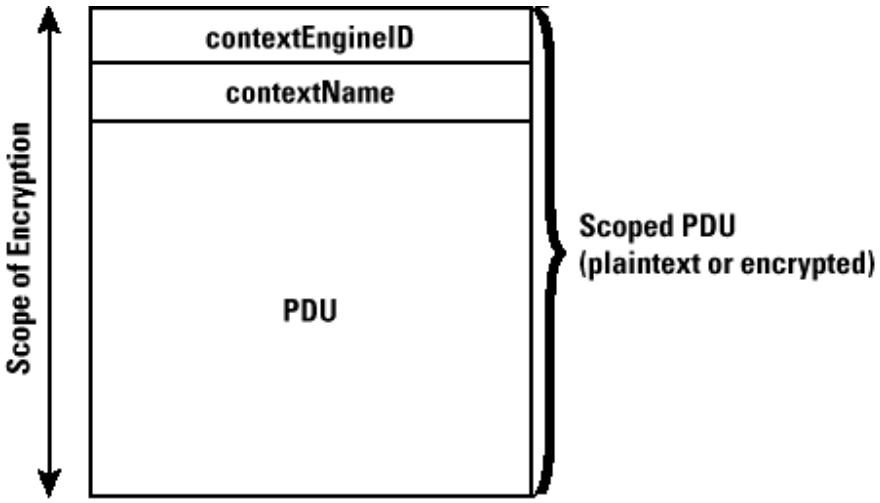
Door het authenticeren van het bericht worden de *Integrity* en *Authenticity* gewaarborgt. Omdat elke gebruiker een eigen sleutel heeft die bekend is bij de engine, kan de engine aan de hand van de digest controleren of de gebruiker wel echt geauthenticeerd is. Als een kwaadwillende zich wil voordoen als geauthenticeerde gebruiker moet hij/zij deze sleutel kraken. Deze sleutel betreft een rij van 64 bits en is dus erg moeilijk om te kraken. Met brute force is het mogelijk, mits de kwaadwillende daar de apparatuur en tijd voor heeft. Verder is het vrijwel onmogelijk om de authenticiteit te kraken omdat er gebruikt wordt gemaakt van hash functies en allerlei handelingen om enig patroon in de digest te voorkomen. Hierdoor kan de niet geauthenticeerde gebruiker geen juiste digest meesturen en zich dus niet voordoen als geauthenticeerde gebruiker.

2.2 Privacy Protocol

Om (data) confidentiality tegemoet te komen stelt SNMP dat een gedeelte van de te verzenden berichten via het netwerk gencodeerd is. Dit gedeelte is bijvoorbeeld de request van een gebruiker aan een engine, en staat in de scopedPDU. Daarom moet deze data met grote zorg worden behandeld zodat kwaadwillende mensen of systemen de verstuurde data niet kunnen achterhalen. (privacy). Dit wordt gedaan door de scopedPDU te coderen. Op deze manier moeten alleen geauthoriseerde partijen de berichten kunnen lezen. In het User-Based Security Model wordt dit verwezenlijkt met een zogenaamd privacy protocol, ook wel bekend onder de naam: CBC-DES Symmetric Encryption Protocol. De scopedPDU bestaat uit drie attributen (zie figuur x):

- <contextEngineID>
- <contextName>
- <PDU>

Hier zullen we verder niet op ingaan, omdat dit irrelevant is voor wat we aan het onderzoeken zijn.



Figuur 2: ScopedPDU

In de figuur 2 zie je dat een scoped PDU gecodeerd of platte tekst mag zijn. SNMP geeft u namelijk de keuze of u berichten wilt verzenden zonder of met gebruik van privacy. Zo kun je de tijd die het extra in beslag neemt op privacy-controleroutines verkleinen. Hier kun je voor kiezen als er informatie is die niet

relevant is (zoals een papier-op bericht van de printer), of als je het netwerk veilig waant. In ons onderzoek nemen we uiteraard aan dat de beschikbare beveiliging van SNMP versie 3 volledig ingeschakeld is.

2.2.1 De globale werking van het privacy protocol

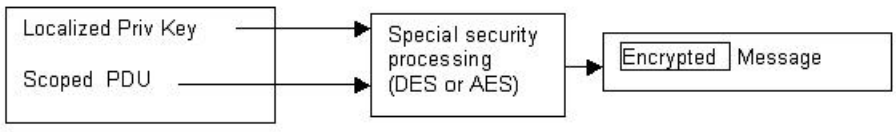
De netwerkbeheerder geeft elke gebruiker in het netwerk een eigen wachtwoord. Wanneer een gebruiker een bericht wil versturen naar een engine, voert het drie gegevens in:

- De gebruikersnaam,
- Het authenticatie-wachtwoord,
- Het privacy-wachtwoord.

In hoofdstuk x hebben we uitgelegd hoe het authenticatie-wachtwoord gebruikt wordt voor authenticatie. Waar het nu om draait is de privacy. Net zoals bij authenticatie wordt er een "lokale sleutel" gegenereerd (in dit geval de initialisatievector (IV)) en via een hash-functie $ met als argumenten de SALT (volgt zo) en het privacy wachtwoord (<privKey>):

$$Lokalesleutel(IV) = \$(< privKey >, SALT); \quad (1)$$

Met behulp van deze lokale sleutel kunnen we de scopedPDU coderen. Om hiervoor te zorgen moet er een encryptiemethode toegepast worden die zeer lastig te kraken is.



Figuur 3: De Encryptie van de ScopedPDU

In SNMP versie 3 wordt de scopedPDU gecodeerd via het *CBC-DES Symmetric Encryption Protocol*. Het CBC-DES Symmetric Encryption protocol gebruikt symmetrische cryptografie waarbij dezelfde sleutel wordt gebruikt voor zowel het coderen van informatie als voor het decoderen. Het voordeel is dat het coderen en decoderen veel sneller gaat dan asymmetrische coderingen/decoderingen. Daarom is deze vorm van encryptie ook gekozen voor het SNMP-protocol. De encryptiemethode maakt gebruik van het *Cipher Block Chaining principe*, dat hieronder nader wordt uitgelegd.

2.2.2 Cypher-Block Chaining Principe (CBC)

Cipher block chaining (CBC) is een coderingsmethode waarbij een reeks van bits (in ons geval 8 octets, oftewel 64 bits) gecodeerd wordt als een enkel blok met een cijfersleutel toegepast op het gehele blok. CBC maakt hiervoor gebruik van een initialisatievector (IV) van een bepaalde lengte. Hoe de initialisatievector berekent wordt, wordt zodadelijk uitgelegd. Een van de karakteristieken aan dit principe is dat de decodering van een gecodeerd blok afhangt van alle voorgaande gecodeerde blokken.

Encryptie van een bericht in SNMP De data die verstuurd moet worden is dus opgebouwd uit een reeks van 8 octets. Als de octet niet volgemaakt kan worden, dan wordt de serie van 8 aangevuld. De initialisatievector wordt gebruikt om de keten aan de gang te krijgen:

1. De initialisatievector wordt geXORed met de eerste 8 octets,
2. Het resultaat uit 1 is de sleutel die geXORed wordt met het tweede blok van 8 octets.
3. Enzovoorts totdat alle blokken zijn gecodeerd.

Als we XOR als een functie zien van twee argumenten $a1$ en $a2$ (ieder 8 octets), met als resultaat de exclusive-OR van $a1$ en $a2$ dan kunnen we de encryptie als volgt in rij-notatie noteren:

$$C_0 = XOR(P_0, IV); \tag{2}$$

$$C_{n+1} = XOR(C_n, P_{n+1}); \tag{3}$$

IV De initialisatievector.

P_x De verzameling platte tekst die de data voorstelt die gecodeerd moet worden. Het subscript geeft het blok (van 8 octets) aan, waarbij x = 0 gelijk is aan het eerste blok in de te coderen data.

C_x De verzameling gecodeerde data die klaar is om (volgens het privacy-principe) verzonden te worden. Het subscript geeft het blok (van 8 octets) aan, waarbij x = 0 gelijk is aan het eerste blok in de gecodeerde data.

Een enkele fout in een bit in een van de voorgaande blokken, heeft tot gevolg dat alle volgende blokken verkeerd ontcijferd worden. Daarnaast raakt een bericht (serie van blokken van 8 octets gecodeerde informatie) corrupt als de volgorde van de blokken in het bericht veranderd wordt.

Decryptie van een bericht in SNMP Bij het decoderen wordt eerst gecontroleerd of de versleutelde data een veelvoud is van 8. Is dit niet het geval, dan wordt het decodeerproces afgebroken. Vervolgens wordt hetzelfde principe als bij encryptie toegepast:

1. De initialisatievector wordt geXORed met het eerste blok van 8 octets, je hebt dan het eerste blok gedecodeerd.
2. Voor alle volgende blokken wordt het vorige gecodeerde blok gebruikt.
3. Enzovoorts totdat alle blokken zijn gedecodeerd.

Als we XOR als een functie zien van twee argumenten a1 en a2 (ieder 8 octets), met als resultaat de exclusive-OR van a1 en a2, dan kunnen we de encryptie als volgt noteren:

$$P_0 = XOR(C_0, IV); \tag{4}$$

$$P_{n+1} = XOR(C_n, C_{n+1}); \tag{5}$$

IV de initialisatievector.

P_x De verzameling platte tekst die de data voorstelt die gedecodeerd is. Het subscript geeft het blok (van 8 octets) aan, waarbij x = 0 gelijk is aan het eerste blok in de gedecodeerde data.

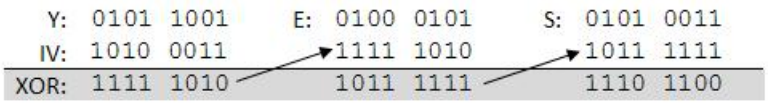
C_x De verzameling gecodeerde data die gedecodeerd dient te worden. Het subscript geeft het blok (van 8 octets) aan, waarbij x = 0 gelijk is aan het eerste blok in de gecodeerde data.

Voorbeeld encodering Om het principe te tonen, gebruiken wij per blok 8 bits en stellen we de letters voor in hun ASCII-waarde. Stel je het volgende stuk tekst voor:

Bericht:	`Y E S`
Binair:	`0101 1001 0100 0101 0101 0011`

Tabel 1: Originele tekst

Stel je voor dat de initalisatievector (IV) gelijk is aan: $\{1, 0, 1, 0, 0, 0, 1, 1\}$ De encodering ziet er als volgt uit:



Figuur 4: Een encryptievoorbeeld volgens CBC

1e rij: Originele bericht

2e rij: Sleutels gebruikt voor de encodering.

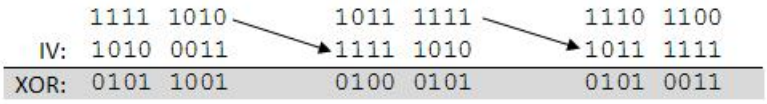
3e rij: Gecodeerde bericht (Resultaat van een XOR van de bovenste 2 rijen)

Het gecodeerde bericht luidt dus:

Message:	•⌉∞
Binary:	1111 1010 1011 1111 1110 1100

Tabel 2: Gecodeerd bericht

Voorbeeld decodering We zullen het vorige gecodeerde bericht decoderen:
De initalisatievector (IV) was gelijk aan: $\{1, 0, 1, 0, 0, 0, 1, 1\}$ De encodering ziet er als volgt uit:



Figuur 5: Een decryptievoorbeeld volgens CBC

1e rij Versleutelde bericht

2e rij Sleutels gebruikt voor de decodering

3e rij Gedecodeerde bericht (Resultaat van een XOR van de bovenste 2 rijen)

Door in de ASCII-tabel te kijken hebben, we inderdaad weer het juiste bericht teruggevonden:

Bericht:	`Y E S`
Binair:	`0101 1001 0100 0101 0101 0011`

Tabel 3: Gedecodeerd bericht

De berekening van de initialisatievector (IV) Elk blok dat verzonden wordt, bestaat uit 8 octets. Het is daarom vanzelfsprekend dat de IV ook uit 8 octets bestaat, wil je er een XOR op toepassen. Zoals bekend, bevat elke gebruiker het attribuut privacy key (<privKey>). Dit attribuut bestaat uit 16 octets, waarvan de laatste 8 octets als pre-IV wordt gebruikt.

De initialisatievector moet bij elke verzending van gecodeerde data anders zijn, zodat het kraken van de gecodeerde data erg moeilijk wordt. Per verzonden pakket moet er daarom een unieke waarde bestaan die hiervoor zorgt, afgekort tot SALT.

SALT bevat de concatenatie (afgekort tot *concat(x, y)*) van <snmpEngineBoots> van de engine waar het bericht heen gestuurd dient te worden en een lokale 32-bitwaarde die de versturende SNMP engine willekeurig genereert bij het opstarten.

Tenslotte wordt er een hashfunctie # op SALT losgelaten die hetzelfde is als de hashfunctie voor het authenticatieprotocol. Het resultaat wordt afgekapt tot 8 octets.

De generatie van de initialisatievector ziet er dus als volgt uit:

Pre-IV: Laatste 8 octets van <privKey> van de gebruiker.

SALT: #(concat(<snmpEngineBoots>, lokale 32-bitwaarde));

IV: XOR(Pre-IV, SALT);

Het terugberekenen van de initialisatievector (IV) Elk binnenkomend bericht bevat een veld *message privacy parameters* (<msgPrivacyParameters>). Uit dit veld wordt de SALT gehaald. De IV kan zoals we net hebben uitgelegd, berekend worden door een XOR toe te passen op de SALT en de privacy key.

In het opgestuurde bericht bevindt zich bovendien de *message username*, die aangeeft welke gebruiker het bericht heeft verstuurd. En omdat elke SNMP engine de privacy key kent van zijn gebruikers, kan de gebruikersnaam gebruikt worden om privacy key te vinden die gebruikt kan worden om de IV te berekenen.

2.2.3 Conclusie

Om *data confidentiality* te waarborgen heeft het User-Based Security Model dus aangeraden om de volgende attributen te gebruiken:

- <privKey>,
- <snmpEngineBoots>,
- Willekeurige lokale 32-bitwaarde.

Bij het verzenden van een bericht moet de gebruiker ter verificatie zijn gebruikersnaam in combinatie met zijn privacy key invoeren. De privacy key wordt vervolgens indirect gecodeerd meeverstuurd. Deze indirectheid is de speerpunt van de data confidentiality.

Bij de kraakpogingen gaan we er vanuit dat het bericht succesvol geauthenticeerd is (dus de kraker heeft de authenticatie omzeild/afgewend/etcetera). Wat nodig is om de scoped PDU te ontcijferen is enkel de privacy key (<privKey>). Daarnaast beperken we ons hier alleen tot de data confidentiality zaken, dus message replay en andere zaken zijn (nu nog niet) niet aan de orde.

Voor krakers is dus alleen de SALT en de gebruikersnaam bekend, maar omdat de SALT die verstuurd wordt elke keer anders is, kan de kwaadwillende geen patroon vinden in de combinatie gebruikersnaam met wachtwoord. Patroonherkenning tussen de combinatie gebruikersnaam en privacy key is dus (vrijwel) onmogelijk. Daarnaast staan de hashfuncties erom bekend wel hn, maar niet terug te kunnen coderen/decoderen. Het is "vrijwel onmogelijk" omdat er altijd het risico bestaat dat de willekerugie lokale 32-bitwaarde achterhaald wordt door het precieze tijdsmoment te bepalen waarop de waarde is berekend.

Een andere manier om de privacy key te achterhalen is door systematisch te proberen, een woordenlijst van veelgebruikte keys afgaan, etcetera. Het wachtwoord dat geraden moet worden, bestaat uit 8 octets (64 bits) en wachtwoorden van deze omvang kunnen met de huidige techniek succesvol gekraakt worden.

Waar zich wel een probleem in kan voordoen is dat de privacy key achterhaald kan worden als de kraker toegang heeft tot de engine. Daar staan immers de wachtwoorden opgeslagen. Dit zal hoogstwaarschijnlijk in gecodeerde vorm zijn middels een hashfunctie, maar in dit geval is het dus wel mogelijk patronen te herkennen in de combinatie gebruikersnaam met privacy key.

Voordelen

- Patroonherkenning tussen gebruikersnaam en wachtwoord (vrijwel) onmogelijk.

Nadelen

- Brute-force attacks zijn geregeld succesvol tegen sleutels met een lengte van 64 bits.
- Alle voordelen gelden niet als de kraker toegang heeft tot de opgeslagen attributen op de engine.

2.3 Aanvallen

2.3.1 Brute-force attack op sleutels

Bij een brute-force aanval tegen een sleutel wordt elke mogelijke optie uitgeprobeerd, net zolang tot de ingevoerde waarde overeenkomt met die van de sleutel. Deze methode is zeer inefficint, door de zeer lange duur maar wel 100De formule voor de schatting van de maximumtijd om een wachtwoord te vinden (gebaseerd op drie miljoen wachtwoorden per seconden) is:

$$seconden = \frac{karakters^{posities}}{3000000} \tag{6}$$

2.3.2 Replaying Messages

Bij een message replay attack wordt een valide message onderschept door een derde partij. Deze herhaalt op een later tijdstip dit bericht terwijl hij zich voor doet als degene waarvan hij het bericht heeft onderschept. Stel Client A wil zichzelf authoriseren. Agent B vraagt dan om het wachtwoord. A stuurt dit wachtwoord, dat met behulp van eerder genoemde hashfunctie wordt gecodeerd, op naar B. Ondertussen is de niet-geauthoriseerde Client C mee aan het luisteren en onderschept dit bericht. Als de verbinding tussen A en B is verbroken verbind C zich met Agent B terwijl hij zich voordoet als A. Hij stuurt nu het onderschepte bericht met het wachtwoord op naar B, die dit accepteerd. Client C is nu geauthoriseerd.

2.3.3 Modification of information

Modifcation of information is zoals vertaald het aanpassen van informatie. Dit kan gevaarlijk zijn als de kwaadwillige het bericht zo veranderd dat bepaalde niet geauthoriseerde management operaties veranderen. Het veranderen van zo'n bericht gebeurd in transit. Dat wil zeggen dat het bericht onderweg aangepast wordt.

2.3.4 Masquerade

Masquerade is een aanval waarbij de indringer de identiteit aanneemt van een geauthoriseerde gebruiker om zo informatie op te vragen of te veranderen binnen het systeem.

2.3.5 Disclosure

Disclosure is het uitlekken van informatie tussen contacten tussen agents en managers.

2.3.6 Message Stream Modifacation

Message Stream Modifacation richt zich op deze berichten stroom en verg- root het aantal her-ordeningen, vertragingen en herhalingen zo, dat dit aantal groter

wordt dan ooit door de natuurlijke operaties zou kunnen. Hierdoor kun- nen niet-geauthoriseerde management operaties aangetast worden.

2.3.7 Denial of service

Een *denial of service* (DoS) vindt plaats als een systeem niet kan reageren op een vraag/verzoek van de agent. Zo zijn er vier soorten van DoS:

- het geven van ongeldige commando's, waardoor er een buffer volloopt.
- het consumeren van systeembronnen, zodat opdrachten niet uitgevoerd kunnen worden, omdat het systeem bezet is.
- het herconfigureren van systeembronnen, zodat opdrachten niet meer uitgevoerd kan worden.
- het fysiek vernietigen van een systeembron, bijvoorbeeld het verwijderen van een netwerkkabel.

2.3.8 Traffic analysis

Bij traffc analysis wordt een heel groot aantal berichten onderschept en bekeken om patronen te herkennen. Hoe meer berichten wordt bekeken hoe beter de informatie kan zijn. Het nadeel van traffic analysis is dat het toegepast kan worden op gecodeerde berichten. De tijd tussen 2 berichten kan ook bestudeerd worden met het zogenaame 'hidden Markov model'. De autheurs van dit model claimen dat het 50 keer sneller is dan brute-force.

3 Resultaten

We hebben per gestelde aanval onderzocht in hoeverre SNMP hiertegen beveiligd. We hebben dit gedaan door elke aanvalspoging een cijfer te geven: een 1, 2 of 3.

1. Er is geen ondersteuning voor deze beveiliging of de beveiliging is ronduit slecht. Na enkele pogingen is de kans vrij groot (>50%) dat er gevoelige informatie data is opgehaald.
2. Niet geweldig beveiligd, na enkele dagen inspanning kan het systeem al gekraakt zijn.
3. Goede beveiliging, praktisch kan het systeem (vrijwel) niet gekraakt worden.

Als het gemiddelde cijfer groter is dan een 2,00 dan vinden we dat het protocol voldoende bescherming biedt.

3.1 Brute-force attack op sleutels

Omdat het aantal posities al 64 is gaat het hier al om jaren om de juiste waarde te vinden. Toch is het al praktisch haalbaar om het te kraken. Er is eigenlijk maar 1 manier om brute-force tegen te gaan en dat is om het uitvoeren zolang te laten duren dat het voordeel ervan teniet wordt gedaan. Dit wil dus zeggen: Een zo groot mogelijke sleutel kiezen met zoveel mogelijk verschillende tekens. In het geval van SNMP wordt er gebruik gemaakt van een 64 bits sleutel met 2 karakters. Invullen van de formule voor de schatting van de maximum tijd geeft:

$$jaren = \frac{2^{64}}{3000000 * 3600 * 24 * 365} = 194980.81 jaren \tag{7}$$

Omdat dit niet binnen dagen te kraken is, geven wij brute-force attacks op sleutels het cijfer 3.

3.2 Replaying Messages

De maatregel die in SNMP wordt gebruikt heet Timestamping of Time Synchronization. In dit geval wordt er gebruik gemaakt van de tijd in de engine van de Agent. Elke niet-geauthoriseerde client moet een lokale variabele latestReceivedEngineTime bij houden. De geauthoriseerde Engine stuurt periodiek zien lokale tijd door aan de client samen met zijn MAC, Message Authentication Code, die door de client wordt gecontroleerd. De client slaat vervolgens deze tijd op in zijn lokale variabele. Als de client zich nu wil authoriseren moet hij ook deze variabele, gecodeerd met een hashfunctie, meesturen. De Agent controleert de latestReceivedEngineTime van de client tegen zijn lokale tijd en hij zal alleen berichten accepteren die binnen een redelijke marge vallen. Dit zal

meestal tot de volgende keer dat de Agent zijn lokale tijd verstuurd zijn. Dit kan uiteraard alleen maar werken als de Agent vaak genoeg zijn tijd verstuurd, anders kan binnen de marge het onderschepte bericht nog worden verstuurd. Het cijfer dat deze beveiliging tegen replaying messages krijgt is een 3.

3.3 Modification of Information

SNMP claimt dat het hier tegen beveiligd is, omdat het bericht gecodeerd is. Het bericht is zo gecodeerd dat enig patroon in verschillende berichten niet te vinden is, zodat de kwaadwillige geen 'nuttige' veranderingen kan aanbrengen. Hoogstens kan hij/zij het bericht hier en daar willekeurig aanpassen zodat het gedecodeerde bericht niets zinvols meer bevat, waardoor de ontvanger het bericht negeert of als niet geauthenticeerd beschouwt. Modifcation of information richt zich echter op het aanpassen van infor- matie zo, dat de infomatie iets van schade kan aanrichten. Hierbij wordt er van uitgegaan dat de kwaadwillige geen prive gegevens van de gebruikers bezit. SNMP is dus goed beveiligd tegen modifcation of information door het sterke encryptie protocol. We kennen het volgende punt toe: 3.

3.4 Masquerade

Bij het authenticatieprotocol zagen we dat er gebruik wordt gemaakt van een privacy key. Omdat de indringer deze niet weet, zal het in principe geen misbruik van de situatie kunnen maken. Echter is de versleutel- ing telkens hetzelfde voor dezelfde opdracht van dezelfde gebruiker. Hierdoor bestaat de kans dat er een patroon wordt herkend, alhoewel die kans bij een hash-functie relatief klein is. Het cijfer dat masquerade verdient, is een 2.

3.5 Disclosure

Het privacy protocol verzorgt deze eis, door de daadwerkelijke gevoelige data (commando's, requests, etcetera) per bericht anders te encoderen. Door de hashfunctie valt het bericht niet terug te decoderen, waardoor er geen patroon gevonden kan worden in de verstuurde data. Echter kan er wel minder schadelijke informatie opgevangen worden; namelijk de informatie die niet gecodeerd is via het privacy protocol. Dus alle informatie dat niet in de scopedPDU staat. Met behulp van deze informatie kan de indringer verdere aanvallen voorbereiden. Zo kan de gebruikersnaam gebruikt worden om de computer van deze gebruiker te traceren, om vervolgens de minder sterk beveiligde masquerade toe te passen en dus te doen alsof de indringer deze gebruiker is. De daadwerkelijke taak om te voorkomen dat er (gevoelige) informatie "uitlekt" is in algemene lijnen zeer goed. Voor de rest hebben aanvallen vooral te maken met overige beveiligingen:

- Houdt de gebruiker zijn mond?
- Laat het geen papieren achter in het openbaar?

- Staat de computer in een openbare ruimte en kan er bij de hardware?
- ...

Het cijfer dat we aan disclosure geven is daarom een 3.

3.6 Message Stream Modification

SNMP beschermt hier gedeeltelijk tegen. Tegen her-ordeningen en vertragingen kan SNMP niets doen, maar tegen de vele herhalingen wel. (zie Replaying Messages) Hierdoor kan het wel schadelijk zijn, maar omdat de herhalingen onschadelijk zijn zal dit niet zo snel gebeuren. Omdat SNMP slechts gedeeltelijk beschermd is tegen Message Stream Mod- ifcation, geven we een 2.

3.7 Denial of Service

Het geven van ongeldige commando's

De SNMP-server is zo gebouwd dat het berichten die niet succesvol zijn gedecodeerd afgewezen worden. Deze berichten worden direct verwijderd. Er is dus geen sprake van een buffer die volloopt. Wat natuurlijk wel kan voorkomen is dat er miljoenen onzinnige berichten worden gestuurd, die ieder processortijd kosten van de SNMP-Server. Bij een voldoende aantal opdrachten bestaat de mogelijkheid nog wel dat het systeem dit niet meer aankan.

Het consumeren van systeembronnen

Consumeren van systeembronnen is mogelijk, omdat elke opdracht (zinnig en onzinnig) van een SNMP-agent of indringer processortijd kost om uit te voeren. Hier biedt SNMP geen enkele beveiliging tegen.

Het herconfigureren van systeembronnen

SNMP is bedoeld om een netwerk te beheren. Het heeft mogelijkgheden om de status van netwerkapparaten te wijzigen en uit te lezen, maar het heeft geen mogelijkheid tot het wijzigen van het systeem in de SNMP-agent of server zelf. Dit moet gedaan worden door een netwerkbeheerder op lokaal niveau (d.w.z. wijzigen van het systeem kan alleen door letterlijk bij het systeem te zijn).

Het fysiek vernietigen van een systeembron

Hier biedt het protocol geen enkele bescherming voor. Om dit op te lossen zijn beveiligingen nodig om gebouwen of kamers te beveiligen. Duidelijk is dat dit per situatie afhankelijk is en dus niet opgenomen is in het protocol.

Aangezien er voornamelijk geen beveiliging is tegen een overvloed aan (on)zinnige opdrachten en het maken van een onzinnige opdracht erg simpel is, verdient het protocol het cijfer 1.

3.8 Traffic Analysis

SNMP heeft geen extra bescherming tegen traffic analysis, alleen de stan- daard codering. Toch kost deze methode veel tijd om nuttige informatie te verkrijgen. Het cijfer dat we dus aan Traffic Analysis is een 1.

3.9 Eindresultaat

- Brute-force attack op sleutels: 3
- Replaying Messages: 3
- Modification of Information: 3
- Masquerade: 2
- Disclosure: 3
- Message Stream Modification: 2
- Denial of Service: 1
- Traffic Analysis: 1

4 Dicussie

Een probleem bij de bepaling van de mate van beveiliging van een protocol is hoe je eenduidig kan vaststellen dat de beveiliging goed is. De vraag die dan opkomt is: Wanneer noem je iets beveiligd (en welke factoren bepalen deze beveiliging)? Moet aan alle eisen voldaan worden? Moet het gemiddeld "goed" zijn? Wat is goed? In ons geval besloten we de beveiligingspunten te onderzoeken en het gemiddelde te bepalen.

5 Conclusie

Biedt het User-Based Security Model voldoende bescherming tegen veelvoorkomende kraakpogingen op de gebieden: confidentiality, integrity en authenticity?

Uit de resultaten volgt dat het gemiddelde cijfer voor de beveiliging van het *user-based security model* groter is dan een 2, namelijk:
$(3 + 3 + 3 + 2 + 3 + 2 + 1 + 1)/8 = 2.25$.

Dus volgens onze definitie is het User-Based Security Model een veilig protocol.

Referenties

[STALL] William Stallings, Operating Systems: Internals and Design Principles, 5th Edition, Pearson Prentice Hall, 2005, ISBN:0-13-127837-1.

[RFC3414] User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) (RFC 3414), U. Blumenthal & B. Wijnen, Lucent Technologies, December 2002.