

Sikker applikasjonsutvikling

”Building Secure Software”

Kapittel 8 – Access Control

Kapittel 9 – Race Conditions

Mikal Carlsen Østensen – mikal@ii.uib.no

Institutt for Informatikk
Universitet i Bergen

Onsdag 28. September 2005

INF 329 – Utvalgte emner i programutviklingsteknologi:

Utvikling av sikre applikasjoner



Kapittel 8

Access Control

Side 187 – 208

Access Control

Intro, Del 1

- En bruker blir autentisert av et system.
 - Systemet må da avgjøre hvilke ressurser som brukeren kan få tilgang til.
- Det finnes en rekke ulike AC modeller.
- AC systemer baserer seg ofte på kompleks matematikk og kan være vanskelig å benytte.

Access Control

Intro, Del 2

- De mest brukte AC modellene
 - UNIX AC system
 - Windows modellen som baserer seg på Access Control Lister (ACL)
- UNIX modellen har mange varianter.
- I nyere Linux distribusjoner finnes det støtte for filsystemer med ACL.

Access Control

The UNIX Access Control Model

- Hver bruker representert som en integer.
 - **User ID (UID)**
- Brukere kan også tilhøre "grupper".
 - Virtuelle samlinger
 - Tillater brukere å samarbeide på prosjekter.
 - Brukere kan tilhøre flere grupper samtidig.
 - **Group ID (GID)**

Access Control

The UNIX Access Control Model

- Administratoren har UID 0.
 - Som oftest rot kontoen på maskinen.
- Admin eller brukere som låner UID 0 privilegiene, har fullstendig tilgang og kontroll over hele maskinen.

Access Control

The UNIX Access Control Model

- Alle objekter i et system får tildelt sin egen UID og GID, inkludert:
 - Filer
 - Enheter
 - Kataloger
- I tillegg har hver av disse, følgende tilgangs tillatelser assosiert med seg:
 - Hvem som har tilgang til å **lese** objektet. **(r)**
 - Hvem som har tilgang til å **skrive** til objektet. **(w)**
 - Hvem som har tilgang til å **kjøre** objektet. **(x)**

Access Control

The UNIX Access Control Model

- Det finnes tre slike **rwX** sett med aksess tillatelser per fil.
 - Brukeren som eiere filen.
 - Enhver bruker som tilhører en gruppe som eier filen.
 - Alle andre brukere av systemet.

Access Control

The UNIX Access Control Model

- Et program som kjøres tildeles brukerens UID.
 - Vanligvis gjelder dette også barneprosesser av programmet.

Access Control

The UNIX Access Control Model

- Objekter blir alltid aksessert fra en kjørende prosess.
- Ved et slik aksess forsøk vil OS se på noen gitte verdier som er tilegnet den kjørende prosessen.
 - Den effektive UID (EUID)
 - Den effektive GID (EGID)
- Disse verdiene blir sammenlignet med tillatelsene som trengs for å gjennomføre aksessen.

Access Control

The UNIX Access Control Model

- Vanligvis er EUID den samme som den virkelige UID til en prosess.
- Unntaket er programmer som trenger tilgang til spesielle system ressurser. EUID og EGID endres da med følgende type program:
 - setuid program**
 - setgid program**

Access Control

The UNIX Access Control Model

- Sikkerhets risikoen til et setuid program.
 - Enhver bruker som kan kjøre et slik program kan få tillatelse til å gjøre akkurat det samme som eieren av programmet.
 - Dette vil si eierskap som gir programmet nødvendige system ressurser, ofte UID 0.
 - Dersom det er en feil i setuid programmet, kan en angriper dermed få full tilgang til maskinen.

Access Control

The UNIX Access Control Model

- Et vanlig angrep.
 - Angriperen bryter seg inn via en generisk bruker konto.
 - Benytter da et hull i en nettverks server eller utnytter et dårlig valgt passord.
 - Utnytter så et dårlig skrevet setuid program som er eid av rot brukeren.
 - Tilegner seg ønskede privilegier og kontroll.

Access Control

Hvordan UNIX tillatelser virker

- Det er som sagt tre set med tillatelser assosiert med filer og kataloger.
- Verd å merke seg med filer:
 - Hvis en fil er skrivbar i en gitt gruppe, må en bruker her ha en spesifikk tillatelse for å kunne skrive til filen.
 - Rotbruker kan kun kjøre andre sine filer, hvor det finnes minst en tillatelse for dette i systemet.
 - Rotbruker kan fort være uheldig å gjøre uopprettelig skade.

Access Control

Hvordan UINIX tillatelser virker

- Verd å merke seg med kataloger:
 - Lese rettighetene spesifiserer om man har tillatelse til å se en liste av innholdet.
 - Kjøre tillatelsene spesifiserer om en bruker kan traversere katalogen eller på annen måte bruke filer i den.
 - Skrive tillatelsen spesifiserer om brukeren kan, legge til, endre, eller fjerne en fil.

Access Control

Hvordan UNIX tillatelser virker

- Andre former for tillatelser:
 - Tekst bittet ("Sticky bit")
 - Satt av rotbruker på eldre systemer for unngå at et program skiftet over til disk.
 - Setuid og setgid
 - De har kun portable oppførsel på kjørbare filer.
 - Indikerer om UID, GID, EUID og EGID identifikatorene kan endres av den kjørbare filen.
 - Hvis aktivert vil EUID til en prosess bli satt til fileiers UID.
 - Tilsvarende for EGID og GID.

Access Control

Modifisering av filattributter i UNIX

- UNIX shelllet tilbyr to enkle kommandoer for å endre fil attributter:
 - ***chmod*** setter tillatelser til en fil eller katalog.
 - ***chown*** endrer eierskap til en fil eller katalog.

Access Control

Modifisering av filattributter i UNIX

■ ***chmod*** kommandoen

- Input er enten en spesifikasjon på endringer i tillatelse.
- Eller en spesifikasjon på nye tillatelser.
- Input er også hvilke fil eller mappe dette gjelder for.
- Det finnes i tillegg en rekke opsjoner å velge fra.

Access Control

Modifisering av filattributter i UNIX

■ **chmod** aksess typer:

- u – fil eier
- g – gruppe
- o – andre

■ **chmod** tilgang:

- + indikerer godkjenning.
- indikerer ikke godkjent.
- = indikerer godkjenning, mens resten blir slått av.

■ **chmod** tillatelser:

- r – les
- w – skriv
- x – kjør
- s – setuid eller setgid
- t – tekst tillatelse ("sticky")

Access Control

Modifisering av filattributter i UNIX

■ ***chmod go-rwx ****

- Vil føre til at all filene i en katalog ikke lenger er tilgjengelig for andre enn brukeren.

■ ***chmod u-rwx ****

- Vil medføre at ingen har tilgang til filen, unntaket er rot som kan skrive og lese filen, men ikke kjøre den.

■ ***chmod u=rwx ****

- Lar bruker gjøre hva som helst med filen, mens alle andre brukere har ingen tilgang til filen.

Access Control

Modifisering av filattributter i UNIX

Alternativ oktal notasjon

4 tall fra 0-7 representerer:

1 (MSB): setuid, setgid, sticky

2: Eier tillatelser

3: Gruppe tillatelser

4 (LSB): Andre tillatelser

Tall	Betydning		
0	setuid av	setgid av	sticky av
1	setuid av	setgid av	sticky PÅ
2	setuid av	setgid PÅ	sticky av
3	setuid av	setgid PÅ	sticky PÅ
4	setuid PÅ	setgid av	sticky av
5	setuid PÅ	setgid av	sticky PÅ
6	setuid PÅ	setgid PÅ	sticky av
7	setuid PÅ	setgid PÅ	sticky PÅ
0	les av	skriv av	kjør av
1	les av	skriv av	kjør PÅ
2	les av	skriv PÅ	kjør av
3	les av	skriv PÅ	kjør PÅ
4	les PÅ	skriv av	kjør av
5	les PÅ	skriv av	kjør PÅ
6	les PÅ	skriv PÅ	kjør av
7	les PÅ	skriv PÅ	kjør PÅ

Access Control

Modifisering av filattributter i UNIX

- ***chmod 0000 blah.txt***

- Forbyr all tilgang til blah.txt

- ***chmod 0644 blah.txt***

- Gir bare-lese tilgang til alle andre untatt eieren.

- ***chmod 0755 foo***

- Tillater alle å kjøre og lese foo, passer best til skript.

- ***chmod 0711 foo***

- Tillater alle å kjøre foo, passer best til binære filer.

Access Control

Modifisering av filattributter i UNIX

- Andre eksempler
 - **chmod ug+rwx,o+r,o-wx blah**
 - **ls -l blah**
 - **-rwxrwxr-- brukernavn/UID gruppenavn/ID**
modifikasjonstidspunkt
- Merk her at modifikasjonstidspunkt kan endres til en tilfeldig verdi ved hjelp av touch kommandoen.
 - Brukes ofte av angripere.

Access Control

Modifisering av eierskap i UNIX

- ***chown*** kommandoen
 - Endrer eller spesifiserer bruker og gruppe eierskap til en fil.
 - Vanligvis bare brukt av rot brukeren.
 - Rot tilgang kreves vanligvis for å endre fil eierskap.
 - Gruppe eierskap kan kun endres til en gruppe som brukeren er en del av allerede.

Access Control

Modifisering av eierskap i UNIX

- Brukere tilhører alltid en hovedgruppe.
 - Denne gruppen er viktig for setgid og nylig opprettede filer.
- ***newgrp* <ny hovedgruppe>**
 - Endrer hovedgruppe

Access Control

Modifisering av eierskap i UNIX

■ ***chown* fred blah.txt**

- blah.txt endrer eierskap til fred.
- Dette kan bare skje dersom rot kjører denne kommandoen.

■ ***chown* .<gruppe> blah.txt**

- Legger blah.txt til nytt gruppe medlemskapet.

■ ***chown* fred.users blah.txt**

Access Control

Egenskapen `umask` i UNIX

■ ***umask***

- Det er tall som består av tre oktal tall.
- Bestemmer hvilke aksess tillatelser filer laget med en kjørende prosess får.
- Hovedoppgaven er å spesifisere hvilket aksess bit som aldri skal settes under opprettelsen av en ny fil.

Access Control

Det programmerbare interfacet

- Alle kommandoene som er beskrevet til nå kan kjøres med et program.
- Her følger 2 eksempler på slike program.
- I det første eksempelet benytter vi ***fchmod()*** i stedet for ***chmod()*** for unngå RC (kapittel 9).
- ***fchmod()***
 - Input: En åpen fil descriptor
Nye tillatelser (mode_t, 16 bits integer)

Access Control

Det programmerbare interfacet

```
1. #include <sys/types.h>
2. #include <sys/stat.h>
3. #include <stdio.h>

4. /* do equivalent of: chmod u=rw file */

5. int set_perms_to_0600(FILE * f)
6. {
7.     int fd;

8.     /*
9.      * fchmod works on a file descriptor, not a FILE *. Therefore,
10.     * convert the FILE* to an fd.
11.     */

12.     fd = fileno(f);
13.     if (fchmod(fd, S_IRUSR | S_IWUSR)) {
14.         perror("set_perms_to_0600");
15.         return -1;
16.     }
17. }
```

Access Control

Det programmerbare interfacet

```
1. #include <sys/types.h>
2. #include <unistd.h>
3. #include <stdio.h>

4. int chown_to_100(FILE * f)
5. {
6.     int fd = fileno(f);
7.     if (fchown(fd, .1, 100)) {
8.         perror("chown_to_100");
9.         return -1;
10.    }
11. }
```

Access Control

setuid programmering

- Et setuid program som ikke eies av rot kan bare foreta to operasjoner på UID og EUID.
 - Vi kan bytte UID og EUID med hverandre.
 - Vi kan sette restriksjoner på å aksessere en annen brukers UID.
- I et tilfelle der en annen brukers program blir kjørt er dette en smart ting å gjøre.
 - Ellers ender vi opp med å kjøre en annen brukers program med dine rettigheter, uten å vite konsekvensen av det.

Access Control

setuid programmering

- Vanligvis er setuid program eid av rot brukeren.
 - Krever som regel spesielle privilegier.
- Eksempler på situasjoner hvor bruk av setuid program er hensiktsmessig for å åpne opp maskinen til flere brukere:
 - Programmer som er avhengig av nettverksporter lavere enn 1024.
 - Kall til `chown()`, `chroot()`
 - Bruker administrasjon, som endring av passord
 - Direkte tilgang til enhetsdrivere.

Access Control setuid programmering

- Problemet
 - All aksess til rot, kan potensielt misbrukes dersom det finnes den minste feil i koden.
- Det anbefales å være veldig restriktiv med bruk av enhver form utdeling av rot privilegier.
- Det finnes flere ulike måter å begrense risikoene på, men det er ofte svært tungvinte løsninger. Dette kan bl.a være bruk av:
 - Daemons for moderering av tilgang til enhetsdrivere.
 - UINIX domene sockets
 - Bruk av en forked pipe på setuid programmet.

Access Control

setuid programmering

■ Andre problem:

- Unngå bruk av bruker "nobody", siden denne blir brukt av mange programmer. Dersom en av dem blir brutt, blir resten brutt samtidig.
 - Opprett heller en ny bruker for hvert applikasjon som kjøres.
- Skript som er setuid var tidligere en stor sikkerhets risiko på UNIX.
 - Angripere fikk lett tilgange til å kjøre ondsinnet kode med privilegiene til eieren av setuid skripte.
 - Vær svært varsom vis du lager et slikt skript.

Access Control

Access Control i Windows NT

- Windows benytter sikkerhets IDer (SID).
 - For individuelle brukere (account SID).
 - For grupper (group SID).
- Windows benytter ulike typer token
 - Aksess token er den viktigste
 - Avgjør om en gitt entitet har blitt autentisert tidligere.
 - Etterlignings token er også viktig
 - Tillater en applikasjon å benytte sikkerhets profilen til en annen bruker.
 - Tilsvare setuid, men har en helt annen implementasjon.

Access Control

Access Control i Windows NT

- Windows har også sikkerhets attributter.
 - Lagrer aksess tokens
 - Spesifisere privilegier til en entitet
 - Brukes til å avgjøre om gitte rettigheter kan overføres til en annen bruker.
- NT tilbyr lang høyere grad av tilpassete privilegier enn UNIX gjør.
 - For eksempel følger retten til å overføre eierskap implisitt med i UNIX.
 - I NT implementeres dette som separate attributter.

Access Control

Access Control i Windows NT

- UNIX tilbyr bare tre typer filrettigheter.
 - Les, skriv, kjør
- NT har fire grunn typer av filrettigheter.
 - "Ingen tilgang" Filen kan overhode ikke røres.
 - "Lese tilgang" Tillater resultat på spørring, lesing av data og kjøring av filen.
 - "Endre tilgang" Tillater at filen kan endres, slettes, og at den viser eierskap og tillatelser til filen.
 - "Full kontroll" Tilsvarer "Endre tilgang" pluss anledning til å endre fil tillatelser og å ta eierskap over en fil.

Access Control

Access Control i Windows NT

- En av hovedfordelene med NT modellen er den dynamiske utvidbarheten den har.
- Kernen holder på de viktigste egenskapene.
- På NT maskiner er det flere ulike typer brukere, i motsetning til UNIXs to.
 - Standard bruker
 - Administrator, tilsvarer UNIX UID 0.
 - Gjest, noe mer restriksjoner enn en standard bruker.
 - Operator, har visse nyttige administrator privilegier.

Access Control

Access Control i Windows NT

- Operator typen, inneholder flere brukere med ulike subset av administrator rettigheter.
- Print operatoren
 - En tjeneste som har tilgang til det meste som er relatert til printeren.
 - Har derimot ingen adgang til filer utenfor print spooler katalogen.
- Bruken av operatører begrenser evnen en angriper har til å komprimere systemet.

Access Control

Access Control i Windows NT

- Access Control List (ACL)
- Hver entitet i NT har sin egen ACL
 - Medfører en langt høyere grad av granularitet enn UNIX kan tilby.
 - Listen inneholder brukere og grupper og deres assosierte egenskaper.
- Vi kan foreksempel gi brukere i samme gruppe ulike rettigheter til filer som tilhører gruppen.
 - Bruk av grupper er strengt tatt unødvendig her.
- ACL støtter også arv, noe som letter konfigurasjons oppgaver.

Access Control

Minimalisering av UNIX problemer

- Opprette et virtuelt filsystem.
 - Et angrep vil dermed kun påvirke filer i det virtuelle filsystemet.
 - Ikke en ideell løsning.
- "Sandboxing"
 - OS påtvinger en "just-in-time" AC regel.
 - OS kan forby kall fra et program ut i fra hvordan kallet blir gjort.
 - Krever mye arbeid å sette opp reglene.

Access Control

Oppsummering

- AC er en av de mest fundamentale sikkerhets servicene et OS tilbyr.
- AC er vanskelig å bruke riktig.
- AC modellene er ofte svært ulik hverandre.



Kapittel 9

Race Conditions

Side 209 – 230

Race Conditions

Intro, Del 1

- Blant de mest vanlige klassene av feil man finner i ferdig programvare.
- Finnes i systemer hvor man kjører flere tråder eller prosesser samtidig som kan påvirke hverandre.
- Det kan være veldig vanskelig å finne disse feilene.

Race Conditions

Intro, Del 2

- De kan få tilsynelatende deterministiske programmer til å virke svært ikke-deterministiske.
- Med økende grad av applikasjoner som benytter flere tråder, prosesser eller distribuert prosessering, vil problemet bare fortsette å vokse seg større og større.

Race Conditions

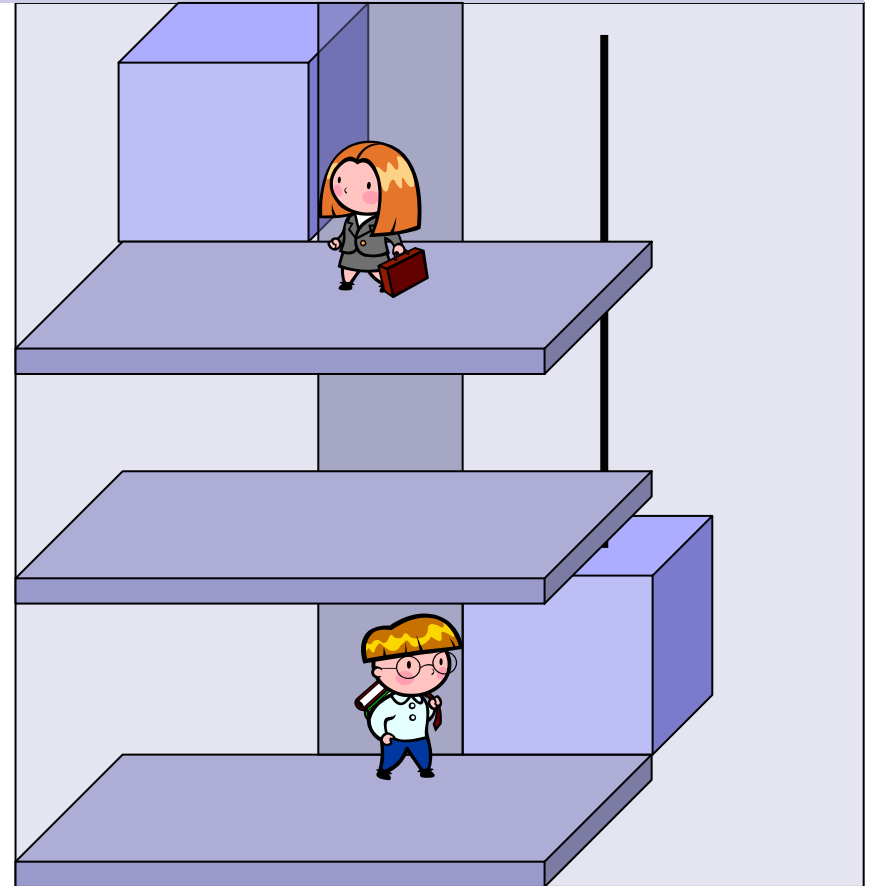
Intro, Del 3

- RC representerer først og fremst et robusthets problem for en applikasjon.
- Men RC kan også representere til dels store sikkerhets problemer.
- En av de største farene finner vi ved aksess kontrollen til filsystemer.

Race Conditions

Hva er en Race Condition?

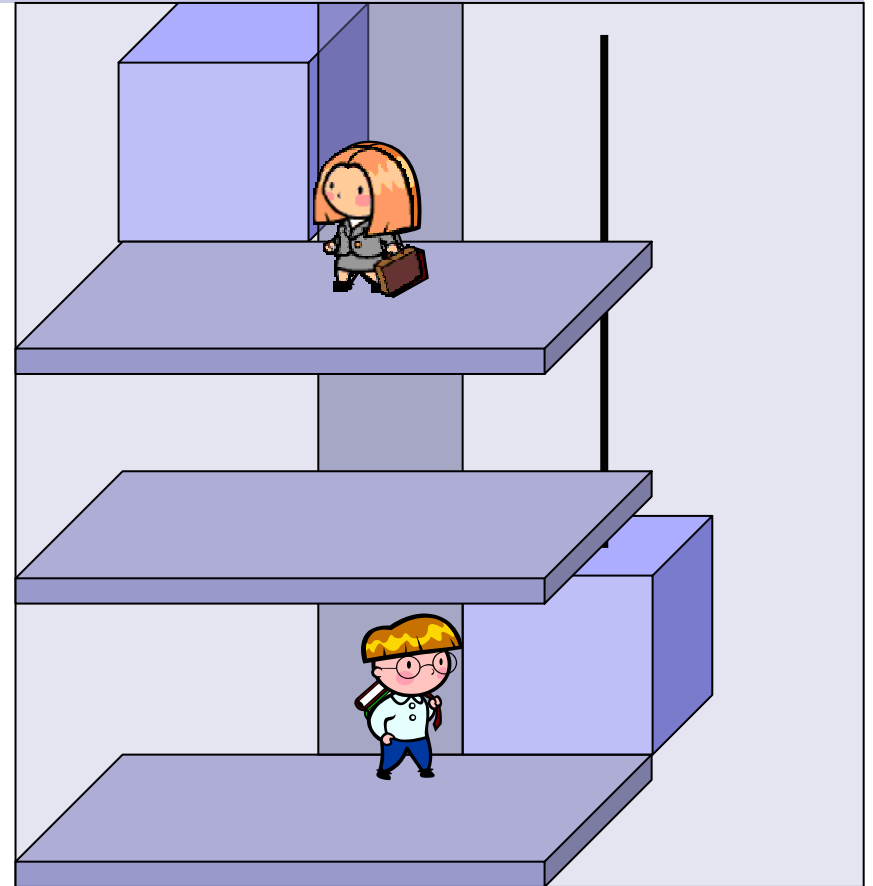
- Alice og Bob jobber i samme bygget.
- De ønsker å treffe hverandre i gangen, og avtaler et møte klokken 3 med email.
- Klokken 3 venter de på hverandre, men i ulike etasjer.



Race Conditions

Hva er en Race Condition?

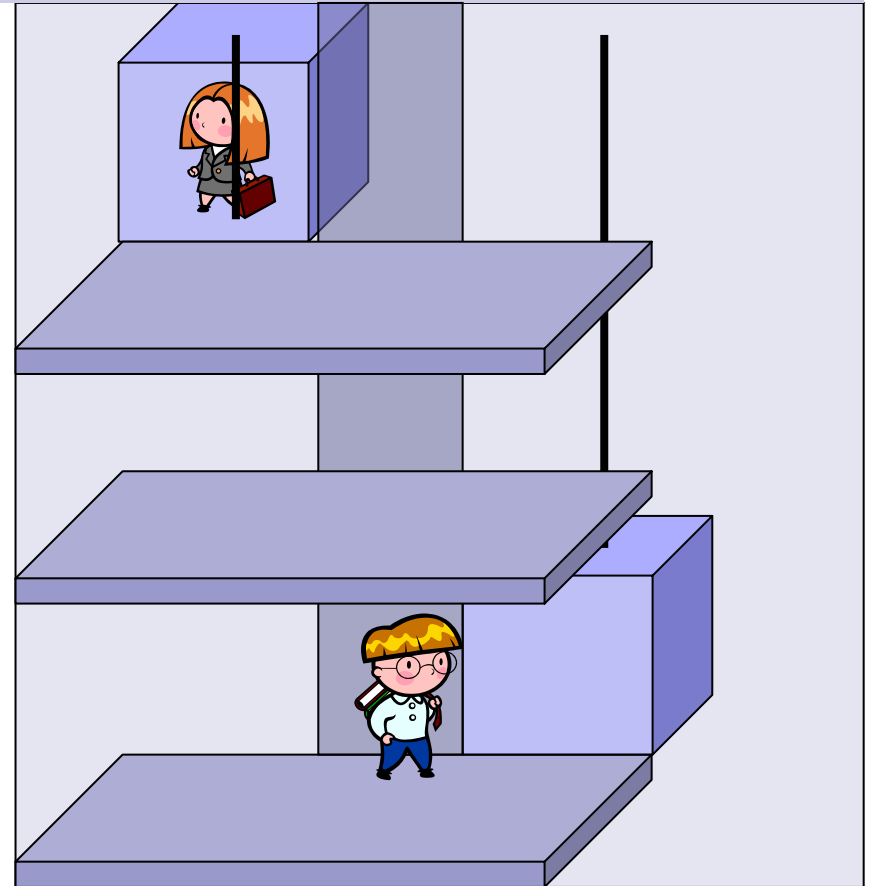
- Hva skjer når de begynner å lete etter hverandre?



Race Conditions

Hva er en Race Condition?

- Hva skjer når de begynner å lete etter hverandre?



Race Conditions

Hva er en Race Condition?

- Dersom begge befinner seg i hver sin heis samtidig vil de passere hverandre.
- Når både Alice og Bob antar noe om hvor den andre befinner seg i dette tilfellet, har de endt opp i en Racing Condition.

Race Conditions

Hva er en Race Condition?

- En Racing Condition oppstår når en antakelse må være sann for et gitt tidsrom, men er i virkeligheten ikke er det.
- I hver Racing Condition eksisterer det et sårbarhetsvindu.
 - En periode hvor overtredelser av de gitte antakelsene medfører feil oppførsel av systemet.

Race Conditions

Hva er en Race Condition?

- Denne koden teller antall side treff.
- Java servelets er flertrådet, og har i dette tilfellet en RC.
- Hvor er sårbarhets vinduet i denne koden?

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {

    int count = 0;

    public void doGet(HttpServletRequest in,
        HttpServletResponse out)
        throws ServletException, IOException {

        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        count++;
        p.println(count + " hits so far!");

    }
}
```

Race Conditions

Hva er en Race Condition?

- Man antar her at variabelen **count** ikke har endret seg fra den er inkrementert til den blir skrevet ut.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {

    int count = 0;

    public void doGet(HttpServletRequest in,
        HttpServletResponse out)
        throws ServletException, IOException {

        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        count++;
        p.println(count + " hits so far!");
    }
}
```

Race Conditions

Hva er en Race Condition?

- Man antar her feilaktig at variabelen **count** ikke har endret seg fra den er inkrementert til den blir skrevet ut.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {

    int count = 0;

    public void doGet(HttpServletRequest in,
        HttpServletResponse out)
        throws ServletException, IOException {

        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        p.println(++count + " hits so far!");
    }
}
```

Race Conditions

Hva er en Race Condition?

- Man tillater her en pause mellom to instruksjoner.
- I flertråds systemer kan en slik pause fort bli lengre enn antat, og man åpner for mulige feil.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {

    int count = 0;

    public void doGet(HttpServletRequest in,
        HttpServletResponse out)
        throws ServletException, IOException {

        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        p.println(++count + " hits so far!");
    }
}
```

Race Conditions

Hva er en Race Condition?

- I pausen kan fort telleren bli økt av en annen tråd.
- Slike feil er ofte svært sjelden.
- Det skal derfor gjerne veldig mye til for å oppdage og i tillegg kunne reprodusere en slik potensiell feil.
- Dette gjelder også for større sårbarhets vindu.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {

    int count = 0;

    public void doGet(HttpServletRequest in,
        HttpServletResponse out)
        throws ServletException, IOException {

        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        p.println(++count + " hits so far!");
    }
}
```

Race Conditions

Hva er en Race Condition?

- Hvordan utnyttes dette i praksis?
 - En potensiell angriper tar kontroll over maskin ressursene.
 - Angriperen kan deretter få maskinen til å gå saktere, for å øke sårbarhets vinduet.
 - Bruker så et automatisert skript til å utnytte en potensiell feil.
 - En feil trenger bare å oppstå en gang.
 - Så selv med ods 1:1.000.000 trenger ikke angrepet å ta mer enn noen få sekunder.

Race Conditions

Hva er en Race Condition?

- Hvordan fikse dette problemet?
 - Minske sårbarhets vinduet til 0, ved å forsikre seg om at alle antakelser som er blitt gjort holder uansett.
 - Dette gjøres ved å omgjøre den risiko fylte koden til atomisk kode ("atomic code").
 - Dette vil si relevant kode som kjøres som en enhet uten risiko for at feil kan oppstå.

Race Conditions

Hva er en Race Condition?

- Hvordan lager vi atomisk kode?
 - Vi kan låse primitiver.
 - Dette gjelder spesielt i flertråds systemer.
 - I eksempelet vårt ville vi benyttet en objektlås for å oppnå dette ved hjelp av Java nøkkelordet ***synchronized***.
 - Dette vil for eksempel medføre at dersom vi har 10 synkroniserte metoder i en Java klasse, vil bare en tråd kunne kjøre en av disse metodene om gangen.

Race Conditions

Hva er en Race Condition?

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

Forbedret

```
public class Counter extends HttpServlet {

    int count = 0;

    public synchronized void
doGet(HttpServletRequest in, HttpServletResponse
out)
throws ServletException, IOException {

        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        count++;
        p.println(count + " hits so far!");
    }
}
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

Opprinnelig

```
public class Counter extends HttpServlet {

    int count = 0;

    public void doGet(HttpServletRequest in,
HttpServletResponse out)
throws ServletException, IOException {

        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        count++;
        p.println(count + " hits so far!");
    }
}
```

Race Conditions

Hva er en Race Condition?

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

Forbedret

```
public class Counter extends HttpServlet {
```

```
    int count = 0;
```

```
    public synchronized void
doGet(HttpServletRequest in, HttpServletResponse
out)
throws ServletException, IOException {

        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        count++;
        p.println(count + " hits so far!");
    }
}
```

- Hva er problemet med denne løsningen?
 - Den kan ha en svært negativ effekt på ytelsen.
 - I dette eksempelet vil servleten bare kunne betjene en bruker om gangen. Pga. at **doGet** er startpunktet.

Race Conditions

Hva er en Race Condition?

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {

    int count = 0;

    public void doGet(HttpServletRequest in,
        HttpServletResponse out)
        throws ServletException, IOException {

        int my_count;
        out.setContentType("text/plain");
        PrintWriter p = out.getWriter();
        synchronized(this) {
            my_count = ++count;
        }
        p.println(my_count + " hits so far!");
    }
}
```

Anbefalt

- En mer effektive løsning er å forsikre seg om at koden som er atomisk, er så liten som mulig.
 - "Critical Section"
- I Java kan vi benytte ***synchronized*** på en enkel kodeblokk hvis vi ønsker.

Race Conditions

Hva er en Race Condition?

■ Foreløpig oppsummering

- RC kan oppstå når to eller flere operasjoner skjer samtidig, og hvor en av de seinere operasjonene avhenger av den første.
- I tidsrommet mellom hendelser kan en angriper utnytte svakheter i systemet til å påtvinge dette en ny oppførsel.
- Dette krever som regel svært mye kunnskap av angriperen:
 - Forstå den sikkerhet kritiske sammenhengen.
 - Forstå hvordan man oppnår en perfekt timing.
 - Kunnskap om antakelser som utviklerne kan ha hatt.

Race Conditions

Hva er en Race Condition?

■ Foreløpig oppsummering

- Benevnelsen "Race Condition" omtaler et kappløp som pågår mellom angriperen og utvikleren.
- Et suksessfullt angrep involverer å endre situasjonen "raskt og skittent" på en måte som ikke har vært forutsett av utvikleren.

Race Conditions

Best bruk

- Fil baserte RC er de mest notoriske med hensyn på sikkerhets kritiske RC.
- Dette er hovedsaklig er problem på UNIX maskiner siden Windows API vanskeliggjør angrep med f.eks. bruk av ***handels.***

Race Conditions

Time-of-Check, Time-of-Use

- Et vanlig hendelses forløp for en filbasert RC:
 - Før bruk sjekkes en fil for en egenskap.
 - Sjekken må være gyldig ved bruks tidspunktet for å virke skikkelig, men dette kan hende ikke er tilfelle.
 - Slike feil kalles TOCTOU.

Race Conditions

Time-of-Check, Time-of-Use

■ Eksempel

- Et program som kjører **setuid** på rot blir spurt om å skrive til en fil.
 - Denne eies av brukeren som kjører programmet.
- Rot brukeren kan skrive til hvilket som helst fil.
 - Programmet må være forsiktig og unngå å skrive til noe som den aktuelle brukeren ikke har tilgang til.

Race Conditions

Time-of-Check, Time-of-Use

■ Eksempel forts.

□ Anbefalte løsning:

- Sett EUID til UIDen som kjører programmet.

□ Feil løsning (ofte brukt):

- Benytt kallet **access** for å oppnå det samme resultatet.

Race Conditions

Time-of-Check, Time-of-Use

- Access kallet sjekker om den virkelige UIDen har tillatelse til en ønsket sjekk, og returnerer 0 hvis ikke.
- En teksteditor som må kjøre som rot er et eksempel på dette.

```
/* access returns 0 on success*/  
if(!access(file, W_OK)) {  
    f = fopen(file, "wb+");  
    write_to_file(f);  
}  
else {  
    fprintf(stderr, "Permission denied when  
trying to open %s.\n", file);  
}
```

Race Conditions

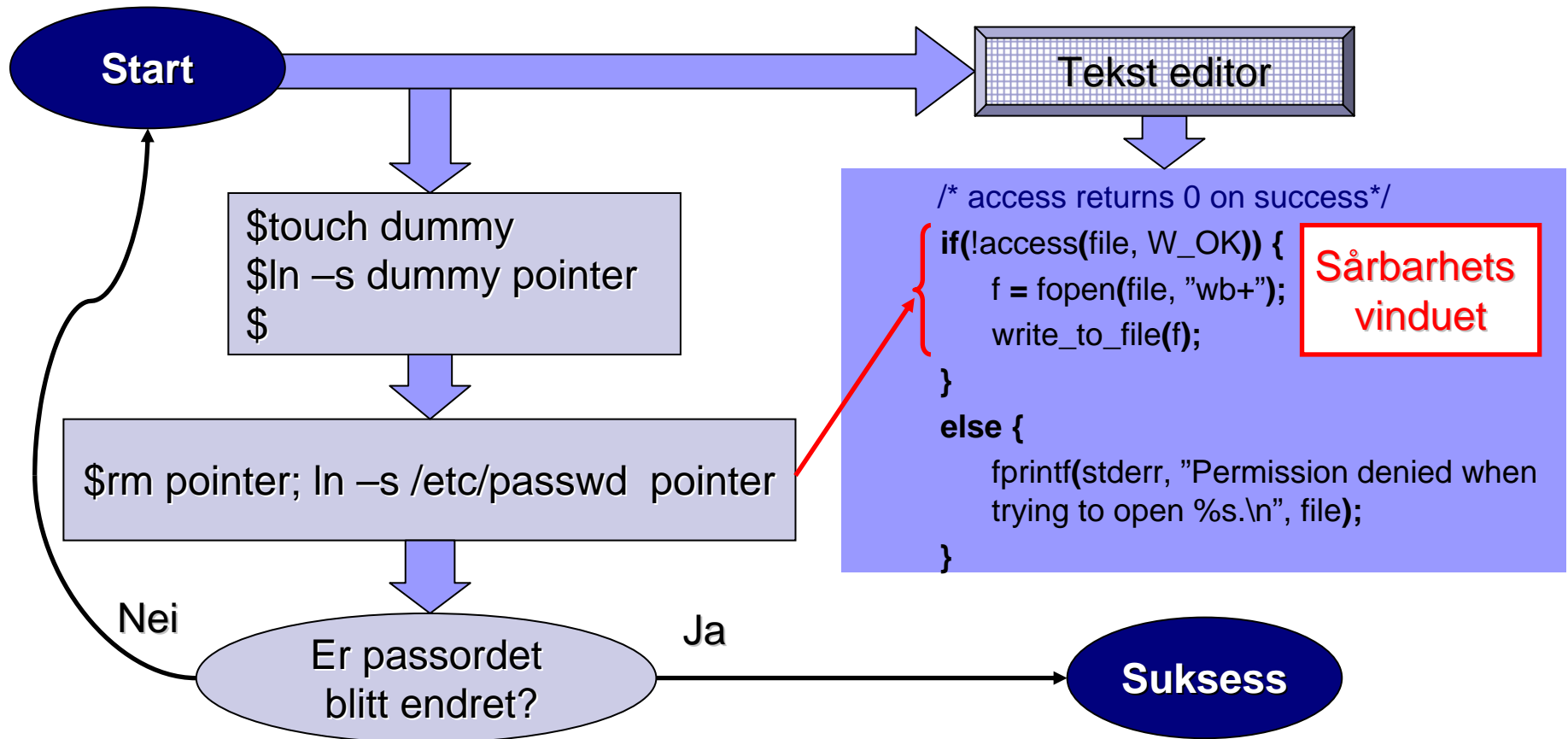
Time-of-Check, Time-of-Use

- I dette tilfellet vil angriperen kunne lage en ondsinnet fil som f.eks. `/etc/passwd`.
- Det er da bare et spørsmål om å utnytte RCen for å installere det nye passordet.
- Sårbarhets vinduet oppstår her mellom **access** og tiden det tar å åpne en file med **fopen**.

```
/* access returns 0 on success*/  
if(!access(file, W_OK)) {  
    f = fopen(file, "wb+");  
    write_to_file(f);  
}  
else {  
    fprintf(stderr, "Permission denied when  
    trying to open %s.\n", file);  
}
```

Race Conditions

Time-of-Check, Time-of-Use



Race Conditions

Time-of-Check, Time-of-Use

- Forutsetning for et angrep:
 - Lokal akcesess, enten lovlig eller ulovlig.
 - Programmet med RCen må kjøre med en rot EUID.
 - Programmet må ha denne EUIDen for den perioden som RCen eksisterer.
- Målet er å få rot privilegier.

Race Conditions

Time-of-Check, Time-of-Use

- Hvordan unngå TOCTOU problemer:
 - Unngå bruk av systemkall som tar et filnavn som input.
 - Bruk heller en fil handler eller en fil descriptor.
 - Bruk av fil descriptorer og fil pekere forsikrer oss om at vi har full oversikt over hva som skjer.

Race Conditions

Time-of-Check, Time-of-Use

- Hvordan unngå TOCTOU problemer:
 - Benytt ***fstat()*** på en fil etter at du har åpnet den, istedenfor ***stat()*** før den er åpnet.
 - Overlat aksess sjekker til det underliggende systemet.
 - Dette betyr at du setter EUID eller EGID med ***setgroups(0,0)*** i stedetfor å benytte ***access()***

Race Conditions

Time-of-Check, Time-of-Use

- Hvordan åpne en fil på en sikker måte:
 - 1) lstat() filen før du åpner den, og lagre stat strukturen.
 - 2) Kjør open()
 - 3) Fstat() fil descriptoren som ble returnert i 2), og lagre stat strukturen.
 - 4) Sammenlign de to stat strukturene.

Race Conditions

Time-of-Check, Time-of-Use

- Følgende kode viser et eksempel på denne teknikken.
- Dette programmet vil åpne en fil på en sikker måte ved å simulere ***fopen w+***
- Dvs. at filen vil bare bli opprettet dersom den ikke allerede eksisterer, i motsatt tilfelle blir den trunkert.

Race Conditions

Time-of-Check, Time-of-Use

```
1. #include <sys/stat.h>
2. #include <sys/types.h>
3. #include <sys/stat.h>
4. #include <fcntl.h>
5. #include <unistd.h>
6. #include <stdio.h>
7. #include <errno.h>

8. FILE * safe_open_wplus(char *fname)
9. {struct stat  lstat_info, fstat_info;
10.  FILE *fp;
11.  char *mode = "rb+"; /* We perform our own truncation. */
12.  int fd;

13.  if (lstat(fname, &lstat_info) == .1) {
14.    /*
15.    * If the lstat() failed for reasons other than the file
16.    not
17.    * existing, return 0, specifying error.
18.    */
19.    if (errno != ENOENT) {
20.      return 0;
21.    }
22.    if ((fd = open(fname, O_CREAT | O_EXCL | O_RDWR,
23. 0600)) == .1){
24.      return 0;
25.    }
26.  }
```

```
25.  mode = "wb";
26.  }else {
27.    /* Open an existing file */
28.    if ((fd = open(fname, O_RDWR)) == .1) {
29.      return 0;
30.    }
31.    if (fstat(fd, &fstat_info) == .1 ||
32.        lstat_info.st_mode != fstat_info.st_mode ||
33.        lstat_info.st_ino != fstat_info.st_ino ||
34.        lstat_info.st_dev != fstat_info.st_dev) {
35.      close(fd);
36.      return 0;
37.    }
38.    /* Turn the file into an empty file, to mimic w+
39. semantics. */
40.    ftruncate(fd, 0);
41.  }
42.  /* Open an stdio file over the low-level one */
43.  fp = fdopen(fd, mode);
44.  if (!fp) {
45.    close(fd);
46.    unlink(fname);
47.    return 0;
48.  }
49.  return fp;
50. }
```

Race Conditions

Sikker filtilgang

- Beste og enkleste løsningen for å sikre filer.
 - Flytt dem til en sikker katalog.
 - Dvs. la kun et gitt program med riktig UID ha tilgang til fil operasjoner.
 - Dette vil fjerne all fare forbundet med RC, siden angriperen må ha riktig UID for å lykkes.

Race Conditions

Sikker filtilgang

■ Hvordan oppnå dette?

- Vi må forsikre oss mot at angriperen ikke kan endre noen av foreldre katalogene.
 - Lag en katalog på ønsket sted.
 - Traverser så mappe treet oppover med ***chdir()***
 - For hvert nivå opp til og med rot sjekker vi:
 - At mappen ikke er en link
 - At bare rot eller aktuell bruker kan endre på katalogen (eier egenskapene til UID og GID)

Race Conditions

Sikker filtilgang

- Følgende eksempel illustrer dette
- Vi spør her om en gitt katalog er sikker å bruke med tanke på RC.
- Vi sender med en troverdig UID også, vanligvis brukes ***getuid()***
- Programmet returnerer 0 ved suksess.

Race Conditions

Sikker filtilgang

```
1. #include <sys/types.h>
2. #include <sys/stat.h>
3. #include <sys/symlimits.h>
4. #include <fcntl.h>
5. #include <unistd.h>
6. static char
7. safe_dir(char *dir, uid_t owner_uid)
8. {
9.     char        newdir[PATH_MAX + 1];
10.    int          cur = open(".", O_RDONLY);
11.    struct stat   linfo, sinfo;
12.    int          fd;
13.    if (cur == -1) {
14.        return -1;
15.    }
16.    if (lstat(dir, &linfo) == .1) {
17.        close(cur);
18.        return -2;
19.    }
20.    do {
21.        chdir(dir);
22.        if ((fd = open(".", O_RDONLY)) == .1) {
23.            fchdir(cur);
24.            close(cur);
25.            return -3;
26.        }
27.        if (fstat(fd, &sinfo) == .1) {
28.            fchdir(cur);
29.            close(cur);
30.            close(fd);
31.            return -4;
```

```
33.     }
34.     close(fd);
35.     if (linfo.st_mode != sinfo.st_mode ||
36.         linfo.st_ino != sinfo.st_ino ||
37.         linfo.st_dev != sinfo.st_dev) {
38.         fchdir(cur);
39.         close(cur);
40.         return -5;
41.     }
42.     if ((sinfo.st_mode & (S_IWOTH | S_IWGRP)) ||
43.         (sinfo.st_uid && (sinfo.st_uid != owner_uid))) {
44.         fchdir(cur);
45.         close(cur);
46.         return -6;
47.     }
48.     dir = "..";
49.     if (lstat(dir, &linfo) == .1) {
50.         fchdir(cur);
51.         close(cur);
52.         return -7;
53.     }
54.     if (!getcwd(newdir, PATH_MAX + 1)) {
55.         fchdir(cur);
56.         close(cur);
57.         return -8;
58.     }
59. } while (strcmp(newdir, "/"));
60. fchdir(cur);
61. close(cur);
62. return 0;
63. }
```

Race Conditions

Sikker filtilgang

- Sikker sletting av filer:
 - Ved bruk av en sikker mappe, vil det være sikkert å slette en fil med ***unlink()*** ellers ikke.
 - For helt sikker sletting må filen overskrives minst 7 ganger.
 - Er du ekstremt paranoid
 - Benytt Peter Gutmanns 35-pass algoritme som et minimum.

Race Conditions

Midlertidige filer

- Det er vanlig å opprette midlertidige filer i delte områder som f.eks. /temp
- Disse filene kan være utsatt for akkurat de samme angrepene som vanlige filer.
- Vanligste strategi er for angriper er å gjette filnavn, noe som ofte kan være enkelt.
- Her følger en løsning på problemet:

Race Conditions

Sikker filtilgang

- 1) Velg et prefiks for ditt filnavn. `/tmp/my_app`
- 2) Generer minimum 64 bits høy kvalitets tilfeldighet.
- 3) base64 encode så disse.
- 4) Konkatener prefikset med den enkodete tilfeldige dataen.
- 5) Sett umask. `0066`
- 6) Bruk ***fopen()*** til å lage filen.
- 7) Slett filen så umiddelbart ved hjelp av ***unlink()***
- 8) Du kan nå lese, skrive eller søke som ønsket på filen.
- 9) Steng filen til slutt.

Race Conditions

File Locking

- Ved å låse filer vil vi på de fleste filsystemer unngå å risikere RC.
- Unntaket er NFS versjoner eldre enn versjon3.

Race Conditions

File Locking

- Den beste løsningen for atomisk fil låsing med lockfile.
 - Lag en unik fil (host navn + pid)
 - Bruk ***link(2)*** for å lage en link til lockfile.
 - Hvis ***link()*** returnerer 0, er låsen en suksess.
 - Ellers bruk ***stat(2)*** på den unike filen for å se om link telleren har økt til 2, da er låsen også en suksess.

Race Conditions

File Locking

- Bakdeler med fillåsing er:
 - Vis en gitt fil er låst, kan andre måtte vente i evigheter for å få aksess.... som i **CVS**
 - Dette oppstår gjerne dersom et program krasjer etter at det har låst en fil.
 - Alternativet er å stjele låser.
 - Innebærer en fare for ny RC.

Race Conditions

Andre Race Conditions

- Sikkerhets kritiske RC oppstår også i andre tilfeller.
 - Vanlig i komplekse systemer
 - Effektive betalings systemer med flere back-end databaser.
 - Tidligere gikk det an å overskrive kontoen sin flere ganger om natten på en minibank.
 - Eldre versjoner av Java har problemer med å oppdatere policyene som blir brukt for eks. i multimedia presentasjoner.
 - RC oppstår i mellomrommet mellom sletting og opprettelse av en policy.

Race Conditions

Konklusjon

- Flertråds programmer er ekstremt vanskelig å debugge.
- I slike programmer er RC den mest sikkerhetsaktuelle problemstillingen.
- Unngå bruk av flertråds systemer så langt det lar seg gjøre.
- For unngå RC, må du tenke deg om minst to ganger før du antar at disse problemene er løst.