Bachelor
Android klient til MiG

Lasse Caramés Andreas Hjordt Jensen

11. januar 2011

Resumé

De senere år har vi set en stor udbredelse af smartphones. Vi er gået fra simple telefoner, som ikke kunne meget mere end at sende beskeder og foretage opkald, til apparater, der reelt ligner laptops i lommestørrelse. Hardwaremæssigt har vi nu nogle muligheder, som udvisker grænsen mellem pc og telefon.

Med introduktionen af Android er afstand mellem pc og telefon blevet yderligere reduceret. Programmører kan anvende en lang række af de samme biblioteker som når, der udvikles til pc'ere. Dette faktum gør, at vi har mulighed for at flytte en lang række af de programmer, som til dagligt anvendes på pc'en, over på vores telefon. Motivationen skulle være lige så åbenlys som motivationen for at introducere fjernbetjeningen til tv apparater. - Den er lige ved hånden!

Grundet ovenstående samt det faktum, at der ikke allerede eksisterer een, er det derfor oplagt at implementere en MiG klient til Android. Udover nogle få mangler har vi formået at implementere en klient, der giver brugeren de samme muligheder som ved brug af hjemmesiden.

Indhold

1 Indledning

Med denne opgave vil vi implementere en klient til Minimum Intrusion Grid, der virker på Android-platformen. I den forbindelse vil vi undersøge de implementeringsmuligheder, der er, og derefter implementere den løsning, vi finder bedst egnet til udviklere såvel som brugere af MiG.

Udarbejdelsen af en MiG klient til Android har kombineret flere datalogiske emner, herunder særligt sikkerhed og netværkskommunikation.

Til slut er vi endt med et program, der er lige så simpelt som princippet bag selve løsningen. Trods dette ligger der mange overvejelser omkring sikkerhed, brugbarhed og genbruglighed bag.

Det endelige resultat er dog meget langt fra vores oprindelige billede af en mobil MiG klient.

2 Analyse

I det følgende vil vi undersøge forskellige muligheder for implementering, samt udvælge den bedst egnede.

2.1 Overordnet design

I starten af projektet var vi nødt til at træffe nogle helt overordnede beslutninger, hvad angår sprogvalg og metode, for at spore os ind på den løsning, vi ville lave.

I denne proces har vi forsøgt både at tage hensyn til brugerne af programmet, altså brugerne af MiG, samt guidelines fra MiG.

2.1.1 Metode

For at tilgængeliggøre MiG på Android-platformen er der 2 muligheder, som vi vil belyse:

- Implementere en klient med egen brugerflade, som ligner og føles som et Android program
- Vise og anvende den eksisterende hjemmeside

Det kan være besværligt at anvende en brugerflade, der er designet til en mobil platform med en betydelig mindre skærm [5]. Netop derfor vælger store aktører som Facebook, Twitter, diverse mediehuse m.m. enten at lave en integreret klient til brugen af deres online indhold eller at lave en særligt designet side til mobile enheder. MiG har ikke et særligt webinterface til mobile enheder.

Med andre ord risikerer man en svært tilgængelig brugerflade med små knapper, der er svære at ramme, tekst, der er svær at læse osv.

Der er til gengæld flere gode ting ved at tilgængeliggøre hele den eksisterende hjemmeside, som den er.

- Al eksisterende funktionalitet tilgængeliggøres straks for brugeren
- Eksisterende brugere kender brugerfladen
- Ny funktionalitet vil straks være tilgængeligt for brugeren
- Der skal kun vedligeholdes én brugerflade

Kan det accepteres, at den eksisterende brugerflade vises, som den er på mobile enheder, vil denne løsning ikke umiddelbart kræve vedligeholdelse, heller ikke hvis der ændres i MiG.

En anden løsning er at lave en klient af Android brugerfladekomponenter, en integreret klient, der giver brugeren mulighed for at interagere med MiG. Med denne løsning ville vi enten skulle udvælge en delmængde af funktionaliteten og implementere denne eller forsøge at implementere al funktionalitet.

Ændres integrationen senere på server-siden, skal Android klienten ligeledes opdateres, og ligeså ved tilføjelse af ny funktionalitet. Det vil altså betyde, at en yderligere brugerflade skal vedligeholdes ud over den eksisterende [11].

Vi ville til gengæld kunne levere en klient, der føles og virker som et Android program, som man ellers kender dem fra Android telefoner. Dvs. at knapper, lister, tekst m.m. vil virke og føles som andre applikationer, man kender [3]. Ydermere vil brugerfladen kunne tilpasses, så den virker og præsenterer indholdet bedst muligt på en mobil enhed. Dvs. at knapper, menuer og andre interaktive kontroller vil have en størrelse og tilgængelighed, der er passende i forhold til en trykfølsom skærm på 2-4 tommer.

2.1.2 Udviklingsplatform

Skal man lave applikationer til Android, findes der et veldokumenteret SDK hertil. Det officielle programmeringssprog heri er Java. Ydermere opdaterer Google løbende SDK'et og Android platformen.

Der findes dog også en integrationsløsning til Android, SL4A, der muliggør brugen af scriptsprog, herunder Python. Idet Python er det foretrukne sprog til MiG [11], ser vi det som en særlig interessant mulighed.

2.2 SL4A

Scripting Layer for Android (SL4A) giver mulighed for at eksekvere forskellige scripting sprog på Android platformen. På nuværende tidspunkt er der varierende understøttelse af Python, Perl, JRuby, Lua, BeanShell, JavaScript og Tcl. I det følgende vil vi udelukkende betragte SL4A ved brugen af Python. Vi har ikke berørt nogle af de andre scripting moduler, som SL4A inkluderer, da disse ikke er relevante for vores projekt.

Ydelse SL4A er kun et tyndt bindeled mellem android platformen og de understøttede scripting sprog. Python scripts eksekveres gennem CPython implementering, som er den mest udbredte Python implementering. CPython er skrevt i C og anses for at være en meget optimeret samt gennemtestet implementering.

Integration Ovenstående er både godt og dårligt. Det er godt, fordi vores kode eksekveres af en meget hurtig fortolker. Men denne hastighed har sin pris. I og med at koden ikke eksekveres af Androids standard virtuelle maskine, Dalvik, afgrænser vi os fra at bruge en stor del af selve Android platformen. Der er nemlig ikke adgang til meget af standard API'en, som der kan gøres brug af ved at bruge Googles egen SDK. Eksempelvis kan det nævnes, at hele den grafiske brugergrænseflade ikke er tilgængelig med undtagelse af nogle få dialoger og WebViewet.

Scripts eksekveret under SL4A er typisk noget simplere end standard Android applikationer, API'en er simpelthen bare skrevet på et højere niveau. Et eksempel herpå kunne være måden hvorpå, der tages billeder. Dette gøres med et enkelt funktionskald til metoden cameraCapturePicture, men som tidligere nævnt er de typisk også noget mere begrænset mht. til muligheder. Det kommer i det konkrete eksempel til udtryk ved, at funktionen kun tager et filnavn som argument, som billedet så gemmes i, og der er derved ikke mulighed for at indstille kontrast, zoom osv.

Modenhed SL4A er, fra udviklernes side, klassificeret som alpha software, hvilket skal forstås som en meget tidlig version af programmet. Men dette ser vi ikke som noget problem, da vores kode, som før nævnt, eksekveres gennem CPython.

Sikkerhed Valget af SL4A er ikke problemfrit. Et af problemerne er, at alle scripts, som eksekveres af SL4A, har de samme rettigheder. Dvs. alle scripts kan foretage opkald, der koster penge, de kan læse information om alle kontakter gemt på telefonen samt en lang række andre ting. Det er som udgangspunkt ikke hensigtsmæssigt, at et lille spil eller lignende har disse rettigheder.

Dette kan specielt blive problematisk, hvis en anden applikation, som kun har adgang til SD-kortet, ønsker at tilegne sig yderlige rettigheder ved at placere eller tilføje kode til et, eller alle, eksisterende scripts. Når scriptet så eksekveres vil den tilføjede kode eksekveres med flere rettigheder end de, der var givet den oprindelige applikation.

Opsummering SL4A er hurtigt og tilbyder en meget highlevel API, men har sin begrænsninger mht. sikkerhed og tillader kun yderst begrænset integration til Android platformen.

2.2.1 Python XMLRPC

Vores originale tanke var at implementere en delmængde[1] af den funktionalitet, vi kender fra webinterfacet. Dette skulle implementeres som en integreret klient.

Da MiG allerede understøtter XMLRPC, burde selve kommunikationen mellem klient og server ikke volde de store problemer.

Umiddelbart havde vi antaget, at det var muligt at tilgå android GUI-framework for derved at opnå en mere integreret brugergrænseflade. Det viste sig dog, at SL4A ikke giver adgang til hele android API'en, men kun til en delmængde af denne. Og hele GUI-delen er slet ikke tilgængelig. Vi ville altså være tvunget til at anvende WebView.

[1] Det skulle som minimum være muligt at starte, stoppe og se status på jobs.

Opsummerende kan det siges, at hvis vi vil bruge XMLRPC fra Python, kan vi ikke bruge Androids GUI-framework. Vi ville være tvunget til at lave vores egen GUI i HTML.

En løsning, som ville give os adgang til Androids GUI framework, er at bruge XMLRPC med Androids SDK. Men dette ville stride mod MiG's grundlæggende retningslinjer om at bruge Python til så meget som muligt [11]. Dette ville også tilføje yderligere kompleksitet til MiG projektet som helhed, da vi herved introducerer endnu et interface, der skal vedligeholdes. Oven i købet i et sprog som os bekendt ikke anvendes andre steder i MiG.

2.2.2 Proxy

Proxyservere anvendes til rigtig mange ting, eksempelvis caching, logning, sikkerhed m.m [6, s. 38].

Vores idé er at anvende en lokal proxyserver til at håndtere X509-certifikatet, som browseren i Android ikke kan. Browseren vil så skulle anvende denne lokale proxyserver, når den tilgår MiG. Det vil betyde, at browseren forbinder med en HTTP forbindelse til proxyserveren, som så forbinder med en HTTPS-forbindelse til MiG. I praksis betyder det, at forbindelsen mellem telefonen og MiG er krypteret, mens forbindelsen mellem browser og proxyserver på telefonen er ren HTTP.

Et af problemerne med denne løsning er anvendelsen af en proxyserver i Android. Der er ingen mulighed for at sætte en proxy i standardbrowseren. Det er til gengæld muligt at sætte en proxy for sin mobile bredbåndsforbindelse. Der er dog et generelt problem på Android, der gør, at der ikke kan sættes en proxy for WIFI-forbindelser [2].

Der findes browsere til Android, hvori det er muligt at sætte proxy, fx. Mozillas Firefox. Idet løsningen er målrettet til folk, der anvender MiG, og derfor til folk, der må forventes at have en hvis teknisk forståelse, ser vi det ikke som et stort problem, at en anden browser skal anvendes til denne løsning, og at det kræver noget konfiguration.

2.3 Nedarvning fra Webview

I Android findes der en klasse, der hedder Webview[2], der kan anvendes til visning af HTML-filer enten fra internettet eller lokalt.

Vi havde en idé om, at vi ville være i stand til at nedarve fra WebViewet og overskrive funktionaliten, der havde med forbindelsen at gøre. Det var dog ikke muligt, idet meget lidt af den egentlige funktionalitet i Webviewet er tilgængeligt for programmøren. Den funktionalitet, der omhandler forbindelsen, ligger tilsyneladende i den underliggende C/C++-kode, og Webviewet er derfor bare et visningsmodul, der er bundet til

[2] `http://developer.android.com/reference/android/webkit/WebView.html`

Webkit [1].

2.4 Endegyldigt valg

De to tungt vejende faktorer, som har haft indflydelse på vores endegyldige beslutning, er MiG's preference mht. Python samt det faktum, at vi ikke ønsker at introducere en ny brugerfla,de der skal vedligeholdes.

Vi ser det klart som en svaghed, at SL4A ikke er godt integreret i android platformen. Havde vi valgt Googles egen SDK, og dermed Java, kunne vi have lavet en løsning, som var mere integreret ved at gøre brug af standardiserede grafiske elementer. Men når vi betragter målgruppen, ser vi dette som et lille offer. En grafisk brugergrænseflade er ikke et must. Og lidt browser konfiguration burde ikke skræmme brugerne af MiG.

Så på trods af den manglende integration ser vi stadig proxy løsning, som den bedst egnede. Ved denne løsning kan brugerne anvende en browser, som er frit tilgængelig og optimeret til Android platformen. De ender altså med at bruge et program samt et interface[3], de allerede er hjemme i.

[3]Web interfacet til MiG

3 Implementering

Vi havde tænkt os at tage udgangspunkt i en eksisterende HTTP-proxyserver, hvis vi kunne finde en, hvor vi med få ændringer kunne bruge den som rammen om vores implementering. Det ville kræve, at den var trådet, simpel, samt at den udgående socket blev håndteret i et funktionskald, vi kunne overskrive.

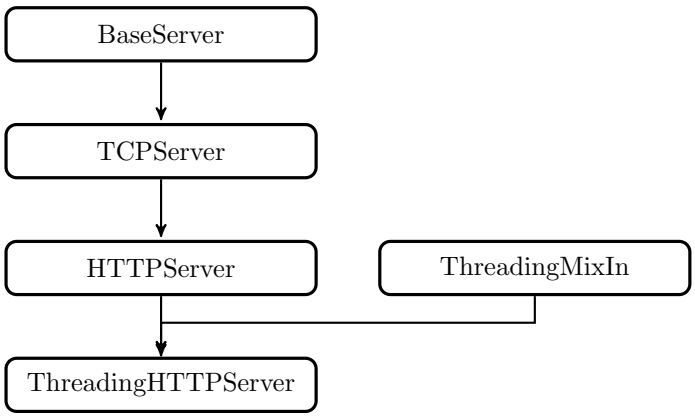
Vores ønske om at bruge en eksisterende proxyserver bunder i ønsket om at anvende kode, der allerede er testet. Ydermere har det den fordel, at udefrakommende opdateringer og forbedringer let kan indføres i egen kode.

3.1 TinyHTTPProxy

TinyHTTPProxy er en open-source HTTP-proxyserver skrevet i Python som har eksisteret siden 2003 [9]. Den er anvendt i mange sammenhæng [7, 8] og må derfor betragtes som et modent stykke software.

Den yder godt med kun ganske lidt overhead [8].

Med mål om at bevare koden med færrest mulige ændringer, så en nyere version let kan erstatte den gamle, har vi kun omskrevet håndteringen af den udgående socket, så denne nu er en instansvariabel. Det har vi gjort for at kunne erstatte den med en SSL-socket efter behov.



Figur 1: Arkitektur af TinyHTTPProxy

TinyHTTPProxy anvender en trådet HTTP-server, bygget af standardbiblioteker, som vist i figur 1. Selve implementeringen af TinyHTTPProxy består derved i implementeringen af en BaseHTTPRequstHandler, der af BaseHTTPServer bliver kaldt for hvert indgående kald til serveren. I den trådede version bliver håndteringen oprettet i en ny tråd for hver indgående kald.

3.2 HTTP

TinyHTTPProxy understøtter GET, POST, CONNECT, HEAD, PUT og DELETE. Vi har valgt ikke at udvide implementeringen med de resterende forspørgselsmetoder [13], idet de ikke er nødvendige for MiG.

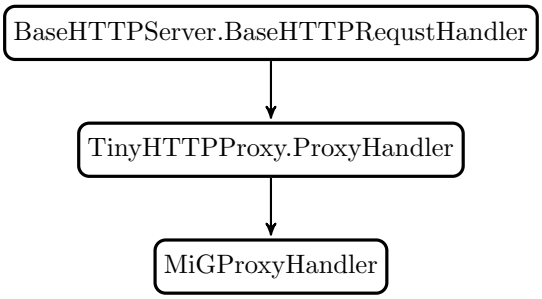
Da TinyHTTPProxy er en simpel proxyserver, der ud over at ændre header-feltet 'connection' til 'close' og slette 'proxy-connection' som anbefalet [13], er implementeringen af GET brugt til alle forbindelsesmetoderne på nær CONNECT. Det kan gøres, fordi headeren, med førnævnte undtagelse, skrives direkte videre til målet for forbindelsen.

Vi gør brug af og understøtter de HTTP metoder, som TinyHTTPProxy også gør og har derved ikke udvidet understøttelsen af protokollen, idet det ikke synes nødvendigt for den opgave, vi skal løse, og det ikke ville bidrage til en bedre understøttelse af MiG.

3.3 MigProxy

Vores implementering nedarver fra TinyHTTPProxy, hvor den overskriver *_connect_to*, der har ansvaret for at forbinde den udadgående forbindelse.

Heri kontrolleres det, om serveren, der forsøges forbundet til, er en af MiGs servere ved at slå værtsnavnet op i et associativt array. Er det ikke tilfældet, laver vi en socket, som TinyHTTPProxy selv ville have gjort det. Er det derimod en MiG server, der forbindes til, indlæser vi certifikat-, nøgle- og certifikatautoritetsfil. Derefter bliver serveren verificeret, som beskrevet under sikkerhed, og en SSL-socket gemmes i den udgående sockets sted.



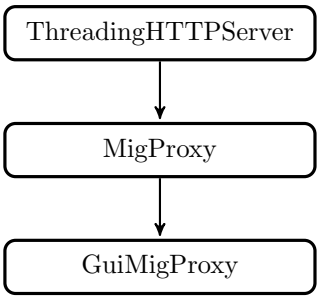
Figur 2: Arkitekturen for MigProxyHandler, der håndterer de indgående kald til MigProxyServeren og derigennem opretter HTTPS forbindelser.

Selve nedarvningen, som illustreret i figur 2, sikrer, at vi har det velfungerende håndteringsmodul fra TinyHTTPProxy. Derved kan vi nøjes med at implementere særtilfældet, der er forbindelse til MiG.

Som nævnt under sikkerhed har vi implementeret så proxyserveren kun lytter på interfacet *lo*. Det har vi gjort ved at nedarve fra den trådede HTTP-server og overskrive *__init__* i MiGProxy. Denne proxyserver startes

ved at sætte den til at håndtere et enkelt indgående kald eller ved at håndtere alle indgående kald.

I forbindelse med introduktionen af brugerfladen havde vi brug for at kunne stoppe og starte proxyserveren. Vi introducerede derfor endnu en nedarvet proxyserver, GuiMigProxy, der implementerer metoder *start* og *stop*. Arkitekturen kan ses i figur 3.



Figur 3: Arkitekturen for GuiMigProxy.

3.4 Overvejelser om SL4A

En måde at begrænse et SL4A scripts mange rettigheder på, ville være at pakke scriptet som en APK[4]. Ved at pakke vores script som APK behøver vi kun at tillade de rettigheder, som netop vores script har behov for. Mere specifikt ville vi kunne nøjes med at tillade adgang til nettet.

En anden fordel ved pakning som APK er, at vi kan distribuere vores pakke uafhængigt af SL4A, da dette, samt Python-pakken, ville være en del af vores pakke.

Hele motivationen for at pakke vores script som APK er altså fire ting.

1. Vores script har kun de rettigheder, som er absolut nødvendige.
2. Vi afskærer brugeren fra SL4A, som vi ser som en sikkerhedsrisiko, grundet de liberale rettigheder SL4A pakken har som standard. Vi fjerner altså SL4A som afhængighed.
3. Vi kan distribuere vores script som standalone applikation.
4. Der kan gennem Android Market distribueres nye versioner af applikationen til brugerne

Men i realiteten er dette ikke trivielt. Ved pakning fungerer det nemlig sådan, at scriptet eksekveres i baggrunden. Programmet startes, og fra brugerens side ser det bare ud som om, det afsluttes igen. Mens scriptet, vores proxy, faktisk kører i baggrunden.

Dette giver en lang række problemer: det er ikke muligt at se eventuelle fejlmeddelelser, det er ikke muligt at se output, det er ikke muligt at

[4]Android programmers pakke format. For mere info se `http://en.wikipedia.org/wiki/APK_(file_format)`

stoppe programmet medmindre, dette gøres gennem en proces manager, der dræber processen, hvilket absolut er en dårlig ide.

Vi omgår ovenstående problemer ved at introducere et WebView, der fungerer som kontrolpanel. Dette holder fokus og sikrer, at applikationen ikke afslutter uden videre. Vi starter vores proxyserver i en tråd, som så kan stoppes og startes fra førnævnte kontrolpanel. Ud over administration af proxytråden tillader kontrolpanelet også brugerdefineret placering af certifikater.

3.5 Brugerdefineret placering af certifikat samt håndtering af dette

Vi har givet brugeren mulighed for selv at definere placeringen af certifikat, nøgle samt rodcertifikat. Dette kan gøres på to måder.

1. Lade brugeren udfylde et HTML input felt for hver fil.
2. Anvende et eksternt program, som tillader, at vi kan vælge en folder.

Første metode er lidt klodset, da den vil introducere hele 3 input felter[5]. Det vil også giver brugeren en del arbejde, og da der er stor sandsynlighed for, at filerne ligger i samme folder, virker anden metode mere attraktiv.

Ved anden metode kom vi frem til, at OI File Manager[6] var brugbar.

Da det ikke er muligt at definere afhængigheder pakker imellem, som vi kender det ved deb, rpm eller ebuilds, må denne afhængighed kontrolleres under afviklingen af vores script.

File manageren kan integreres i vores program ved, at vi undersøger, om den er installeret ved at kalde getLaunchableApplications[7].

Afhængig af udfaldet ville vi enten, med en intent, starte Android Market eller selve File Manageren for i sidste ende at vælge folderen, som indeholder klient certifikatet, nøgle samt rod certifikatet.

Ovenstående viste sig at volde problemer. Når Android Market afsluttes, er processen faktisk ikke færdig. OI File Manageren skal stadig hentes og installeres. Dette betyder, at vi sender en forespørgelse afsted til OI File Manageren før, denne er installeret, hvilket resulterer i et crash. Android SDK'en inholder metoder til at afhjælpe denne situation, men det gør SL4A ikke. Vi overvejede at omgå dette ved at forespørge hvert sekund, om applikationen var blevet installeret. Men hvis applikationen ikke ønskes installeret eller ikke kan installeres, grundet eks. pladsmangel, vil dette resultere i en uendelig løkke.

Vi stødte her på en fejl i OI File Manager, som gør, at første gang den startes, behandler den ikke det ekstra data, der sendes via en Intent.

[5] Det er ikke muligt med HTML at vælge en folder eller flere end een fil ad gangen.
[6] Android program, der kan anvendes til browse filsystemet på telefonen.
[7] `http://code.google.com/p/android-scripting/wiki/ApiReference#getLaunchableApplications`

Det betyder, at der ikke vises en knap til at vælge en folder med. Vi har rapporteret[8] denne fejl og afventer retning.

Ydermere stødte vi på en fejl [4] i SL4A, som betød, at vi ikke kan lukke vores WebView. Det er ikke strengt nødvendigt, at applikationen stopper, når brugeren trykker på tilbageknappen på telefonen, men det er forventet adfærd.

Bortset fra ovenstående fejl og mangler har vi implementeret en brugervenlig proxyserver, som ikke har andre rettigheder end de, der er strengt nødvendige for at fungere.

[8] `http://code.google.com/p/openintents/issues/detail?id=308`

4 Sikkerhed

Sikkerheden i MiG er højt prioriteret. Det er afgørende, at kommunikationen mellem klient og server foregår på en sikker måde. Endvidere er det også relevant, at uvedkommende ikke får adgang til systemet.

I det følgende vil vi ræsonnere om sikkerheden og definere de aspekter, der er relevante for vores implementering.

4.1 SSL/TLS

I det følgende vil vi give en kort gennemgang af Secure Socket Layer, herefter kaldet SSL. I vores tilfælde anvendes SSL af HTTPS til at sikre HTTP kommunikationen mellem vores proxy og MiG serveren.

TLS[9] er en revideret udgave af SSL, som tilføjer protokollen en række kryptografiske algoritmer. Grundprincipperne bag TLS er det samme som SSL, og vi vil tage udgangspunkt i SSL.

4.1.1 Kryptografi ved SSL

Kryptering anvendes hoveddagligt til tre ting i forbindelse med SSL:

Privat data Transformering af tekst således, at kun modtager kan læses teksten

Klient/Server autentificering Verificering af klient og server

Data integritet For at sikre, at der ikke er foretaget ændringer i forhold til den originale tekst

SSL anvender flere kryptografiske metoder til at skabe sikker kommunikation mellem to parter. I det følgende vil vi gennemgå de fire hovedkoncepter.

Symmetrisk kryptering Ved symmetrisk kryptering anvendes en hemmelig nøgle til kryptering samt dekryptering. Det skal bemærkes, at det er den samme nøgle, der bliver brugt af begge parter.

Antages det, at Alice vil sende en besked til Bob, virker metoden ved, at Alice krypterer beskeden med den hemmelige nøgle, sender den krypterede tekst til Bob, som derefter dekrypterer teksten med samme hemmelige nøgle, som Alice har brugt til at kryptere med.

Sikkerheden afhænger af længden på nøglen. Jo længere nøgle, jo sværere vil det også være for en uvedkommende at bryde den. Det skal dog nævnes, at en længere nøgle også medfører længere dekrypteringstid.

[9] Transport Layer Security

Asymmetrisk kryptering Ved asymmetrisk kryptering anvendes et nøglepar, hvor den ene er privat og den anden offentlig. En tekst, der krypteres med den ene, kan kun dekrypteres med den anden. Endvidere er det umuligt at gætte eller beregne den private ud fra den offentlige.

Besked signatur Beskedsignaturen(*eng: Message digest*) anvendes til at verificere integriteten af den transmitterede besked. En signatur er en opsummering af teksten i form af en kort unik numerisk værdi med fast længde. Denne værdi konstrueres ved at beregne en hash-værdi ud fra teksten. Det er altså en unik signatur, der er beregnet ud fra teksten [6].

Både selve beskeden samt dennes signatur krypteres og sendes til modtageren. Ved modtagelse dekrypteres begge dele, og modtageren beregner en ny signatur for teksten, som sammenlignes med den afsenderen vedlage teksten. Hvis en mellemmand har prøvet at modificere teksten under transmissionen, vil den dekrypterede tekst, med stor sandsynlighed, ikke have en matchende beskedsignatur.

Certifikater Et certifikat bruges til at identificere en maskine, en service, en person el. lign. og kan indeholde en lang række felter. De for os relevante felter indeholder data til identifikation, validering samt en angivelse af hvilken autoritet, der har signeret certifikatet.

Identifikation - I vores tilfælde er det relevant, at vi er i stand til at identificere serveren, som vi kommunikerer med. Dette gøres ved at kontrollere, at feltet Common Name indeholder en URL, der matcher domænenavnet på den server, vi har oprettet forbindelse til.

Validering - Et valideringskriterium er, at certifikatets tidsramme stemmer overens med den egentlige tid. Altså er certifikatet validt her og nu. Dette kan valideres ved brug af certifikatfelterne notBefore og notAfter.

Autentifikation - Det er også vigtigt, at vi er sikre på, at beskeden er, som den blev sendt. Det gøres typisk af de biblioteker, der anvendes til håndtering af vores certifikater. Rent praktisk gøres det ved at dekryptere beskeden og kontrollere beskedsignaturen mod den beskedsignatur, man selv udregner på baggrund af beskeden.

Autoritet - Hele princippet i certifikater er baseret på, at vi har tillid til dem, der har udstedt certifikatet. Det kan enten være en certifikat autoritet, men det kan også være selvsigneret som med MiG.

4.1.2 SSL-protokollen

SSL-protokollen ligger ovenpå TCP-protokollen og vil derfor være i stand til at sikre al information sendt via TCP, herunder HTTP. Dvs., at SSL protokollen ligger i netværkslaget mellem TCP/IP og applikationslaget.

Øvre dele af systemet, der befinder sig i applikationslaget, vil derfor kommunikere med SSL-protokollen i stedet for TCP/IP, og SSL-protokollen vil på deres vegne overtage den direkte kommunikation med TCP/IP.

Idet SSL-protokollen overtager den videre kommunikation fra højereliggende lag, har denne mulighed for at aftale kryptering og verificere identiteten af serveren, inden kommunikationen kan påbegyndes.

SSL handshake Ved oprettelse af en SSL-forbindelse udføres først et såkaldt 'handshake', hvori kryptering mv. aftales [6, 10]:

1. Klienten sender første besked, der indeholder SSL version samt den kryptering, klienten foretrækker. Ydermere sendes noget tilfældigt genereret data.
2. Serveren sender SSL version, krypteringsindstillinger, noget tilfældigt genreret data. Ydermere sendes serverens offentlige nøgle.
3. Klienten autentificerer serveren. Kan det lade sig gøre, generes en nøgle, som krypteres med serverens offentlige nøgle. Har serveren bedt klienten autentificere sig signeres noget data, unikt for sessionen, som begge parter kender, og sender det med.
4. Har serveren bedt om det, autentificerer serveren klientet. Lykkes det anvender serveren sin private nøgle til at dekryptere den hemmelige prænøgle. Den hemmelige prænøgle anvendes til at generere den hemmelige nøgle.
5. Både klient og server anvender den hemmelige nøgle til at generere sessionsnøgler, som er symmetriske nøgler. Disse bruges til at kryptere og dekryptere information sendt mellem server og klient. Ydermere anvendes de til at validere integriteten af beskederne.
6. Klienten afslutter oprettelsen ved at sende en besked til serveren indeholdende sin sessionsnøgle.
7. Serveren sender en lignende besked, og forbindelsen er oprettet.

Server autentificering Under klientens autentificering af serveren valideres serveren ved at kontrollere om

- serverens certifikats oprettelses- og udløbsperiode stemmer overens med dags dato.
- certifikatautoriteten er blandt dem, klienten stoler på.
- certifikatautoritsfilen kan valideres mod serverens certifikat.
- domænenavnet i serverens certifikat stemmer overens med serverens domænenavn.

Hvis alle disse kontroller bestås, kan klienten tro på, at serveren er, den den siger, den er, og oprettelsen af forbindelsen fortsætter (skridt 3 i SSL handshake).

Klient autentificering Serveren kan autentificere klienten ved at kontrollere om

- klientens offentlige nøgle validerer med brugerens digitale signatur.
- klientens certifikat ikke er udløbet og er aktivt.
- klientens certifikat er udstedt at en certifikatautoritet, serveren har tillid til.
- klientens digitale signatur kan valideres med certifikatautoritetens offentlige nøgle
- klientens certifikat er blandt tilladte brugere på serveren

Verificeres alle disse kontroller er klienten autentificeret, og serveren kan tillade den videre oprettelse af SSL-forbindelsen.

4.2 Man-in-the-middle

Et "mand-i-midten angreb"er et angreb, hvor en mellemmand, Mallory, går ind som mellemled i en forbindelse mellem Alice og Bob [6].

Hvis Alice beder Bob om hans offentlige nøgle i forsøget på at oprette en sikker forbindelse, og Bob returnerer denne, men Mallory opfanger den, er et muligt mand-i-midten angreb igang. Mallory sender sin offentlige nøgle tilbage til Alice i Bobs sted. Alice vil nu sende sin offentlige nøgle tilbage til afsenderen, som hun tror er Bob, men den sendes nu også til Mallory, som sender sin offentlige nøgle til Bob i Alices sted. Alice og Bob krypterer nu deres beskeder med den offentlige nøgle, de har modtaget, hvilket er Mallorys, og sender deres beskeder gennem Mallory, som kan ændre i dem, som hun vil. Mallory udgiver sig nu for at være begge parter, og idet hun har begge parters offentlige nøgle, vil de beskeder, hun sender, være krypteret, som var de fra henholdsvis Alice eller Bob.

For at undgå dette scenarie har vi implementeret en kontrol af domænenavnet for afsenderen af den offentlige nøgle. Herved begrænser vi muligheden for, at der ikke kan være en mellemmand, der udgiver sig for at være MiG, og det derved rent faktisk er MiG, vi kommunikerer med.

4.3 Kodeord på nøglefil

Det er afgørende for sikkerheden af en nøglefil, at den er beskyttet med et kodeord. Bliver en nøglefil uden kodeord stjålet, kan den direkte anvendes, og brugerens identitet er derved blevet stjålet i forhold til den/de server(e), hvor nøglefilen passer til.

Vores implementering kræver, at man gemmer nøglefilen uden kodeord på sin telefon. Det gør den, fordi SSL-implementeringen til Python, som vi, i vores implementering gør brug af, ikke understøtter dette givet som argument. Det promptes der i stedet efter direkte i konsollen.

Netop det, at vores løsning kræver en nøglefil uden kodeord, ser vi som et af de største sikkerhedsbrister i vores implementering.

4.4 Andre applikationer

Ligesom browseren kan tilgå vores proxy, kan ethvert andet program på enheden det også. Konsekvensen heraf er den samme som ved XSS, som beskrevet herunder.

Dette kunne afhjælpes ved at implementere autentifikation i proxyserveren. Det ville dog øge kompleksiteten af programmet og vil ikke stå mål med sandsynligheden for en udnyttelse af sårbarheden.

4.5 XSS

XSS[10] er en måde hvorpå, man får en klient til at afvikle et script [12].

Netop fordi alle forbindelser i browseren går gennem proxyserveren, og denne automatisk indlæser certifikater efter behov, er en bruger særligt sårbar overfor XSS.

Et eksempel herpå kunne være, at en bruger kører proxyserveren og tilgår en hjemmeside, der har et Javascript, der forsøger at igangsætte et job på MiG. Javascriptet bliver afviklet i brugerens browser, og vil derfor med brugeren rettigheder på MiG kunne eksekvere sin handling.

Det skal dog siges, at udvikleren af det ondsindede Javascript skal vide, at MiG skal tilgås via HTTP og ikke HTTPS. Det skal altså være et meget målrettet angreb, hvor der er kendskab til proxyserveren og MiG.

4.6 Forbindelsen til proxyserveren

Vi har lavet proxyserveren, så den kun lytter på kald fra værten selv, altså på *loopback*-interfacet. Det har vi gjort for, at brugere på samme netværk ikke kan tilgå MiG gennem proxyserveren og derved tilgå MiG som personen, der afvikler denne.

[10] Cross-site scripting

5 Verificering

For at sikre at vores implementering virker som beskrevet og forventet, har vi gennemgået nogle forskellige tests. Disse spænder over lavpraktiske tests mod MiG til gennemgang af koden. Vi vil i dette afsnit gennemgå denne proces.

5.1 _connect_to

Den centrale del af vores implementering ligger i funktionen *_connect_to*. Vi vil derfor gennemgå loggikken i denne herunder.

Algorithm 1 Forbind via SSL socket, hvis domænenavnet tilhører MiG, ellers forbind til domænenavnet

```
1:  if netloc.containsPort() then
2:      p ← netloc_port
3:      h ← netloc_host
4:  else
5:      p ← 80
6:      h ← netloc
7:  end if
8:  if netloc.isA(MiG_host) then
9:      p ← MiG_port
10:     key ← MiG_key
11:     cert ← MiG_cert
12:     ssl ← 1
13: end if
14: sock ← connect(h, p)
15: if ssl = 1 then
16:     sock ← ssl(sock)
17:     verifyhost(sock)
18: end if
```

Selve algoritmen er rigtig simpel. Input til algoritmen er *netloc*, der kommer fra en URL på følgende form:

```
scheme://netloc/path;parameters?query#fragment
```

I linje 1 undersøger vi, om denne indeholder en port. Gør den det, så splitter vi *netloc* i port og domæne. Gør den ikke, bruger vi standardporten 80.

I linje 8 kontrollerer vi, om domænenavnet er i vores liste af MiG-domæner. Er det et MiG-domæne, ved vi, at vi skal lave en sikker forbindelse. Vi indlæser derfor nøgle og certifikat i linjerne 10-11 og sætter porten.

I linje 14 forbinder vi vores socket til domænet på den port, vi er kommet frem til.

I linje 15 tjekker vi, om der er sat et flag, der indikerer, at det er en ssl-forbindelse, og er det det, så pakker vi vores socket i et SSL lag. Derefter verificerer vi, om domænenavnet og COMMON_NAME er det samme.

5.2 Lavpraktisk test

Vi har som en del af vores kvalitetsikring forsøgt at anvende vores implementering i nogle AVD'er[11] og på en telefon med Android 2.2.

I AVD'erne har vi forsøgt at afvikle programmet på Android 2.1 og Android 2.2 for at sikre os, at der ikke var nogle problemer med kompatibiliteten.

Opsætningen til testene foregik ved, at certifikater blev kopieret til `/sdcard/.mig/`. Herefter blev vores proxyserver installeret på telefonen, og proxyforbindelsen blev sat op enten i en browser eller i telefonen.

Selve testen gik ud på at tilgå `http://dk.migrid.org` og anvende funktionaliteten på siden:

1. Tilgå forsiden
2. Tryk Submit Job og upload et job
3. Tryk Files og lav en folder, slet en folder, upload en fil og slet en fil
4. Tryk Jobs og fremsøg job
5. Gå til Shell og brug den

Derefter tilgik vi `http://www.migrid.org` og `http://dk-cert.migrid.org`, der begge videresender én til sin forside på `dk.migrid.org`. Alle menuer, dialoger og andet funktionalitet, vi faldt over på siden, virkede som forventet.

Ydermere er samme gennemgang lavet i Ubuntu med Firefox. Herudover har vi anvendt proxyserveren til vores helt normale surfing fra vores computere under rapportskrivningen.

Vi har valideret disse tests ved at holde den resulterende side op mod det forventede resultat, vi kunne fremskaffe ved at logge på med vores certifikater.

5.3 Ydelse

Vi har testet ydelsen af vores proxyserver med forskellige mål for øje. Det har vi gjort for at sikre at:

- Svartiden ikke forringes betydeligt ved brug af proxyløsningen
- Ressourceforbruget kontrolleres ved længere tids brug

[11] Android Virtual Device

5.3.1 Svartider

For at sikre os, at svartiderne ved brugen af proxyserveren ikke stiger proportionelt med antallet af forespørgelser, har vi igangsat proxyserveren på en PC. Over en periode på 20 min har vi vha. ab[12] sendt forespørgsler gennem proxyserveren mod `http://dk.migrid.org`.

Testen blev udført på en 20mbit/20mbit forbindelse. Der blev sendt:

1. 300 forespørgelser 8 ad gangen
2. 100 forespørgelser 10 ad gangen
3. 200 forespørgelser 10 ad gangen

Testen viste, at 95% af forespørgelserne blev besvaret inden $185ms$, og $2,5\%$ af forespørgelserne havde en større svartid end $380ms$.

Vi har brugt stresstesten til at overveje, om vores gennemsnitlige svartid er acceptabel for en bruger, der sidder på MiG, og vi synes ikke at opleve betydeligt overhead.

Der er dog nogle få af forespørgelserne, der hænger lidt, når presset øges, men ved almindelig brug er det ikke et problem.

5.3.2 Ressourceforbrug

For at måle vi frigiver ressourcer og afslutter vores tråde, har vi kørt en test over 4 timer, hvor vi undervejs har kigget på hukommelsesforbruget.

Selve testen blev udført ved, at vi startede ab med nedenstående antal forespørgelser over et interval på 5 timer:

1. 1000 forespørgelser 4 ad gangen
2. 1000 forespørgelser 5 ad gangen
3. 400 forespørgelser 1 ad gangen
4. 400 forespørgelser 2 ad gangen
5. 400 forespørgelser 3 ad gangen
6. 400 forespørgelser 4 ad gangen
7. 10000 forespørgelser 10 ad gangen
8. 5000 forespørgelser 10 ad gangen

Der blev alt i alt foretaget 18600 forespørgelser gennem proxyserveren mod MiG.

Under og efter forsøget kiggede vi på hukommelsesforbruget, der under hele testen faldt til 0.4% efter hver kørsel, og alle tråde var lukket ved afslutning.

Idet proxyserveren ryder helt op efter sig selv, og alle tråde lukkes efter brug, er der ingen tegn på lækage af hukommelse.

[12] Apache HTTP server benchmarking tool

6 Fejl og mangler

I dette afsnit vil vi gennemgå de fejl og mangler, der er i vores implementering.

6.1 Brugerfladen

Brugerfladen er ikke distribueret med APK'en. Grundet en fejl i SL4A kan der ikke refereres til ressourcer[13].

Brugerfladen skal derfor manuelt placeres i roden af SD-kortet og skal hedde index.html.

6.2 Filmanager

Filmanageren vi gør brug af har en fejl, der gør, at den ikke virker første gang, den anvendes efter installation.

Ydermere har vi ikke mulighed for at afvente installationen, hvorfor vores implementering lukker imens.

6.3 Fennec

Det er ikke muligt på en telefon, der ikke er rooted, at konfigurere proxyindstillingerne i Fennec. Vi havde håbet på, at vi kunne sætte disse ved opstart af programmet, men det tillades ikke.

6.4 Manglende validering af input

Der valideres ikke på, om certifikater mv. ligger, hvor brugeren påstår, er valide, er uden kode osv.

6.5 Manglende output

Der er ikke muligt at se hvad, der skrives til stdout. Dette skyldes, at programmet afvikles i baggrunden, når det pakkes som APK med SL4A. Dette kan evt. implementeres ved at skrive stdout til en fil på SD-kortet.

6.6 Kodeord på nøglefil

Der må ikke være kode på nøglefilen. Koden kan evt. fjernes ved brug af OpenSSL således:

```
openssl rsa -in key.pem -out newkey.pem
```

[13] `http://code.google.com/p/android-scripting/issues/detail?id=470`

7 Brugervejledning

Følg instruktionerne på `http://sites.google.com/site/minimumintrusiongrid/getting-started` for at få adgang til MiG.

7.1 Installation

For at anvende proxyserveren skal der fortages en række installationer samt konfigurationer.

7.1.1 Installation af certifikater

Certifikaterne overføres til SD-kortet på din telefon enten ved at mounte kortet gennem et USB kabel eller ved at forbinde gennem Android debugger[14].

Som udgangspunkt anbefaler vi, at de lægges under /sdcard/.mig/, men stien kan defineres under kørsel af GUI versionen af klienten.

7.1.2 Installation af APK

Der findes flere forskellige programmer i Android Market, som kan bruges til at installere APK'er fra SD-kortet. Eks. Apps Installer, App Installer, z-App Installer eller Fast Installer. Android debuggeren kan også bruges som vist nedenfor. (Det antages, at Android SDK er installeret på computeren)

1. Tilslut telefonen gennem USB.
2. Åben en terminal og skriv: adb devices
   Bekræft at enheden er tilsluttet.
3. I terminalen skrivs: adb install /path/to/migrid.apk

Grundet en fejl i SL4A var det ikke muligt for os at distribuere brugerfladen sammen med APK'en, derfor skal files index.html overføres til SD-kortet og placeres i roder af dette. Dvs. /sdcard/index.html. Der er indgivet en patch, som retter dette problem, se `http://code.google.com/p/android-scripting/issues/detail?id=470`

Såfremt ovenstående trin blev udført med succes, skulle du nu gerne have installeret Android MiG Klient.

7.1.3 Installation af SL4A samt script

SL4A kan installeres ved at følge instruktionerne på `http://code.google.com/p/android-scripting/`.

BEMÆRK: SL4A er ikke nødvendigt hvis GUI[15] versionen anvendes.

Efter installation af SL4A overføres proxyserver script filerne[16] til SD-kortet. Alle SL4A scripts skal placeres under /sdcard/sl4a/scripts/

[14] adb push .mig/ /sdcard/.mig/
[15] DVS. den pakkede apk version
[16] migrid.py og TinyProxy.py

7.1.4 Installation af Fennec

I Android 2.2 er det ikke muligt at angive globale proxy instillinger for WIFI forbindelser[17]. Det kan man derimod i Fennec[18], og derfor skal denne installeres.

Åben Android Market og søg efter 'firefox' og installer den applikation, der hedder: Mozilla Firefox Web Browser

7.2 Anvendelse

7.2.1 Konfiguration af Fennec

Før proxyen kan anvendes, skal Fennec konfigureres til at bruge den. Dette gøres ved at følge nedenstående trin.

1. Åben Fennec of gå til about:config
2. Skriv "proxy.http"i søge feltet
3. Klik på "network.proxy.http"og indtast 127.0.0.1
4. Klik på "network.proxy.http"og indtast 8000

Efter konfiguration af Fennec kan applikationen startes.

Der er mulighed for at angive placeringen af certifikatfilerne samt starte/stoppe proxyen.

[17] Det kan sagtens angives for mobile bredbåndsforbindelser per APN
[18] Firefox for mobile enheder

8 Konklusion

Vi har muliggjort tilgangen til MiG fra Android-platformen, omend med en løsning, der ligger langt fra vores indledende forestillinger. Løsningen giver brugeren mulighed for at anvende den eksisterende brugergrænseflade.

Sikkerhedmæssigt er vores løsning kompatibel med MiG. Vi verificerer serveren og implementerer sikker kommunikation server og klient imellem. Vi har dog været nødt til at fjerne kodeordet fra nøglefilen, hvilket introducerer et åbenlyst sikkerhedsbrist.

Yderligere har vi belyst en række sikkerhedsmæssige aspekter, der er relevante for vores implementering.

Vi har lavet en løsning, der er forholdsvis simpel at bruge, men ikke er særlig godt integreret i Android. Løsningen er yderligere begrænset grundet fejl i de værktøjer, vi har valgt at bruge. Dette er dog fejl, der forventes rettet i fremtidige versioner.

Vi har baseret vores kode på en eksisterende, velfungerende implementering og struktureret vores egen kode således, at denne let kan udvides. hvad angår funktionalitet, og tilpasses andre services, der anvender samme sikkerhedsmodel(HTTPS med X.509).

Litteratur

[1] Android architecture. http://developer.android.com/guide/basics/what-is-android.html, December 2010.

[2] Android issue 1273. http://code.google.com/p/android/issues/detail?id=1273, December 2010.

[3] User interface guidelines. http://developer.android.com/guide/practices/ui_guidelines/index.html, December 2010.

[4] Webview exit bug. http://code.google.com/p/android-scripting/issues/detail?id=467#c6, December 2010.

[5] Luca Chittaro. *Human-computer interaction with mobile devices and services*. Springer-Verlag Berlin Heidelberg, 5th edition, 2003.

[6] Geirge Coulouris, Jean Dollimore, and Tim Kondberg. *Distributed Systems - Concepts and Design*. Addison Wesley, 4th edition.

[7] Gail Kaiser et. al. Rust: The reusable security toolkit. http://cups.cs.cmu.edu/soups/2008/posters/atreya.pdf, 2008.

[8] Wouter Joosen et. al. Sessionshield: Lightweight protection against session hijacking. 2010.

[9] Suzoki Hisao. Tiny http proxy. http://www.oki-osk.jp/esc/python/proxy/, June 2003.

[10] Mozilla. Introduction to ssl. https://developer.mozilla.org/en/Introduction_to_SSL, September 2005.

[11] Henrik Hoey Karlsen og Brian Vinter. Minimum intrusion grid - the simple model. 2005.

[12] Kevin Spett. Cross site scripting. http://people.cis.ksu.edu/~hankley/d764/Topics/SPIcross-sitescripting.pdf, 2005.

[13] W3C/MIT. Hypertext transfer protocol – http/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999.

9 Bilag

A Kildekode

```
#!/bin/sh -
"exec" "python" "-O" "$0" "$@"

__doc__ = """Tiny HTTP Proxy.

This module implements GET, HEAD, POST, PUT and DELETE methods
on BaseHTTPServer, and behaves as an HTTP proxy. The CONNECT
method is also implemented experimentally, but has not been
tested yet.

Any help will be greatly appreciated. SUZUKI Hisao
"""

__version__ = "0.2.1"

import BaseHTTPServer, select, socket, SocketServer, urlparse

class ProxyHandler (BaseHTTPServer.BaseHTTPRequestHandler):
    __base = BaseHTTPServer.BaseHTTPRequestHandler
    __base_handle = __base.handle

    server_version = "TinyHTTPProxy/" + __version__
    rbufsize = 0                        # self.rfile Be unbuffered

    def handle(self):
        (ip, port) =  self.client_address
        if hasattr(self, 'allowed_clients') and ip not in self.allowed_clients:
            self.raw_requestline = self.rfile.readline()
            if self.parse_request(): self.send_error(403)
        else:
            self.__base_handle()

    def _connect_to(self, netloc):
        i = netloc.find(':')
        if i >= 0:
            host_port = netloc[:i], int(netloc[i+1:])
        else:
            host_port = netloc, 80
        print "\t" "connect to %s:%d" % host_port
        try: self.sock.connect(host_port)
        except socket.error, arg:
            try: msg = arg[1]
            except: msg = arg
            self.send_error(404, msg)
            return 0
        return 1

    def do_CONNECT(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
        try:
51          if self._connect_to(self.path):
                self.log_request(200)
53              self.wfile.write(self.protocol_version +
                                 " 200 Connection
                                    established\r\n")
55              self.wfile.write("Proxy-agent: %s\r\n" %
                    self.version_string())
                self.wfile.write("\r\n")
57              self._read_write(300)
        finally:
59          print "\t" "bye"
            self.sock.close()
61          self.connection.close()

63  def do_GET(self):
        (scm, netloc, path, params, query, fragment) =
            urlparse.urlparse(
65          self.path, 'http')
        if scm != 'http' or fragment or not netloc:
67          self.send_error(400, "bad url %s" % self.path
                )
            return
69      self.sock = socket.socket(socket.AF_INET, socket.
            SOCK_STREAM)
        try:
71          if self._connect_to(netloc):
                self.log_request()
73              self.sock.send("%s %s %s\r\n" % (
                    self.command,
75                  urlparse.urlunparse(('', '', path,
                        params, query, '')),
                    self.request_version))
77              self.headers['Connection'] = 'close'
                del self.headers['Proxy-Connection']
79              for key_val in self.headers.items():
                    self.sock.send("%s: %s\r\n" % key_val
                        )
81              self.sock.send("\r\n")
                self._read_write(self.sock)
83      finally:
            print "\t" "bye"
85          self.sock.close()
            self.connection.close()
87
    def _read_write(self, max_idling=5):
89      iw = [self.connection, self.sock]
        ow = []
91      count = 0
        while 1:
93          count += 1
            (ins, _, exs) = select.select(iw, ow, iw, 3)
95          if exs: break
            if ins:
97              for i in ins:
                    if i is self.sock:
99                      out = self.connection
                    else:
101                     out = self.sock
                    data = i.recv(8192)
103                 if data:
                        out.send(data)
```

```
                        count = 0
            else:
                print "\t" "idle", count
            if count == max_idling: break

    do_HEAD = do_GET
    do_POST = do_GET
    do_PUT  = do_GET
    do_DELETE=do_GET

class ThreadingHTTPServer (SocketServer.ThreadingMixIn,
                           BaseHTTPServer.HTTPServer):
                               pass

if __name__ == '__main__':
    from sys import argv
    if argv[1:] and argv[1] in ('-h', '--help'):
        print argv[0], "[port [allowed_client_name ...]]"
    else:
        if argv[2:]:
            allowed = []
            for name in argv[2:]:
                client = socket.gethostbyname(name)
                allowed.append(client)
                print "Accept: %s (%s)" % (client, name)
            ProxyHandler.allowed_clients = allowed
            del argv[2:]
        else:
            print "Any clients will be served..."
        BaseHTTPServer.test(ProxyHandler,
            ThreadingHTTPServer)
```

../Bachelor/src/TinyProxy/TinyHTTPProxy.py

```
#!/usr/bin/python

from TinyHTTPProxy import ProxyHandler,
    ThreadingHTTPServer
import os, socket, BaseHTTPServer
from OpenSSL import SSL as ssl

class MigProxy(ProxyHandler):
    cert_port, sid_port = 443, 443

    #key_path = os.path.expanduser('~/.mig/key.pem')
    # Path to key without passphrase
    # Use default .mig location with fallback to non-
        standard Android location
    key_path = os.path.expanduser('~/.mig/tmp.pem')
    if not os.path.exists(key_path):
        key_path = os.path.expanduser('/sdcard/.mig/tmp.
            pem')
    cert_path = os.path.expanduser('~/.mig/cert.pem')
    if not os.path.exists(cert_path):
        cert_path = os.path.expanduser('/sdcard/.mig/cert
            .pem')
    cert_dict ={'cert_port': cert_port,
                'ssl_cert': cert_path,
                'ssl_key': key_path}
    wrap_targets = {'www.migrid.org': cert_dict,
                    'dk.migrid.org': cert_dict,
                    'dk-cert.migrid.org': cert_dict,
```

```
25                      'dk-sid.migrid.org': cert_dict}
    wrap_ssl = False
27
    def verify_cb(self, conn, x509, errno, errdepth,
        retcode):
29 #         print self.host
        print "CN: %s" % x509.get_subject().commonName
31      print "errdepth: %d" % errdepth
        print "errno: %d" % errno
33      return True
        if errno == 0:
35          if errdepth != 0:
                # don't validate names of root
                    certificates
37              return True
            else:
39              #if x509.get_subject().commonName != self
                    .host:
                return x509.get_subject().commonName == u
                    "*.migrid.org"
41      else:
            print "Hmmmm...."
43          return False

45  def _connect_to(self, netloc):
        # Split netloc into host and port
47      if netloc.find(':') >= 0:
            host, port = netloc.rsplit(':',1)
49      else:
            host, port = netloc, 80
51      # Check if netloc is in targets
        if netloc in self.wrap_targets:
53          host, port = host, self.wrap_targets[netloc][
                'cert_port']
            self.wrap_ssl = True
55          self.key = self.wrap_targets[netloc]['ssl_key
                ']
            self.cert = self.wrap_targets[netloc]['
                ssl_cert']
57      #try:
        self.sock.connect((host, int(port)))
59      if self.wrap_ssl:
            self.host = host
61          context = ssl.Context(ssl.SSLv23_METHOD)
            context.use_privatekey_file(self.key, ssl.
                FILETYPE_PEM)
63          context.use_certificate_file(self.cert, ssl.
                FILETYPE_PEM)
            context.set_verify(ssl.VERIFY_PEER | ssl.
                VERIFY_FAIL_IF_NO_PEER_CERT, self.
                verify_cb)
65          context.set_verify_depth(2)
            # create socket and connect to server
67          self.sock = socket.socket()
            self.sock = ssl.Connection(context, self.sock
                )
69          self.sock.connect((host, int(port)))
            self.sock.do_handshake()
71      #except Exception as ex:
        #    print "Shit.."
73      #    print type(ex)
        #    print ex.args
```

```
75          #     print ex
            #     return 0
77          return 1

79  class MiGProxy(ThreadingHTTPServer):
        """HTTP(S) Proxy listening only on local interface
            but forwarding
81      connections on any available internet interface.
            Connections to a
        set of configured MiG URLs are transparently wrapped
            in SSL with
83      client certificate while all other connections are
            left alone.
        """
85      def __init__(self, server_address, handler):
            # Force access from local interface only using
                method from
87          # http://code.activestate.com/recipes/439094/
            # to find and use only the IP address of the
                loopback device
89          import fcntl, struct
            # We could support other device only by changing
                "lo" to
91          # another device name
            device_name = "lo"
93          s = socket.socket(socket.AF_INET, socket.
                SOCK_DGRAM)
            listen_address = socket.inet_ntoa(fcntl.ioctl(
95              s.fileno(), 0x8915,  # SIOCGIFADDR
                struct.pack('256s', device_name))[20:24])
97          ThreadingHTTPServer.__init__(self, (
                listen_address, server_address[1]), handler)

99  if __name__ == '__main__':
        from sys import argv
101     if argv[1:] and argv[1] in ('-h', '--help'):
            print argv[0], "[port [allowed_client_name ...]]"
103     else:
            if argv[2:]:
105             allowed = []
                for name in argv[2:]:
107                 client = socket.gethostbyname(name)
                    allowed.append(client)
109                 print "Accept: %s (%s)" % (client, name)
                MigProxy.allowed_clients = allowed
111             del argv[2:]
            else:
113             print "Any clients will be served..."

115         BaseHTTPServer.test(MigProxy, MiGProxy)
```

../Bachelor/src/TinyProxy/MigProxy.py

B ab resultater - svartider

Herunder findes resultaterne fra ab-testen i forbindelse med måling af svartider.

```
$ ab -n 300 -c 8 -X localhost:8000 http://dk.migrid.org/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking dk.migrid.org [through localhost:8000] (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Finished 300 requests


Server Software:        Apache/2.2.9
Server Hostname:        dk.migrid.org
Server Port:            80

Document Path:          /
Document Length:        581 bytes

Concurrency Level:      8
Time taken for tests:   5.774 seconds
Complete requests:      300
Failed requests:        0
Write errors:           0
Total transferred:      274200 bytes
HTML transferred:       174300 bytes
Requests per second:    51.95 [#/sec] (mean)
Time per request:       153.985 [ms] (mean)
Time per request:       19.248 [ms] (mean, across all concurrent requests)
Transfer rate:          46.37 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:    96  152 116.5    119     814
Waiting:       96  151 115.8    118     812
Total:         96  152 116.5    119     814

Percentage of the requests served within a certain time (ms)
  50%    119
  66%    128
  75%    140
  80%    158
  90%    185
  95%    390
  98%    769
  99%    779
 100%    814 (longest request)
```

```
$ ab -n 100 -c 10 -X localhost:8000 http://dk.migrid.org/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking dk.migrid.org [through localhost:8000] (be patient).....done


Server Software:        Apache/2.2.9
Server Hostname:        dk.migrid.org
Server Port:            80

Document Path:          /
Document Length:        581 bytes

Concurrency Level:      10
Time taken for tests:   1.255 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      91400 bytes
HTML transferred:       58100 bytes
Requests per second:    79.68 [#/sec] (mean)
Time per request:       125.504 [ms] (mean)
Time per request:       12.550 [ms] (mean, across all concurrent requests)
Transfer rate:          71.12 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       1
Processing:    95  120  14.7    118     163
Waiting:       95  119  14.5    118     162
Total:         95  120  14.8    118     163

Percentage of the requests served within a certain time (ms)
  50%    118
  66%    124
  75%    129
  80%    131
  90%    142
  95%    147
  98%    163
  99%    163
 100%    163 (longest request)


$ ab -n 200 -c 10 -X localhost:8000 http://dk.migrid.org/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking dk.migrid.org [through localhost:8000] (be patient)
Completed 100 requests
Completed 200 requests
Finished 200 requests
```

```
Server Software:        Apache/2.2.9
Server Hostname:        dk.migrid.org
Server Port:            80

Document Path:          /
Document Length:        581 bytes

Concurrency Level:      10
Time taken for tests:   2.618 seconds
Complete requests:      200
Failed requests:        0
Write errors:           0
Total transferred:      182800 bytes
HTML transferred:       116200 bytes
Requests per second:    76.40 [#/sec] (mean)
Time per request:       130.892 [ms] (mean)
Time per request:       13.089 [ms] (mean, across all concurrent requests)
Transfer rate:          68.19 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:    99  128  22.3    124     319
Waiting:       98  127  22.2    123     319
Total:         99  128  22.3    124     319

Percentage of the requests served within a certain time (ms)
  50%    124
  66%    131
  75%    138
  80%    143
  90%    157
  95%    162
  98%    169
  99%    175
 100%    319 (longest request)
```