IN2120 – Informasjonssikkerhet

Del 2: Systemsikkerhet

Gudmund Grov

Høst 2023
Universitetet i Oslo

Oversikt

- Systemarkitektur og overordnet systemsikkerhet
- Prosesser og minnet – buffer overflow
- Privilegienivå
- Virtualiseringsarkitekturer
- Sikker oppstart – UEFI og TPM
- Side-kanaler og skjulte kanaler

Viktigheten av systemsikkerhet

- *«Å bruke kryptering på internett tilsvarer å bruke en pansret bil for å levere kredittkortinformasjon fra en som bor i en pappkartong til en som bor på en parkbenk.»*

  Eugene Spafford

- Moralen er at god kommunikasjonssikkerhet er bortkastet hvis systemsikkerheten er svak.

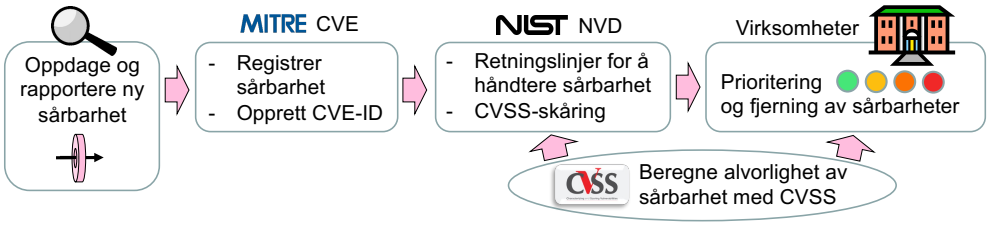
Tilnærminger for å styrke systemsikkerheten

- Fjern feil og sårbarheter i operativsystemet 
  - Nye versjoner og sikkerhetsoppdateringer
- Legg til sikkerhetsfunksjoner i OS og CPU (Central Processing Unit) 
  - Privilegienivåer, NX (No eXecute), ASLR (Address Space Layout Randomization)
- Monitorering av systemsikkerhet 
  - Antivirusverktøy
  - Brannmur, inntrengingsdeteksjon
- Virtualiseringsteknologi 
  - Beskytte prosesser ved å separere virtuelle maskiner
- Tiltrodd beregning, f.eks. 
  - Sikker oppstart med UEFI
  - Sikker maskinvare på plattformen, f.eks. TPM (Trusted Platform Module) 

Kontinuerlig sikkerhets-oppdatering

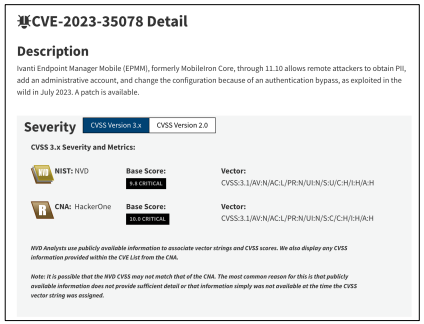
- Nye sårbarheter oppdages kontinuerlig
- Krever kontinuerlig online sikkerhetsoppdatering
- En nulldags- (zero-day) sårbarhet er kun kjent for angriper, slik at programvareprodusenten har hatt null dager (zero days) til å fjerne sårbarheten
- Sikkerhetsoppdatering kan ta tid, slik at angripere ofte utnytter kjente sårbarheter

CVE (Common Vulnerability and Exposures) CVSS (Common Vulnerability Scoring System)



- CVSS (Common Vulnerability Scoring System) beregner alvorlighet av en sårbarhet fra 0 til 10. Virksomheter prioriterer fjerning/mitigering av sårbarheter ut ifra alvorlighet.
  - Kritisk: 9,0–10
  - Høy: 7,0–8,9
  - Middels: 4,0–6,9
  - Lav: 0,1–3,9

Eksempel på CVE: CVE-2023-35078


CVE-2023-35078 Detail

Description

Ivanti Endpoint Manager Mobile (EPMM), formerly MobileIron Core, through 11.10 allows remote attackers to obtain PII, add an administrative account, and change the configuration because of an authentication bypass, as exploited in the wild in July 2023. A patch is available.

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD Base Score: 9.8 CRITICAL Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

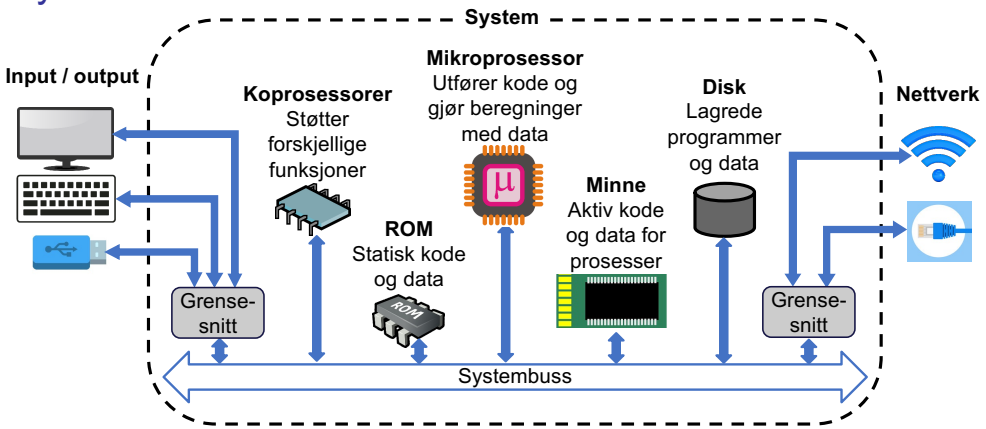
CNA: HackerOne Base Score: 10.0 CRITICAL Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: It is possible that the NVD CVSS may not match that of the CNA. The most common reason for this is that publicly available information does not provide sufficient detail or that information simply was not available at the time the CVSS vector string was assigned.

[https://nvd.nist.gov/vuln/detail/CVE-2023-35078]

Systemarkitektur



Kjørbare filer

- For å forstå noen systemsikkerhetstiltak må vi først ha en basisforståelse av hvordan programmer/filer kjøres
  - (både «godartede» programmer og skadevare).
- En kjørbar fil består av «dead bytes»
  - Det vil ofte være en binærfil (hvis kompilert)
- En kjørbar fil inneholder instruksjoner som vill be utført av CPU når filen kjøres
  - Slike filer må følge et gitt format (f. eks Portable Executable (PE) som brukes i Windows)
- Det finnes mange verktøy som lar deg inspisere ulike aspekter av (kjørbare) filer:
  - strings, winhex, PEview, IDA, ..

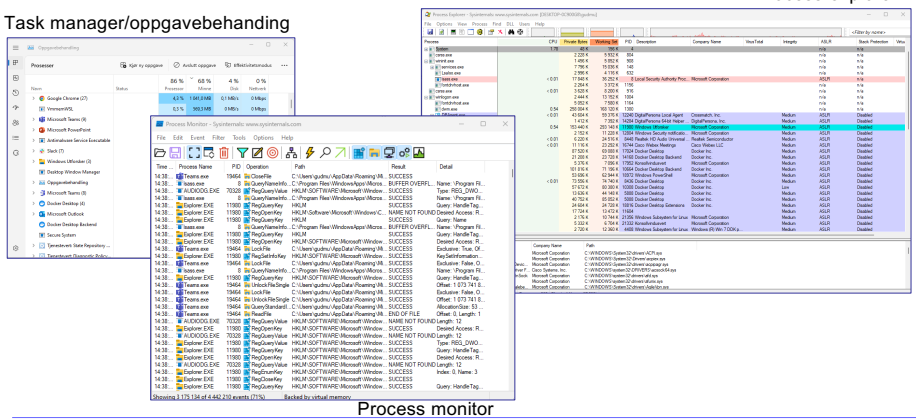
Prosesser

- Når et program eller en bruker starter en (kjørbar) fil opprettes en ny prosess.
  - en del av minnet allokeres til prosessen
  - relevante deler lastes inn i minnet
  - og instruksjonene i filen kjøres i CPU.
- Man kan se litt informasjon om prosesser som kjører i task manager / oppgavebehanding
- For Windows (10/11) kan man også laste ned *Sysinteral suite*.
  - Inneholder et sett med verktøy for å overvåke ulike egenskaper av prosesser, f.eks.:
    - Process Explorer gir informasjon om prosesser som kjører (mer detaljer enn task manager)
    - Process Monitor gir informasjon om hvordan ulike prosesser kommuniserer med systemet.

Overvåking av prosesser – eksempel verktøy

Task manager/oppgavebehanding

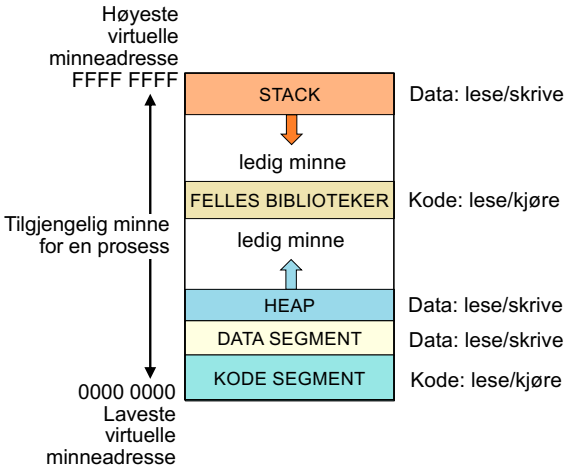
Process-explorer



Process monitor

Virtuelt minne for en prosess

- Hver prosess har et eget virtuelt minneområde.
- Virtuelle minneadresser oversettes av OS til fysiske minneadresser før de leses og skrives i system-minnet.
- Selv om alle prosessene har samme virtuelle adresseområde, har de i virkeligheten separate fysiske adresseområder.
- Dette prinsippet gjør at en prosess ikke har tilgang til fysiske adresser for andre prosesser, kun til sitt eget fysiske adresseområde, indirekte gjennom oversettelsen fra virtuelle til fysiske minneadresser.



Buffer overflow

- Buffer overflow er en klassisk sårbarhet, f.eks
  - Morris-ormen (en av de første dataormene på nett; 1988)
  - SQL-slammer (infiserte ca. 75 000 Microsoft SQL-tjenere på omtrent 10 minutter; 2003)
- Går i prinsippet ut på at minnet overskrives og blir korrupt
  - Dette får angriper til å kjøre egen kode
- Disse vil da kjøres med samme privileger som det opprinnelige programmet
  - Er sårbarheten i et root/admin program vil angriper kunne kjøre sin kode med root/admin-privelegiene
- Det er flere måter å oppnå en buffer overflow
  - vi vil fokusere på en stack-basert versjon

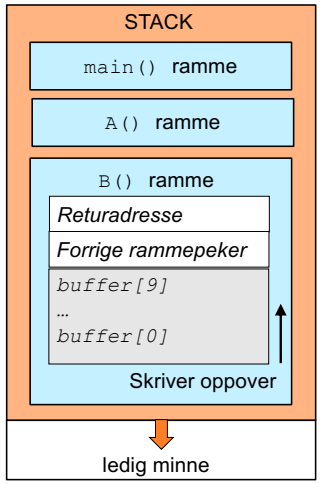
Buffer overflow

- Hvert funksjon/metode-kall har en egen ramme på stacken
  - Inneholder argumenter, lokale variabler, +++
  - rammeregister som inneholder adressen til hvor rammen begynner
  - **returadresse** til neste instruksjon når funksjon terminerer
- Eksempel:

```
main(){
  A();
}
```

```
A(){
  B();
}
```

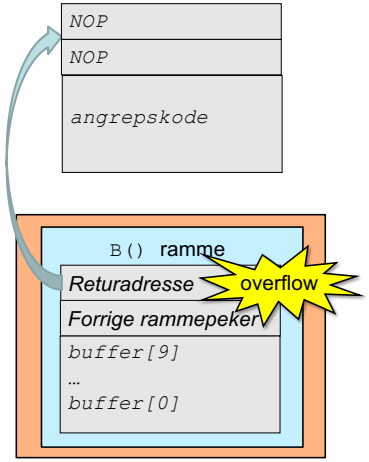
```
B(){
  char buffer[10];
  buffer = read();
  …
}
```



Buffer overflow

```
B(){
  char buffer[10];
  buffer = read();
  …
}
```

- I sårbarheten fylles `buffer` med mer enn allokert minne
- Som følge kan returadresse overskrides
- Målet er å få den til å peke på en plass i minnet hvor angrepskode ligger
  - Denne vil da kjøre istedenfor kode som opprinnelig lå i returadressen
- Man må vite nøyaktig hvor koden ligger
- Løses ofte med et sett `NOP`-instruksjoner (som ikke gjør noe) før angrepskoden
  - Da holder det å treffe adressen på en `NOP`-instruksjon



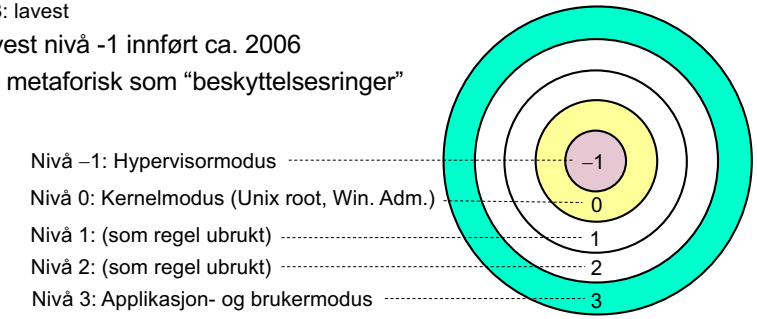
Buffer overflow – mottiltak

- No Execute (NX): OS tillater ikke å kjøre kode på stacken
- Stack canaries: tilfeldig verdi lagret annet sted som sjekker om data har blitt skrevet over
- ASLR (Address Space Layout Randomization): angriper må vite hvor i minnet angrepskoden som skal kjøre ligger. ASLR gjør det vanskeligere å finne denne adressen

- Bruk av (sikrere) programmeringsspråk og statisk analyse
- Skriv bedre kode (innebygd sikkerhet)

OS privilegienivåer

- Hierarkiske privilegienivåer ble introdusert i X86 CPU arkitekturen til Intel (og AMD) i 1985 (Intel 80386), med 4 nivåer
  - Nivå 0: høyest
  - Nivå 3: lavest
- Nytt høyest nivå -1 innført ca. 2006
- Illustrert metaforisk som "beskyttelsesringer"



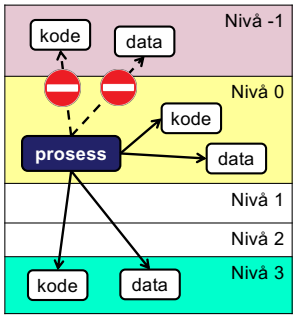
Hva skjedde med nivå 1 og 2 ?

- Det viste seg etterhvert at den hierarkiske sikkerheten som de fire beskyttelsesringene ga, ikke samsvarte særlig godt med behovene til systemprogrammereren, og at det ga liten eller ingen forbedring i forhold til den enkle sikkerhetsmodellen med bare to nivåer. Beskyttelsesringer kunne implementeres effektiv i maskinvare, men det var lite annet å si om dem.
- Å implementere operativsystemer med mange sikkerhetsnivåer viste seg å være en blindgate.

Maurice Wilkes (1994)

Prinsipp for bruk av privilegienivåene

- En prosess kan få tilgang til, og endre data og programvare på samme eller lavere privilegie-nivå som seg selv
  - (samme eller høyere tallverdi).
- En prosess som kjører i kernelmodus (nivå 0) kan få tilgang til data og SW på nivå 0, 1, 2 og 3
  - men ikke på nivå -1
- Målet til angriper er å få tilgang til kernel (nivå 0) eller hypervisormodus (nivå -1), f.eks.
  - gjennom exploits
  - ved å lure brukerne til å installere skadevare

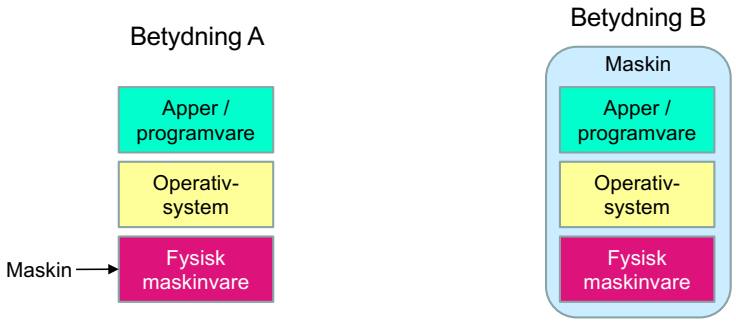


Eksempel

Virtualisering

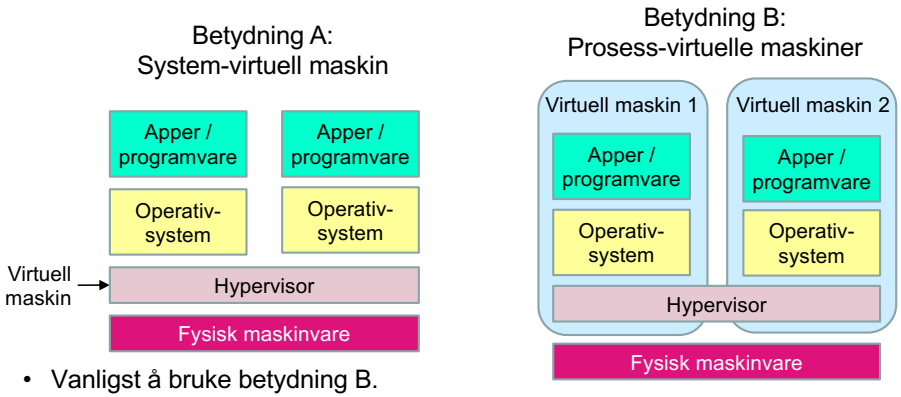
- Systemsikkerhet kan økes ved økt/bedre separasjon av prossessene gjennom virtualisering
- En virtuell maskin er programvare som spiller rollen som fysisk maskinvare
  - kalles en *hypervisor*.
- Eksempler på bruk av virtualisering
  - Skyleverandører driver store serverparker
    - Hver kunde får sin egen virtuelle maskin (VM)
    - Mange kunder deler den samme maskinvaren
    - Lett å migrere VM mellom servere for å øke/redusere kapasiteten
  - Testing og programvareanalyse
    - Potensielt skadelige eksperimenter kan trygt utføres i isolerte omgivelser
    - Ta et snapshot av den nåværende tilstanden til operativsystemet
    - Systemet kan til enhver tid settes tilbake til snapshot-tilstanden
    - Analyse av skadevare

Ulike betydninger av begrepet “maskin”



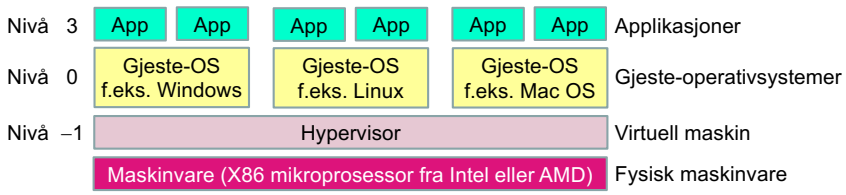
- Maskinvare eller datamaskin.
- Betydning tolkes fra kontekst, eller forklares eksplisitt.

Ulike betydninger av begrepet “virtuell maskin”



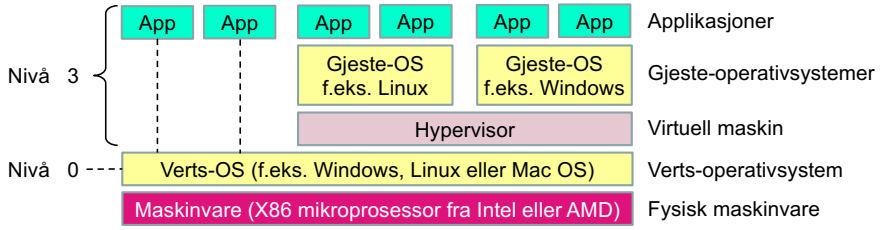
- Vanligst å bruke betydning B.
- Betydning tolkes fra kontekst, eller forklares eksplisitt.

Type 1 virtualisering (native)



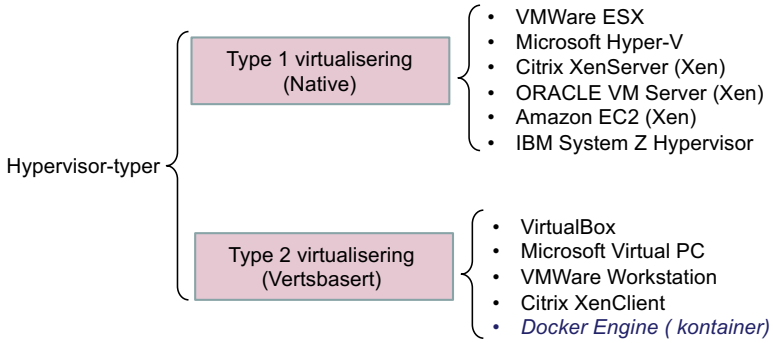
- Type 1 virtualisering er den mest direkte og mest effektive måten å kjøre en virtuell maskin på.
- Type 1 virtualisering gir en logisk struktur på privilegienivåene ved at hypervisoren (den virtuelle maskinen) er mer privilegert enn gjeste-operativsystemene som den styrer.
- Gjeste-operativsystemene kjører med privilegienivå 0, som de nettopp er designet for.

Type 2 virtualisering (vertsbasert)

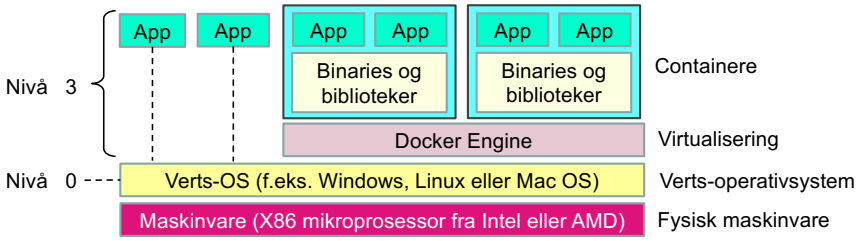


- Hypervisor-programmet blir installert som en vanlig applikasjon på vertsoperativsystemet.
- Enkelt for vanlige brukere å installere og bruke.
- Nivå -1 brukes ikke: hypervisor og gjeste-OS kjører som vanlige applikasjoner på nivå 3.
- Operativsystemer trenger å utføre privilegerte funksjoner som krever nivå 0 (eller nivå -1).
- Privilegerte funksjoner må kalles gjennom verts-operativsystemet, noe som forårsaker ekstra forsinkelse i prosesseringen. Derfor er type 2 en relativt ineffektiv virtualiseringsarkitektur.

Produkter for virtualisering



Docker Engine og containere (vertsbasert)



- Docker Engine gjør det mulig for containeriserte applikasjoner å kjøre uansett plattform.
- Docker Engine er en form for virtualisering som gjør gjeste-OS unødvendig.
- Eliminerer avhengigheter på OS-plattformer, fordi avhengighetene er innebygd i containeren.
- En container er en programvareenhet med kode og alle dens avhengigheter, slik at applikasjonen kjører raskt og pålitelig, og kan lett porteres mellom ulike plattformer som har Docker Engine.

Virtualisering og sikkerhetsmål

- Gjeste-(operativ)systemene må ikke kunne aksessere eller bli påvirket av hverandre
- Gjeste-(operativ)systemene må ikke kunne påvirke hypervisor-programmet
- Gjeste-(operativ)systemene må ikke kunne detektere at de er virtualisert

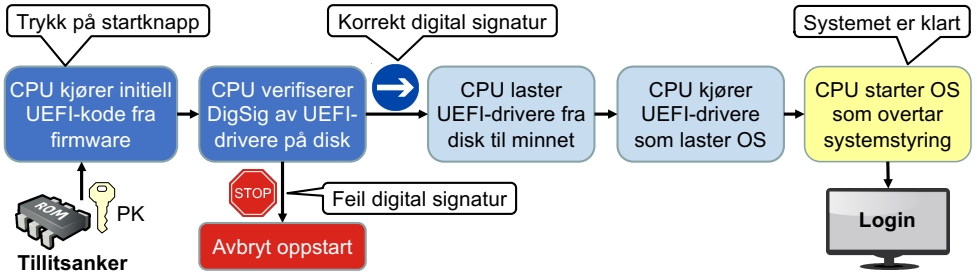
Oppstart av systemet

- Windows 10/11 (og andre moderne OS) har mange sikkerhetstiltak:
  - F.eks. virtuelt minne, OS-privilegier, Windows Defender / antivirius, tilgangskontroll, …
  - … men disse beskytter bare etter OS har startet
  - … og moderne skadevare (f.eks. rootkits / bootkits) kan bli startet **før** OS
- Tradisjonelt ble oppstart av en computer håndtert av BIOS:
  - Starter viktige maskinvarekomponenter som motherboard, CPU og minne
  - Laster OS og starter det
- BIOS gjør noen integritetssjekker men har en del sårbarheter som kan utnyttes av skadevare.
  - F.eks. BIOS vet ikke om det er et tiltrodd operativsystem eller skadevare som startes
- UEFI har erstattet BIOS i moderne computere for å mitigere slike sårbarheter.
- Kombinasjon av UEFI og TPM 2.0 brukes i nyere versjoner av Windows (10/11) for å oppnå sikrere og mer tiltrodd oppstart

UEFI

- Unified Extensible Firmware Interface (UEFI) erstatter BIOS i moderne computere.
- Det er en spesifikasjon som definer grensnitt mellom operativsystemet og maskinvaren (firmware til maskinvare)
- Intel utviklet den originale Extensible Firmware Interface (EFI)
  - UEFI erstattet EFI i 2005.
- UEFI addresser flere begrensinger med BIOS, inkludert sikkerhetsaspekter

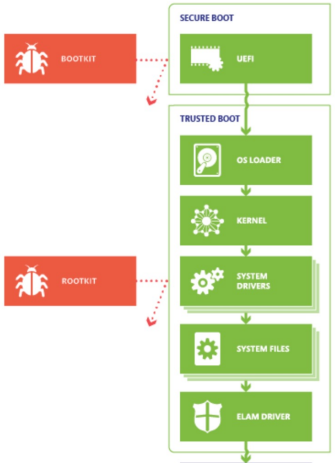
Sikker oppstart med UEFI



- Programmoduler for oppstart er digitalt signert av computerleverandøren.
- UEFI-kode i ROM er usignert men antas å være korrekt, og initierer oppstartsekvensen.
- Programmoduler som lastes sjekkes for korrekt digital signatur med Platform Key.
- Hvis en feil signatur detekteres vil oppstartsekvensen avbrytes.
- Når UEFI er ferdig fortsetter Windows 10/11 tillitskjeden ved å verifisere hver komponent før den lastes (Microsoft kaller denne fasen *tiltrodd oppstart)*

Tiltrodd oppstart

- Tar over etter UEFI / sikker oppstart
- Windows sjekker integritet av hver komponent før de lastes
  - Bootlader verifiserer først digital signatur av Windowskjernen før den lastes
  - Windowskjernen verifiserer deretter hver component før de starter
- Hvis en feil finnes kjøres ikke den komponenten
  - I noen tilfeller kan Windows automatisk reparere feilen og dermed kjøre komponenten



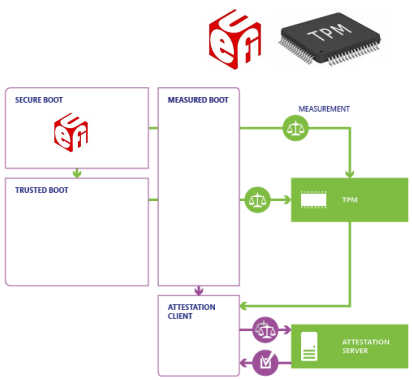
[kilde: https://docs.microsoft.com/en-us/windows/security/information-protection/secure-the-windows-10-boot-process]

TPM

- *TPM (Trusted Platform Module)* er en koprosessor bygget inn i mange ulike systemer
  - teknisk sett både en spesifikasjon og dens implementasjon
- Kan brukes til å utføre noen kryptografiske operasjoner i et beskyttet miljø
- Passiv: mottar kommandoer og returner svar
- Ofte innbygget (i motherboard) på maskiner
- Windows 11 «krever» TPM 2.0
- Mål er å øke tiltro i en plattform/system/computer

Målt oppstart (*measured boot*)

- UEFI og OS lagrer hash av «komponenter» som lastes og lagrer den i TPM
  - Gjøres steg for steg
  - bruk av TPM påser at hash i en fase ikke kan endre hash av tidligere faser
  - Gir en tillitskjede
- Mot slutt av oppstarten
  - Signerer TPM logg
  - Windows starter en attesteringsklient
  - Klient verifiserer loggen
    - og potensiell annen sikkerhetsinformasjon
  - Bruker en ekstern attesteringstjener for å verifisere/sjekke sikkerhetstilstanden
- Mitigerende tiltak gjøres basert på denne sjekken



[kilde: https://docs.microsoft.com/en-us/windows/security/information-protection/secure-the-windows-10-boot-process]

Sikker (og tiltrodd) oppstart vs målt oppstart

- Sikker og tiltrodd oppstart
  - Tillitskjede basert på sertifikat lagret i firmware
  - Neste steg starter bare hvis korrekt signatur
    - Enheten starter dermed ikke hvis verifisering feiler
  - Nye komponenter/oppdatering må signeres og sertifikat håndteres
- Målt oppstart
  - Tillitskjede lagret i TPM
  - Verifiseres etter at stegene i oppstart er ferdige
  - En uavhengig tredjepart kan utføre verifisering (ekstern attestering)

Annet bruk av TPM i Windows 10/11

- Vi har illustrert bruk av TPM for sikker/tiltrodd/målt oppstart og ekstern attestering
  - TPM har også andre bruksområder i Windows
- Krypto
  - Beskyttelse av kryptonøkler (nøkler blir aldri lastet inn i minnet og/eller forblir i TPM)
  - Beskyttelse mot ordbokangrep (angrep som gjetter et stort antall PIN/passord)
- Virtuelle smartkort
  - Brukes f.eks. for multifaktorautentisering (mer om det senere i kurset)
- Bitlocker / enhetskryptering
  - Krypterer data på disk
  - Bruker logg fra målt oppstart
    - Frigir bare nøkler for dekryptering når målingen har spesifikke/ønskede verdier

Sidekanaler og skjulte kanaler

- En ***sidekanal*** er en utilsiktet kanal som avgir informasjon som skyldes den fysiske implementeringen av et system
- Eksempel på sidekanaler er: tidsforbruk for instruksjon i CPU kan si noe om verdi (f.eks. av enkelte bits i krypteringsnøkkel), strømforbruk, lyd, stråling
- En ***skjult kanal*** *er en* mekanisme som ikke er designet for kommunikasjon, men som misbrukes (med vilje) for å overføre informasjon (på en måte som bryter med sikkerhetspolicy)
- Meltdown og Spectre er eksempler på sidekanaler i mikroprosessorer
  - Moderne prosessorer «gjetter» (mulige) neste instruksjoner (selv om de kan bryte sikkerhetspolicyer), og data lever fortsatt i cache selv om de ikke velges
  - Sårbarheten kan medføre at andre prosesser kan lese en prosess data mens den kjører

Oppsummering

- Overordnet om systemarkitektur og tilnærminger for systemsikkerhet.
- Virtuelt minne for hver prosess
  - Litt om prosesser og hvordan minnet brukes – virtuelt minne
  - Et type angrep kalt buffer-overflow
- Privilegienivå
- Sikker oppstart (UEFI og TPM)
- Virtualisering og virtualiseringsarkitekturer
- Side-kanaler og skjulte kanaler

- Andre tema som er viktig for å oppnå god systemsikkerhet vil dekkes senere i kurset (f.eks. tilgangskontroll og brannmurer)

Slutt på presentasjonen