

# Oblig 1 - Logge inn, bli kjent med filer, prosesser og minne

IN2120 - høst 2023

Velkommen til IN2120! Dette blir en snill første labøvelse, der du først og fremst skal klare å logge deg inn på maskinene vi bruker til labene.

Dette er et kurs der ferdighetsnivået blant studentene spriker veldig: Noen var hackere allerede før de begynte på UiO, mens andre knapt vet hvordan man swiper på et nettbrett. Målet er at alle skal ha mulighet til å henge med, derfor er det mye tekst som er forklaring.

Det er mye tekst her, men det meste kan du trygt hoppe over hvis du skjønner greia. Du er ikke nødt til å pugge hver eneste kommando og tekniske detalj, men du skal forstå hvordan dette fungerer i grove trekk.

Forhåpentligvis blir oppgavene også litt gøyale og interessante for alle! :)

## Oppgave 1

**Logg deg inn på labmaskinen din, ved å følge instruksjonene og bruke innloggingsinfoen du har fått på UiO-mailet din. Det skal komme opp et Windows-skrivebord. På skrivebordet ligger det et ikon for en fil som heter "*flagg.txt*". Dobbelklikker du på denne ser du et navn. Dette navnet er svaret på første oppgave!**

For å få poeng må du fylle inn dette svaret: Åpne en hvilken som helst nettleser og gå inn på:

<https://svar.in2120.uiocloud.no>

Du logger deg inn med UiO-brukernavnet ditt og passordet til labmaskinen din (det du fikk på mail). Trykk deg inn på *Spørsmål 1* under *Oblig 1* og fyll inn navnet du fant i *flagg.txt* på labmaskinen.

Nå skal du få litt erfaring med å se hva en fil egentlig inneholder. Som eksempel er det et siste tegn etter linjeskiftet i *flagg.txt*, som ser ut som bare mellomrom/space (slike tegn er kjent som *whitespace* på fagspråket)... Men Notepad (som du åpnet *flagg.txt* med) viser oss ikke om det faktisk er et vanlig mellomrom eller ikke.

Til denne laben har vi installert **Cygwin**, som gir deg en Linux-kommandolinje i Windows (*verktøysett til Linux heter egentlig GNU, for de som er nøy på det*), fordi Linux (GNU) har mange nyttige verktøy.

Du finner Cygwin på skrivebordet ditt, så dobbeltklikk på *Cygwin64 Terminal* for å få opp en kommandolinje.

I Cygwin navigerer man som i vanlige kommandolinjer både i Windows og Linux: Altså med *cd mappenavn* for å gå inn i en undermappe, eller *cd ..* for å gå opp ett nivå i mappehierarkiet. Til forskjell fra Windows sin *dir* bruker man imidlertid Linux sin *ls* for å liste opp filene og undermappene i mappen man er i. Man kan slenge på en bindestrek bak *ls* for å angi mer nøyaktig hvordan man vil ting skal vises, og for å se alle filene i mappa på en oversiktlig måte (med tilgangsrettigheter, eier osv.) kan man skrive:

***ls -Al***

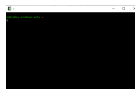


Figure 1: Cygwin sitt kommandovindu

## En liten guide, bare for de som synes kommandolinja virker helt gresk:

```
drwxr-xr-x. 2 omelgsto omelgsto   2048 Aug 17 20:33 'Alle dine hemmeligheter'
-rw-r--r--. 1 omelgsto omelgsto  22360 Aug 10 17:33 Audunsittpassord.png
```

Figure 2: Eksempel på output fra `ls -Al`

Litt forklaring:

- `d` betyr at 'Alle dine hemmeligheter' er en mappe (*directory*)
- `rxw` er tilgangsrettigheter først for eieren, dernest gruppa til eieren, og så for alle andre (*read/write/execute*)
- `omelgsto` er navnet på både brukeren og gruppa som eier filen
- 2048 og 22360 er størrelsen i bytes
- Dato og tid er sist gang filen ble endret
- Filnavn eller mappenavn til sist

Her har du noen eksempler på kommandoer for å navigere i filsystemet:

| Kommando                                       | Windows-kommando                   | Linux-kommando                   | Huskeregul   |
|--|------------------------------------|----------------------------------|--|
| Liste opp innholdet i en mappe                 | <code>dir</code>                   | <code>ls</code>                  | "directory"/"list"   |
| Se alle filer og mapper (også skjulte)         | <code>dir /A</code>                | <code>ls -A</code>               | "All"  |
| Se alle filer som slutter på <code>.txt</code> | <code>dir *.txt</code>             | <code>ls *.txt</code>            | stjerna kan være hva som helst   |
| Se ulike visningsalternativer                  | <code>dir /?</code>                | <code>ls --help</code>           |  |
| Gå inn i en undermappe                         | <code>cd mappenavn</code>          | <code>cd mappenavn</code>        | "change directory"   |
| Gå opp et nivå i mapphierarkiet                | <code>cd ..</code>                 | <code>cd ..</code>               | <i>Ehh.. jeg går ut herfra</i>   |
| Kopiere en fil eller mappe                     | <code>copy filnavn nyttsted</code> | <code>cp filnavn nyttsted</code> | "copy"   |
| Flytt en fil eller mappe                       | <code>move filnavn nyttsted</code> | <code>mv filnavn nyttsted</code> | "move"   |
| Slette en fil (kan ikke angres)                | <code>del filnavn</code>           | <code>rm filnavn</code>          | "delete"/"remove"  |
| Vise hvor i filsystemet du er                  | <code>echo %CD%</code>             | <code>echo \$PWD</code>          | "Current Directory"  |
| Vise en fil (tolkes da som tekst)              | <code>type filnavn</code>          | <code>cat filnavn</code>         | "Present Working Directory"  |
| Lukke kommandolinja                            | <code>exit</code>                  | <code>exit</code>                | "concatenate" = skjøte, fordi<br><code>cat fil1 fil2</code> viser begge. |

Prøv deg gjerne frem litt, så du får en følelse for dette!

Hvis du vil teste ut Windows sin kommandolinje også, kan du skrive inn program-navnet `cmd.exe` i Cygwin (og trykke enter såklart).

Det mest brukte kommandolinje-programmet i Linux heter forøvrig `bash` (og uttales på en morsom måte).

Plasseringen av filer angis nesten likt i Windows og i Linux, men Windows bruker `\` og Linux bruker `/`:

|          |   |           |  |
|----------|---|-----------|--|
| Windows: | <code>rotbokstav:\mappe\undermappe\filnavn.filendelse</code>  | Eksempel: | <code>C:\WINDOWS\SYSTEM32\cmd.exe</code> |
| Linux:   | <code>/mappe/undermappe_som_ofte_angir_filtype/filnavn</code> | Eksempel: | <code>/bin/bash</code>                   |

En annen forskjell å være klar over er at selv om moderne Windows husker på stor og liten bokstav i filnavn og mappenavn, behandler Windows "FilNavn" og "filNAVN" som samme fil, mens for Linux er dette to forskjellige filer. Ellers er det noe som heter at "*i Linux er alt en fil*", slik at Linux gir filnavn til både minneområder, nettverksforbindelser og andre ting, men det får vi ikke bruk for i dette kurset.

Fra kommandolinja kan man starte programmer ved å skrive navnet deres, sende det som kommer ut av et program inn i et annet program, skrive scripts som kjører og vurderer mange kommandoer automatisk, og veldig mye mer! Men i dette kurset skal vi ikke gjøre mer med kommandolinja enn å kjøre enkle kommandoer og finne filer. :)

*Tips: Du gjør det mye enklere for deg selv hvis du venner deg til å bruke tab (knappen over "Caps Lock" på tastaturet) for autocomplete (autofullfør) mens du skriver kommandolinjer.*

*Fun fact: Android er basert på Linux, og macOS er laget for å være som UNIX (som Linux også hermet etter). Så både smarttelefoner og mac-er bruker "linux-kommandoer"!*

Vi skal nå bruke programmet `xxd`, som er et program man finner på så å si alle Linux-maskiner - og som derfor er tilgjengelig inne i Cygwin. Dette programmet lar deg se helt nøyaktig hvilke bits en fil inneholder fra start til slutt - eller motsatt oversette en tekstfil som beskriver nøyaktig hvilke bits en fil skal inneholde, over til en ferdig fil med de bitsene:

Som du kanskje vet er filer på pc-en din satt sammen av *bytes* og hver byte (på moderne maskiner) er satt sammen av 8 *bits* (nuller og enere). Det er veldig tungvindt å lese noe sånt som `01010010 11111111 00001100` osv., så en eller annen luring kom på at hvis vi teller til 16 (ved å bruke bokstavene A-F etter at vi har gått forbi 9) går det akkurat opp i opp med grupper av fire bits:

$$0000 = 0, 0001 = 1, 0010 = 2, 0011 = 3, 0100 = 4, 0101 = 5, \dots, 1001 = 9, 1010 = A, 1011 = B, \dots, 1111 = F$$

Hvis man skal inspisere nøyaktig hva en fil inneholder kikker man altså på byte-verdiene, men istedenfor å tittle på noe sånt som `10100011 01010010` leser man de samme dataene som `A3 52`. Dette 16-talls-systemet er det som heter **hexadecimal**, eller bare *hex* i kortform.

`xxd` (i likhet med alle såkalte *hex editors*) lar deg se nøyaktig hvilke bytes som er i en hvilken som helst fil - representert som hex-verdier. Nærmere bestemt viser `xxd` deg hvilken posisjon (offset) den har kommet til i fila lengst venstre, dernest en liste med byteverdier fra den posisjonen i fila og utover (skrevet som hex) og lengst til høyre de samme byte-verdiene tolket som vanlig tekst. Altså, på hver linje:

<posisjon> <byte-verdiene-der-som-hex> <byte-verdiene-der-som-tekst>

Figure 3: Screenshot av `xxd`, brukt i en Linux-kommandolinje

**Disse hex-verdiene er ikke så kryptiske hvis man skjønner systemet:**

0a (altså en byte som består av bit-verdiene 00001010) er byte-verdien som brukes for å indikere linjeskift i rene tekstfiler (kjent som *line feed*)  
20 (altså 00100000) er vanlig mellomrom  
21 (altså 00100001) er utropstegn  
2e (altså 00101110) er punktum  
48 (altså 01001000) er stor "H"  
61 (altså 01100001) er liten "a"  
osv..

Så "HaHa" ville vært 4861 4861.

Ved å bruke hex, kan man altså se *nøyaktig* hvilke byte-verdier man har å gjøre med, ned til minste lille detalj, uten at noe blir gjemt bort.

Selvfølgelig kan bytes også tolkes som at de representerer andre ting enn tekst, for eksempel lyd, bilder eller en musebevegelse, og i så fall gir ikke bokstav-visningen mening, men hex-visningen er fortsatt presis. Senere i kurset vil du se nærmere på dataoverføring og kommunikasjon på Internet, og da vil du se nøyaktig hvilke bytes som sendes eller mottas ved å se på hex-verdiene.

3

For de som vil se hele oversikten:

Den mest etablerte standarden for hvilke tegn de ulike verdiene skal tolkes som, heter ASCII. ASCII ble laget på en tid da det var naturlig å bruke de første 32 verdiene til å styre tekstpekeren, plinge en bjelle eller liknende, nesten som på en skrivemaskin. I vår tid bruker man som regel Unicode, som er en tabell med mange flere tegn og som bruker opp til 4 bytes for å angi hvert tegn, men de første 127 tegnene i Unicode (nærmere bestemt UTF-8) er direkte kopiert fra ASCII, slik at ASCII fortsatt skal fungere på de aller nyeste maskinene. Dermed er det også ASCII hex-editorer som xxd bruker for å vise deg hvilket tegn byte-verdien tilsvarer.

# ASCII TABLE

| Decimal | Hex | Char                   | Decimal | Hex | Char    | Decimal | Hex | Char | Decimal | Hex | Char  |
|---------|-----|------------------------|---------|-----|---------|---------|-----|------|---------|-----|-------|
| 0       | 0   | [NULL]                 | 32      | 20  | [SPACE] | 64      | 40  | @    | 96      | 60  | `     |
| 1       | 1   | [START OF HEADING]     | 33      | 21  | !       | 65      | 41  | A    | 97      | 61  | a     |
| 2       | 2   | [START OF TEXT]        | 34      | 22  | "       | 66      | 42  | B    | 98      | 62  | b     |
| 3       | 3   | [END OF TEXT]          | 35      | 23  | #       | 67      | 43  | C    | 99      | 63  | c     |
| 4       | 4   | [END OF TRANSMISSION]  | 36      | 24  | \$      | 68      | 44  | D    | 100     | 64  | d     |
| 5       | 5   | [ENQUIRY]              | 37      | 25  | %       | 69      | 45  | E    | 101     | 65  | e     |
| 6       | 6   | [ACKNOWLEDGE]          | 38      | 26  | &       | 70      | 46  | F    | 102     | 66  | f     |
| 7       | 7   | [BELL]                 | 39      | 27  | '       | 71      | 47  | G    | 103     | 67  | g     |
| 8       | 8   | [BACKSPACE]            | 40      | 28  | (       | 72      | 48  | H    | 104     | 68  | h     |
| 9       | 9   | [HORIZONTAL TAB]       | 41      | 29  | )       | 73      | 49  | I    | 105     | 69  | i     |
| 10      | A   | [LINE FEED]            | 42      | 2A  | *       | 74      | 4A  | J    | 106     | 6A  | j     |
| 11      | B   | [VERTICAL TAB]         | 43      | 2B  | +       | 75      | 4B  | K    | 107     | 6B  | k     |
| 12      | C   | [FORM FEED]            | 44      | 2C  | ,       | 76      | 4C  | L    | 108     | 6C  | l     |
| 13      | D   | [CARRIAGE RETURN]      | 45      | 2D  | -       | 77      | 4D  | M    | 109     | 6D  | m     |
| 14      | E   | [SHIFT OUT]            | 46      | 2E  | .       | 78      | 4E  | N    | 110     | 6E  | n     |
| 15      | F   | [SHIFT IN]             | 47      | 2F  | /       | 79      | 4F  | O    | 111     | 6F  | o     |
| 16      | 10  | [DATA LINK ESCAPE]     | 48      | 30  | 0       | 80      | 50  | P    | 112     | 70  | p     |
| 17      | 11  | [DEVICE CONTROL 1]     | 49      | 31  | 1       | 81      | 51  | Q    | 113     | 71  | q     |
| 18      | 12  | [DEVICE CONTROL 2]     | 50      | 32  | 2       | 82      | 52  | R    | 114     | 72  | r     |
| 19      | 13  | [DEVICE CONTROL 3]     | 51      | 33  | 3       | 83      | 53  | S    | 115     | 73  | s     |
| 20      | 14  | [DEVICE CONTROL 4]     | 52      | 34  | 4       | 84      | 54  | T    | 116     | 74  | t     |
| 21      | 15  | [NEGATIVE ACKNOWLEDGE] | 53      | 35  | 5       | 85      | 55  | U    | 117     | 75  | u     |
| 22      | 16  | [SYNCHRONOUS IDLE]     | 54      | 36  | 6       | 86      | 56  | V    | 118     | 76  | v     |
| 23      | 17  | [END OF TRANS. BLOCK]  | 55      | 37  | 7       | 87      | 57  | W    | 119     | 77  | w     |
| 24      | 18  | [CANCEL]               | 56      | 38  | 8       | 88      | 58  | X    | 120     | 78  | x     |
| 25      | 19  | [END OF MEDIUM]        | 57      | 39  | 9       | 89      | 59  | Y    | 121     | 79  | y     |
| 26      | 1A  | [SUBSTITUTE]           | 58      | 3A  | :       | 90      | 5A  | Z    | 122     | 7A  | z     |
| 27      | 1B  | [ESCAPE]               | 59      | 3B  | ;       | 91      | 5B  | [    | 123     | 7B  | {     |
| 28      | 1C  | [FILE SEPARATOR]       | 60      | 3C  | <       | 92      | 5C  | \    | 124     | 7C  |       |
| 29      | 1D  | [GROUP SEPARATOR]      | 61      | 3D  | =       | 93      | 5D  | ]    | 125     | 7D  | }     |
| 30      | 1E  | [RECORD SEPARATOR]     | 62      | 3E  | >       | 94      | 5E  | ^    | 126     | 7E  | ~     |
| 31      | 1F  | [UNIT SEPARATOR]       | 63      | 3F  | ?       | 95      | 5F  | _    | 127     | 7F  | [DEL] |

Figure 4: ASCII-tabellen (kilde: Wikimedia Commons)

Det kan kanskje være litt forvirrende at 32 og 20 står på samme linje når man skriver byte-verdien som et tall: Derfor skriver man ofte 0x20, der 0x bare betyr at verdien er skrevet i hex og ikke desimal. Alternativt kan man bruke "h" eller "b" på slutten for å si at dette er skrevet i "hexadesimal" eller "binær" (binær er bare nuller og enere). Så:

00001100b = 0Ch = 0x0C = 12  
 00010001b = 11h = 0x11 = 17 (forstår du hvorfor?)

## Oppgave 2

Sjekk at du har en "flagg.txt" i mappa du er i, inne i Cygwin, ved å liste opp innholdet i mappa (se kommando ovenfor). Når du har sett at du er i riktig mappe, skriv inn kommandoen:

```
xxd flagg.txt
```

Hva er tallverdien til det aller siste, "usynlige", tegnet som flagg.txt inneholder, skrevet som hex?

Hint: Hvis du ved en tabbe lagret endringer i "flagg.txt" tidligere i laben, slik at den er forandret på, finner du en backup som "flagg.bak". :)

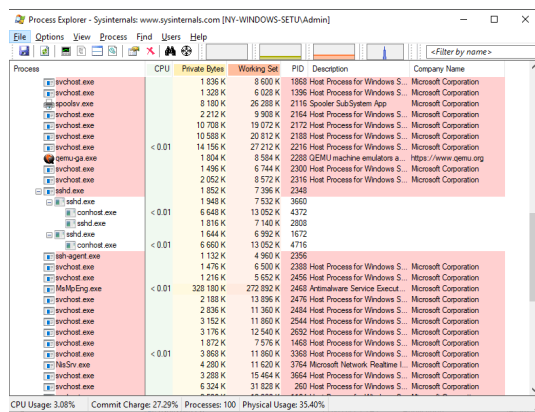


Figure 5: Process Explorer

Det neste du skal gjøre er å få en oversikt over programmene som kjører på maskinen:

Titt på Skrivebordet (bakgrunnen) og klikk deg inn i mappa "Sysinternals Suite". Deretter dobbeltklikker du for å kjøre programmet "**procexp64.exe**". Dette er et kraftig program for å se hva Windows-programmer som kjører på maskinen gjør.

Du skal få opp et slags hierarki over programmer som kjører (med innrykk mot høyre). Dersom du har endret på noe slik at de bare listes opp i en lang liste rett under hverandre kan du trykke "ctrl+t" eller velge "View -> Tree View" i menyen, for å få tilbake innrykket. Innrykket viser deg hvilke programmer som har startet hvilke andre programmer. Generelt er programmet ovenfor og til venstre det programmet som startet et annet program. På fagspråket kaller man dette ofte "morprosess" ("parent process") og "datterprosess" ("child process").

## Oppgave 3

**Hva er filnavnet til programmet som startet services.exe?**

*For spesielt interesserte: services.exe - også kjent som Service Control Manager - styrer de såkalte tjenesteprosessene i Windows, som kjører i bakgrunnen og tilbyr en eller annen funksjonalitet.*

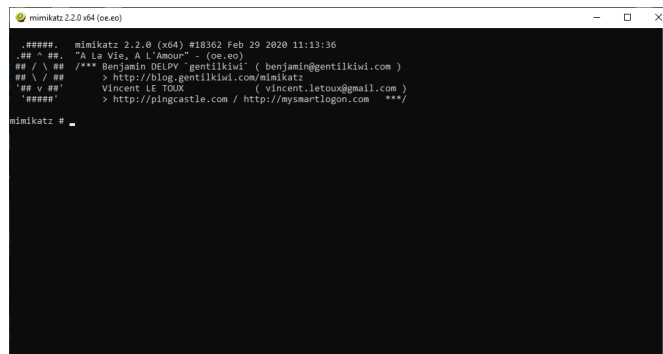
For å se litt nærmere på et program, kan du velge programmet "lsass.exe" med høyreklikk, og deretter velge "Properties" i menyen som spretter opp. (**Ikke** velg "Kill process" på disse grunnprosessene: Da kræsjer alt og du har et problem.)

## Oppgave 4

**Hva står som "Autostart Location:" for prosessen lsass.exe? Det holder at du skriver ordet etter siste "**

*"Autostart Location:" betyr i dette tilfellet hvor det står skrevet at lsass.exe skal startes automatisk når Windows starter.*

Én ting er å se på filer som er lagret et sted, men nå er vi over på å se på ting som kjører i "minnet" til maskinen, så da må vi forstå hva det er:



```
mimikatz 2.2.0 (x64) #18362 Feb 29 2020 11:13:36
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## /** Benjamin DELPY gentilkiwi ( benjamin@gentilkiwi.com )
## \ / ## > http://blog.gentilkiwi.com/mimikatz
'## v ##' Vincent LE TOUX ( vincent.letoux@gmail.com )
##### > http://pingcastle.com / http://mysmartlogon.com ***/

mimikatz # '

```

Alle programmer som Windows, Mac eller Linux kjører, blir lest fra harddisken (eller hvor de nå er lagret) inn i minnet (RAM), og kjøres derfra. Å lese en og en instruksjon fortløpende fra harddisken ville nemlig tatt alt for lang tid! Det finnes mange ulike programmer som lar deg se nøyaktig hva som ligger i minnet til maskinen (på samme måte som hex-editoren xxd viste oss nøyaktig innhold av filer), og da kan man se både hvilke instruksjoner som er i ferd med å utføres, og hva programmene “husker på”. For eksempel må programmer som lar deg skrive inn tekst, holde av et område i minnet til å midlertidig lagre (huske på) hva du har skrevet.

En annen ting som programmer ofte må huske på (for eksempel mens du skriver det inn eller når det skal brukes) er nøkkelverdier og passord...

**mimikatz** er et program som kan lese passord rett ut fra Windows sitt minneområde (og mye annet). Faktisk skal du nå få hente et passord ut av minneområdet til lsass.exe, som er Windows sin login-tjeneste og som du allerede har sett i Process Explorer.

Egentlig ble mimikatz laget for folk som allerede hadde tilgang til å styre maskinen (det som heter “Administrator” på Windows, eller “root” hvis det hadde vært Linux), for å demonstrere hva Windows drev med og hvordan passord var sårbare. Men ganske snart kombinerte folk mimikatz med verktøy som [USAs hemmelige tjenester \(NSA osv.\) hadde laget](#) for å stjele til seg mer tilgang enn man egentlig hadde. Derfor brukes nå mimikatz både av folk som vil bryte seg inn, og av [automatisert skadevare](#). Men slapp av: Her bruker vi programmet i sin opprinnelige form!

mimikatz.exe skal ligge synlig på skrivebordet på labmaskinen din. Windows regner med at du ikke vet ditt eget beste, og kan finne på å slette filer enda du ber Windows holde fingra av datet, så det kan hende Windows har slettet filen når du leser dette. **Hvis mimikatz.exe ikke ligger på skrivebordet lenger, kan du trykke på lenken for “Hent mimikatz på nytt”**. Hvis Windows har slettet *lenken* også, kan du hente mimikatz [her](#) (åpne lenken på labmaskinen din i så fall, og lagre fila et sted hvor du finner den).

Start mimikatz med administrator-rettigheter, ved å høyreklikke på mimikatz.exe og velge “Run as administrator”. Nei, du vil ikke søke i Microsoft sin “Store”, du kan fjerne haka for å “spørre deg hver gang” og ja, du vil la programmet gjøre endringer.

Du skal få opp en ny type kommandolinje, for å skrive kommandoer til mimikatz denne gangen. Skriv først inn en linje for å gi deg selv privilegier som om du var en utvikler som skulle fjerne bugs:

```
privilege::debug
```

mimikatz skal svare **Privilege 20 OK**. Hvis det står noe annet, har du mest sannsynlig glemte å kjøre programmet som administrator.

Nå vet du tilfeldigvis at det er en bruker som heter Raymond som pleier å være logget inn på samme maskin, men han er ikke pålogget akkurat nå... Du kan tenke deg at du har brukt et program i Sysinternals Suite tidligere, mens Raymond var pålogget, til å lagre en “dump” av hva som var i minnet til lsass.exe, for å kunne se nærmere på det senere. Den nøyaktige kommandoen som er brukt er “**procdump.exe -ma lsass.exe lsass.dmp**” (*dette trenger du ikke skrive, det er gjort allerede*).

Dette er en ganske vanlig fremgangsmåte: Hvis man for eksempel ikke får mimikatz til å kjøre, kan man ta en kopi av hva som er i minnet, sende det til seg selv og bruke mimikatz på sin egen maskin.

Nå skal vi be mimikatz om å kikke på denne minnedump-filen, som vi har plassert på `C:\lsass.dmp`. Skriv inn:

```
sekurlsa::minidump C:\lsass.dmp
```

mimikatz skal fortelle deg at den åpner filen. Nå kan du bruke “sekurlsa”-modulen igjen, til å lese ut noe nyttig informasjon fra lsass.exe:

```
sekurlsa::wdigest
```

## Oppgave 5

Bla oppover i det mimikatz spyttet ut, så skal du finne en bruker ved navn “Raymond”, som har logget seg inn på samme labmaskin som deg. Hva er passordet hans?

*Det er mulig du tenker du finner et nytt administrator-passord i samme slengen, men dette har noen heldigvis vært lur nok til å endre etter at dump-filen ble lagret... :P*