Authenticatie voor the internet of things

Tim Cooijmans T.J.P.M.Cooijmans@student.ru.nl
Manu Drijvers ManuDrijvers@student.ru.nl
Patrick Verleg P.Verleg@student.ru.nl

29 juni 2010

1 Inleiding

1.1 Onderwerp

In kader van het vak research and development hebben wij the internet of things bestudeerd. We zien erg veel potentie in dit concept en het concept is zelfs erg goed te realiseren. Voordat het concept werkelijkheid kan worden, zijn er een aantal problemen die opgelost moeten worden. Deze problemen zijn voornamelijk op het gebied van privacy en beveiliging. Voordat we deze problemen gaan behandelen zullen we eerst het basis idee van the internet of things uitleggen.

1.2 The internet of things

Het idee achter the internet of things is dat steeds meer simpele dingen in staat worden gesteld om met elkaar te communiceren. Ze kunnnen informatie delen met elkaar of een wereldwijde schaal. Dit wil zeggen dat bijvoorbeeld een ding in Nederland direct informatie kan delen met een ding in de Verenigde Staten. Een voorbeeld van een simpele toepassing van deze techniek is als volgt:

Stel je voor, je ligt te slapen. Plotseling gaat je wekker af, zonder dat jij die de avond te voren hebt gezet. De wekker heeft namelijk in je agenda gekeken, en gezien dat je eerste afspraak om 10.45 in het Huygensgebouw is. Hij heeft bepaald dat de reistijd vanaf je huis 20 minuten is, en weet dat je een uur nodig heb om op te staan. Dus om 9.25 begint je wekker te rinkelen, en gaan meteen de lampen branden en staat de verwarming al hoog. Als je na een uurtje de deur uit gaat, merkt de deur dat je vertrekt, en laat het de lampen en verwarming weten dat ze weer uit mogen.

1.3 Problemen

Zoals eerder besproken zijn er een aantal problemen: Allereerst is het zonder encryptie mogelijk om dataverkeer tussen artefacten af te luisteren. Dit is natuurlijk natuurlijk niet de bedoeling gezien deze data privégegevens kan bevatten. Een ander probleem is dat het zonder een goede methode om de authenticiteit van een afzender te controleren, mogelijk is dat artefacten andere artefacten ongeauthenticeerd aansturen. Je wil bijvoorbeeld niet dat een ander zomaar je TV uitschakelt.

Een oplossing voor dit probleem is het signen van berichten met behulp van een hash. Hiervoor worden hash-functies gebruikt. De verzender en ontvanger passen beide deze functie toe op een string bestaande uit de sleutel van de ontvanger en het tijdstip waarop het bericht is verzonden. Een goede hashfunctie is niet te kraken in redelijke tijd met de huidige rekenkracht van computers. Een andere eis aan een hashfunctie is dat het niet veel rekentijd kost om de functie toe te passen. Artefacten in the internet of things hebben over het algemeen weinig rekenkracht en je wilt niet dat het lang duurt om een pakket te verzenden of ontvangen. Ook energieverbruik is een belangrijk punt: Als elk artefact deel is van the internet of things is het niet wenselijk dat elk artefact veel energie verbruikt. Wij gaan onderzoeken welk van de gangbare hash-functies veilig is en van de veilige hashing algoritmes het minste processortijd kost, waarbij veilig als volgt gedefineerd is: Het achterhalen van de sleutel op een computer met een x86-processor met een frequentie van 4 GHz moet gemiddeld minimaal 1 jaar duren.

2 Theoretisch kader

2.1 Kandidaat hash-functies

Wij hebben ons gelimiteerd tot de volgende hash-functies: MD5 en SHA-1 beide toegepast op de volgende string:
`[128 bits key][32 bits tijd]`

2.2 Achterhalen van de sleutel

Omdat MD5[4] en SHA1[1] beide niet injectief zijn, zijn er verschillende strings die naar dezelfde hash gemapt worden[5]. Dit zijn collisions. Normaal gesproken zijn collisions in de beveiliging een probleem. In ons geval zou het kunnen voor komen dat een ongeauthoriseerde zender zijn pakket signeert met een verkeerde sleutel, maar dat deze sleutel toch goedgekeurd wordt omdat de hash van deze verkeerde sleutel hetzelfde is als de hash van de echte sleutel. Toch is dit voor ons geen groot probleem, omdat de hash afhankelijk is van de tijd: Als het pakket met een andere timestamp gesigned wordt geven de juiste sleutel en de foute sleutel compleet andere hashes door het avalanche-effect in SHA1[1] en MD5[4]

Om de sleutel te achterhalen zonder voorkennis, moet je dus simpelweg dingen proberen om de echte sleutel te vinden. Dit is een zogenaamde brute-force attack. Omdat er 2^{128} mogelijke sleutels zijn bij zowel MD5 als SHA1, heb je na 2^{128} pogingen de zeker de echte sleutel achterhaald, er vanuitgaande dat je elke keer een andere sleutel probeert. De kans dat je de echte sleutel pas na 2^{128} pogingen raadt, is natuurlijk erg klein. Als je de helft van de sleutels geprobeerd hebt, is de kans $\frac{1}{2}$ dat je de echte sleutel hebt achterhaald.

Gemiddeld kost het je dus $\frac{1}{2} \cdot 2^{128} = 2^{127}$ pogingen om de sleutel te achterhalen, voor zowel MD5 als SHA1. Een x86 processor zal altijd minstens één instructie nodig hebben om een MD5 of SHA1 hash te berekenen. Op een 4 GHz computer maakt $4 \cdot 10^9 \cdot 60 \cdot 60 \cdot 24 \cdot 365 = 126144 \cdot 10^{12}$ berekeningen per jaar. Dit is $\frac{2^{127}}{126144 \cdot 10^{12}} \approx 10^{21}$ jaar. Beide hash-functies voldoen dus ruimschoots

aan onze veiligheidseis (zie 1.3).

2.3 Energieverbruik

Eerder hebben wij genoemd dat een laag energieverbruik belangrijk is voor the internet of things, omdat je zo veel artefacten hebt en je niet wil dat die allemaal constant veel verbruiken. Het energieverbuik hangt af van wat de microcontroller aan het doen is. De microcontroller heeft twee toestanden: running en sleep. Als er iets te berekenen valt is het in runningstate, anders in sleep. Het energieverbruik in runningstate is gemiddeld 6,4 mA, in sleep is dat minder dan 0,2 μA [3]. Het verbruik in runningstate verschilt per instructie, maar dat verschil is relatief erg klein [2]. Daarom hangt het energieverbruik bijna helemaal af van de processortijd van een programma.

Hierdoor is ons onderzoek naar processortijd voldoende om ook een uitspraak te kunnen doen over het energieverbruik.

3 Methode

Om te kunnen onderzoeken welk van de twee eerdergenoemde authenticatiemethoden het minste processortijd verbruikt, gebruiken wij een MicroChip PIC18F452. Dit is een 20MHz 8-bit microcontroller met 1536 bytes RAM en 32K flash memory[3]. Eerst moesten we MD5 en SHA1 uitvoerbaar maken voor de microcontroller. Hiervoor hebben we uit de TCP/IP libraries van Microchip de MD5 en SHA1 functies gehaald, en ze omgeprogrammeerd zodat ze los uitgevoerd kunnen worden. Omdat het werkgeheugen erg beperkt is en MD5 en SHA1 niet ontworpen zijn voor microcontrollers, was het nodig om de code te optimaliseren. De originele code zorgde voor stack overflow door te veel geneste functie aanroepen. Dit hebben we opgelost door code te kopiëren in plaats van functies aan te roepen. We hebben voor beide hashingfuncties een testprogramma geschreven dat het volgende doet:

- Het programma wordt geïnitialiseerd, de LED is uit.
- Het programma staat klaar om hashing iteraties uit te voeren, de LED gaat aan.
- Het programma gaat een bepaalde string een bepaald aantal keer hashen met MD5 of SHA1.
- Het programma is klaar met hashen, de LED gaat uit.

Hiermee kunnen we meten hoe lang het duurt om een bepaald aantal keer een bepaalde string te hashen met MD5 of SHA1.

We zijn als volgt te werk gegaan: Wij filmen de microcontroller met de LED, dan voeren we een bepaalde opdracht uit, bijvoorbeeld tweehonderd keer MD5. Als dit klaar is kijken we op de video op welke frame de LED aan gaat, en op welke frame de LED uitgaat. Zo kunnen we redelijk precies de tijd bepalen die nodig was om deze hashing opdracht uit te voeren.

We hebben beide hash functies met 100 en 200 iteraties tien keer uitgevoerd. Aan de hand hiervan kunnen we bepalen wat een hash-iteratie voor beide algoritmen kost zonder dat de resultaten beïnvloed worden door eventuele berekenin-

gen buiten de iteraties. Dit kan door de gemiddelde processortijd van 200 iteraties te verminderen met die van 100 iteraties en vervolgens te delen door 100.

4 Resultaten

Testresultaten MD5:

testnummer	MD5 $i = 100$ (ms)	MD5 $i = 200$ (ms)
1	0,754	1,467
2	0,776	1,488
3	0,743	1,470
4	0,770	1,419
5	0,792	1,492
6	0,742	1,477
7	0,765	1,465
8	0,753	1,470
9	0,733	1,453
10	0,795	1,433

Testresultaten SHA1:

testnummer	SHA1 $i = 100$ (ms)	SHA1 $i = 200$ (ms)
1	1,713	3,375
2	1,745	3,353
3	1,767	3,396
4	1,728	3,431
5	1,752	3,468
6	1,732	3,410
7	1,780	3,433
8	1,721	3,392
9	1,753	3,404
10	1,774	3,386

Statistische functies MD5:

	MD5 $i = 100$ (ms)	MD5 $i = 200$ (ms)
gem.	0,762	1,463
gem. abs. afw.	0,017	0,017
std. afw.	0,021	0,023

Statistische functies SHA1:

	SHA1 $i = 100$ (ms)	SHA1 $i = 200$ (ms)
gem.	1,747	3,405
gem. abs. afw.	0,019	0,024
std. afw.	0,023	0,033

MD5 1 iteratie: $\frac{1,463-0,762}{100} = 7,01 \cdot 10^{-3} ms$
SHA1 1 iteratie: $\frac{3,405-1,747}{100} = 16,58 \cdot 10^{-3} ms$

5 Discussie

Wij gaan er vanuit dat de verbinding veilig is. Met andere woorden, pakketten kunnen niet door anderen gelezen worden. Als dit niet het geval is, zou je het

verkeer kunnen afluisteren, en zolang een hash voor een bepaald artefact geldig is, pakketen met andere inhoud kunnen sturen naar dat artefact, met deze afgeluisterde hash als handtekening. In praktijk kun je niet altijd garanderen dat een verbinding veilig is.

Het onderzoek is slechts op één microprocessor uitgevoerd. Een andere microprocessor met een andere instructieset zou andere resultaten kunnen opleveren. Echter gebruikt het overgrote deel van de embedded devices een microcontroller met een vergelijkbare instructieset.

6 Conclusie

Uit onze resultaten blijkt dat een MD5 hash berekenen $7,01 \cdot 10^{-3} ms$ kost en dat bij een SHA1 hash $16,58 \cdot 10^{-3} ms$ is. Beide algoritmen voldoen aan onze gestelde veiligheidseis. Het energieverbruik zal bij MD5 lager zijn omdat de processortijd kleiner is (zie 2.3).

Gezien deze resultaten blijkt dat MD5 de meeste geschikte hashfunctie is voor authenticatie binnen the internet of things.

Referenties

[1] D Eastlake and P Jones. RFC3174: US Secure Hash Algorithm 1 (SHA1). Technical report, RFC Editor United States, 2001.

[2] Mark Hempstead, Michael J. Lyons, David Brooks, and Gu-Yeon Wei. Survey of hardware systems for wireless sensor networks. *Journal of Low Power Electronics*, Volume 4, 2008.

[3] Microchip Technology Inc. *PIC18FXX2 Datasheet*, 2006.

[4] R Rivest. RFC1321: The MD5 message-digest algorithm. Technical report, RFC Editor United States, 1992.

[5] Wojciech A. Trybulec. Pigeon hole principle. *Journal of Formalized Mathematics*, Volume 2, 1990.