Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

<u>Hvis</u> du ikke vil være med på opptaket:

	La være å delta med webkameraet ditt.
	La være å delta med mikrofonen din.
	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.

Høyskolen Kristiania

Høyskolen Kristiania

PG3401 Programmering i C for Linux

Bengt Østby

Høyskolen Kristiania

PG3400 Programmering i C for Linux

Lecture #11:
Safe programming + repetition

Tread terminology…

Høyskolen Kristiania

Main thread (no: hovedtråd)

Worker thread (no: arbeidstråd)

Thread pool (no: tråd “pool”), Worker thread pool

Use this makefile

Høyskolen Kristiania

Create these two sub folders

```
# Makefile template

INCLDIR = ./include
CC = gcc
CFLAGS = -O2
CFLAGS += -I$(INCLDIR)

OBJDIR = obj

_DEPS = source.h
DEPS = $(patsubst %,$(INCLDIR)/%,$(_DEPS))

_OBJS = source.o
OBJS = $(patsubst %,$(OBJDIR)/%,$(_OBJS))


$(OBJDIR)/%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

hello: $(OBJS)
	gcc -o $@ $^ $(CFLAGS)

.PHONY: clean
clean:
	rm -f $(OBJDIR)/*.o *~ core $(INCLDIR)/*~
```

List all header files here

List all source files here, but given their .o file convention

Change output filename here

Recap

Høyskolen Kristiania

- Du MÅ opprette de to mappene include og obj
- Du MÅ legge header filer i include mappen
- Object filer og header filer skal listes opp UTEN komma, kun mellomrom mellom navn
- IKKE endre andre ting i den makefilen, make er som et veldig pirkete og lite brukervennlig skriptspråk – hvis du setter inn ETT mellomrom, eller fjernet ett mellomrom, på feil sted virker det ikke
- Vi bruker ikke makefile fordi den er så enkel å bruke (eller forstå), men fordi det gjøre byggingen så lett å forstå – og å reprodusere (og det er viktig for eksamen)
- Sensor vil skrive «make» uten parametere, og da skal det virke ☺

- Andre tips og triks dere ønsker å dele med klassen?

EKSAMEN

Oppgavesettet har ? sider. Det er totalt 6 oppgaver i oppgavesettet.

Det er 2 ukers frist på denne hjemmeeksamen, men forventet arbeidsmengde er 5-7 dager med «normale» arbeidsdager. Fristen kan overlappe med andre eksamener dere har, det er derfor viktig at dere bruker tiden effektivt i starten av eksamensperioden så dere ikke rett før innlevering blir sittende og skulle levere flere eksamener samtidig. Vær obs på at eksamen MÅ leveres innen fristen som er satt i Wiseflow, og oppgaven kan kun leveres via WISEFLOW. Det vil ikke være mulig å få levert oppgaven etter fristen – det betyr at du bør levere i god tid slik at du kan ta kontakt med eksamenskontoret eller brukerstøtte hvis du har tekniske problemer.

Det presiseres at studenten skal besvare eksamen selvstendig og individuelt, samarbeid mellom studenter og plagiat er ikke tillatt.

Merk at oppgavene er laget med stigende vanskelighetsgrad, og spesielt de to siste oppgavene er vanskeligere enn de første oppgavene. Det oppfordres derfor til å gjøre de første oppgavene (helt) ferdige slik at studenten ikke bruker opp all tid på å gjøre de siste oppgavene først.

EKSAMEN

Høyskolen Kristiania

Dette er en praktisk programmeringseksamen, fokus bør derfor være på å forklare hvordan du har gått frem, begrunne valg og legge frem eventuelle antagelser du har gjort i din løsning.

Hvis du ikke klarer å løse en oppgave er det bedre om du forklarer hvordan du har gått frem og hva du ikke fikk til – enn å ikke besvare oppgaven i det hele tatt. Det forventes at alt virker hvis ikke annet er beskrevet i tekstbesvarelsen, hvis du vet at programmet krasjer, ikke kompilerer eller ikke virket slik den var tenkt er det viktig å forklare dette sammen med hvilke skritt du har tatt for å forsøke å løse problemet.

Besvarelsen skal være i 1 ZIP fil, navnet på filen skal være PG3401_H21_[kandidatnummer]. Denne filen skal ha følgende struktur:

\ oppgave_2 \ makefile

\ oppgave_2 \ […]

[…]

Du skal laste opp:

PG3401_H21_[kandidatnummer].pdf

+ zip filen som vedlegg

EKSAMEN

Høyskolen Kristiania

Vær sikker på at alle filer er med i ZIP filen. Hvis du velger å ha flere programmer som en del av løsningen (for eksempel oppgave 7) så legger du det ene programmet i \oppgave_7A og den andre i \oppgave_7B. Hver mappe skal ha en makefile fil, og det skal ikke være nødvendig med noen endringer, tredjeparts komponenter eller parametere – sensor vil i shell på Debian Linux 10 gå inn i mappen og skrive «make» og dette skal bygge programmet med GCC.

Tekstbesvarelsen skal inneholde besvarelse på oppgave 1 (skriv det kort og konsist, trenger ikke noe stor avhandling). Etter den rene tekstbesvarelsen skal det være 1 sides begrunnelse/dokumentasjon for hver oppgave, hver av disse begrunnelsene skal være på en ny side for å gjøre tekstbesvarelsen oversiktlig for sensor. Besvarelsen skal være i PDF format eller i Microsoft Word format (DOCX) og ha korrekt file-extension til å kunne åpnes på både en Linux og en Windows maskin (.pdf eller .docx). Besvarelser i andre formater vil ikke bli lest.

EKSAMEN

Høyskolen Kristiania

Oppgave 1 er litt teori.

Oppgave 2 – 3 er ganske enkle.

Oppgave 4 går ut på at dere skal finne feil i min kode :-)

Oppgave 6 – 7 kan være vanskelige, og de er ment å ha stigende vanskelighetsgrad, det betyr at du bør ta oppgavene i riktig rekkefølge med mindre du sitter helt fast (ikke start på oppgave 7, og så bli sittende fast!) – Sammenlignet med 2020 er det 1 ½ oppgave mindre.

Dere har 2 uker på dere, men for noen er det andre «lange» eksamener i parallell, jeg forventer derfor ikke mer enn 5-7 dagers arbeid på eksamen. PLANLEGG godt og løs eksamen i god tid! Ikke sitt siste to dagene med flere hjemmeeksamener du ikke har startet på og få panikk…

Dere har i dag allerede fått utlevert eksamen. Det er ingen oppgavesett til i dag, så dere kan selv jobbe med det dere synes gir best effekt / læring. Husk at jeg og veiledere kan ikke svare på direkte spørsmål om hvordan dere løser en oppgave på eksamen, men vi kan svare på mer generelle spørsmål.

Alle spørsmål skal stilles på DISKUSJONER under emnesidene på Canvas! (Da får alle svar på det alle har spurt om, og vi risikerer ikke at det blir urettferdig for noen.)

The philosophy of C explains much of the behavior

Høyskolen Kristiania

- C is designed to make it easy to *write compilers*.
- C is designed with the assumption that the programmers *know* what they are doing.
- C is portable in source form.
- Nothing stops somebody from making a compiler with lots of warnings, but you can never rely on getting these!

C gives full control to the programmer, thus also full responsibility for correctness.

Høyskolen Kristiania

Bad coding can lead to bugs, but also security issues…

Høyskolen Kristiania

Safe programming

Some common mistakes

Høyskolen Kristiania

```
if(j == 100);
    j++;
```

Misplaced semicolon

```
if(a = 2)
    printf(”Done!\n");
```

‘=‘ versus ‘==‘

```
if(x = getchar() != EOF)
    doSomething();
```

Precedence of operators

```
int a[100];
for(int i=0;i<=100;i++)
```

Off by one

```
char *cp;
*cp='X';
```

Use before initialization

More common mistakes

Høyskolen Kristiania

```
case 1:
    doThings();
case 2:
        doOtherThings();
```

Accidental fallthrough

```
#define MAX_VALUE 1000;
```

Semicolon at end of macro

```
#define reciprocal(x) 1/x
```

Missing parenthesis in macro

```
#define square(x) (x*x)
w=square(v++);
```

Side effects in macro calls

Potentially dangerous memory overflows

Høyskolen Kristiania

- Remote exploits of apps
- Privilege escalation (mostly an issue for OS)
- Be careful whenever a user can input data!
- If you want to write exploits, learn assembler

Some dangerous functions…

Høyskolen Kristiania

memcpy()

gets()

strcpy()

sprintf()

array lookups

Forgotten null-terminator

Example (Code Red worm – Internet Information Server):

> telnet www.targetsystem.com 80
GET/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucb
d3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u
9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0

Dangling pointers are not easy to exploit, but make for bad bugs

Høyskolen Kristiania

```
{
  char *dp = NULL;
  /* ... */
  {
    char c;
    dp = &c;
  }
  /* c falls out of scope */
  /* dp is now a dangling pointer */
}
```

Integer overflows or underflows are dangerous

Høyskolen Kristiania

- If the number is later used as an index in an array, the user can force read of wrong data

Memory leaks can make a program crash after a while

Høyskolen Kristiania

- Usually from repeated malloc() without free
- Other resources can leak as well; file handles, sockets, etc

“Goto-statements considered harmful”

Høyskolen Kristiania

But can be used in C in place of exceptions:

```
FILE* files[10];
int opened_file;
for(opened_file=0; opened_file<10; opened_file++){
      files[opened_file]=fopen(file_name[opened_file]);
      //Now we need to check if it worked!
      if(files[opened_file]==NULL)
            goto CLOSE_ALL;
}
// Main code of function
CLOSE_ALL:
for(; opened_file>=0; opened_file--){
      fclose(files[opened_file]);
}
```

Goto is (by most developers) seen as very bad practice, do not use it…

Principal of Defensive Programming

- General quality – reducing the number of software bugs and problems
- Making code comprehensive – the source code should be readable and understandable so it is approved in a code audit
- Making the software behave in a predictable manner despite unexpected input or user actions

Defensive Programming

- Beskytt koden mot feil data «fra utsiden»
- Etabler sikre grensesnitt
- Valider alt ved input data gjennom grensesnittene; type, lengde, innhold i data, struktur på data
- Etabler en strategi for hvordan forskjellig typer feil data skal håndteres (erstatt med default data, fast fail, logging, osv)
- Ikke forvent at en ekstern metode kan kalles og vil oppføre seg slik den er dokumentert
- Kode for diagnosering, logging og tracing
- Forsiktig bruk av exception handling; exception handling skal ikke være «kode flyt» – men kritisk feilhåndtering

Avansert håndtering

- Kode som detekterer feil selv
- Kode som kan isolere skadelig data og beskytte seg selv og annen kode fra denne dataen
- Kode som kan gjenopprette seg selv

Eksterne metoder

- Ikke tro at eksterne metoder vil oppføre seg slik de er dokumentert
- Ikke tro at eksterne metoder vil oppføre seg i fremtiden slik de har oppført seg frem til nå
- Ikke vent «for alltid» på at en ekstern metode returnerer
  - «For alltid» er lenge hvis en tjeneste går ned…
- Forvent at returnerte verdier fra eksterne metoder kan være skadelige (metoden kan ha «blitt» hostile)
- Hvis du har skrevet et delt bibliotek (.so / .dll), ikke stol på at det er DITT bibliotek du laster, det kan være malware!

Pre conditions

- Pre conditions er en teknikk for å sikre at data er korrekte før de brukes
- Data sjekkes før de tas i bruk – dette gjør at det er lett å se i koden at alle input data er validert

```
public void CreateAppointment(DateTime dateTime)
{
    if (dateTime.Date < DateTime.Now.AddDays(1).Date)
        throw new ArgumentException("Date is too early");

    if (dateTime.Date > DateTime.Now.AddMonths(1).Date)
        throw new ArgumentException("Date is too late");

    /* Create an appointment */
}
```

Input validering

- All input må valideres

- Hvis et inndatafelt inneholder et telefonnummer; BARE tillat tall og +-tegnet
- Hvis et inndatafelt inneholder et navn; BARE tillat tegn som vanligvis brukes i et navn på språkene du antar å støtte

- Hvis du BRUKER input til noe, spesielt hvis du skal rendre data eller interfacer med for eksempel en backend database og input brukes i queries – da er du sårbar ovenfor alle vanlige C exploits og i tillegg typiske web exploits!

Whitelist ikke blacklist

- Det er nesten umulig, og veldig tidkrevende, å sjekke alle mulige måter å gi input som resulterer i at et skript blir kjørt
- Det samme gjelder andre angrep vi skal se på som SQL Injection og Code Injection

- Best Practice er derfor å begrense input ved å fjerne alle spesialtegn – eller helst fjerne alle tegn du ikke forventer!

- Input er et navn, tillat a-å, A-Å og ‘-’:
  - Angrepsstrengen er: "test <scr<script>ipt> Alert ('pwned'); </script> "
  - Resultat etter sletting blir “test scrscriptipt Alert pwned script“ – som aldri vil kunne skade noen…

Problemer med blacklist

- Problem med svartelisting er hva du skal svarteliste. Tenk deg at du ønsker å svarteliste en gitt IP-adresse 169.254.169.254
- På grunn av datamaskin heltall kan følgende IP-adresser (kanskje) være det samme:
  - 425.510.425.510
  - 2852039166
  - 7147006462
  - 0xA9.0xFE.0xA9.0xFE
  - 0xA9FEA9FE
  - 0x414141410A9FEA9FE
  - 0251.0376.0251.0376
  - 0251.00376.000251.0000376
- Tenk sscanf(“%i.%i.%i.%i”, &iIP1, &iIP2, &iIP3, &iIP4)

SQL Injection sårbarheter

- SQL sårbarheter forutsetter at input fra brukeren brukes til å bygge opp et SQL Statement

- SQL statements skal bygges opp av biblioteksfunksjoner, dette kalles «parameterized input»
- Aldri bygg opp en SQL statement ved å konkatenere strenger som inkluderer tekst fra brukeren
- “ SELECT * FROM Users WHERE Username = ‘ “ + txtUser.Text + “ ‘ AND Password = ‘ “ + txtPassword.Text + ““‘

- Hva skjer hvis txtUser blir satt til ‘ || 1 == 1; --
  - Resultat: SELECT * FROM Users WHERE Username = ‘’ || 1 == 1; --
- All bets are off :-)

Code/Command Injection sårbarheter

- Aldri bruk input fra en bruker for å bygge opp en kommando som skal kjøres på operativsystemet!

- Tenk et input felt «Kontroller om printer kjører» og en drop down liste med printernavn, en angriper ser på siden og oppdager at HTTP Request som sendes så oversettes navnene til en hardkodet liste over IP adresser
- Serveren kjører en lokal kommando «ping + list.IP_addr» i et CMD shell

- Angrepsstreng: «1.1.1.1 & format c:»
- Resultatet vil være at det kjøres to kommandoer:
  - Ping 1.1.1.1
  - Format c:

Hva er “input”

- En vanlig feil er å tenke på input som tekstfelter hvor brukeren kan taste inn data
- URL er også input
- HTTP Header felter er også input
- Cookie verdier er input – ikke stol på at det er den cookien du satt på siden din du får tilbake, det kan være et angrep

- Opplastinger av filer er det farligste
  - Innholdet i en fil kan være skadelig, enten for web serveren eller andre brukere
  - Fil NAVNET kan inneholde en script tag eller utføre SQL Injection :-)

```
> telnet www.targetsystem.com 80
GET/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucb
d3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u
9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0
```

Håndtere login – lagre passord

- Passord skal lagres backend som en hash
- IKKE lagre passord som klar tekst
- Ikke lagre passord kryptert
- Alltid bruk en salt verdi for å beskytte mot Rainbow tables
- Bruk PBKDF2 eller BCRYPT for å lagre passord; algoritmer som er laget for å ta lang tid – vanlige hash algoritmer er laget for å være kjappe…

- Det vanlige er å sende brukernavn og passord i klartekst, men over HTTPS (TLS 1.2)
- Ikke logg inn med hash (uten å være over TLS), det er sårbart ovenfor «send the hash» angrep
- For sikre applikasjoner anbefaler jeg å kryptere passordet

Logge en bruker inn på en tjeneste

- Hvis påloggingen skal være gyldig for mer enn ett miljø må du i utgangspunktet bruke et Single Sign On (SSO) system
- Om du logger deg på 2 eller 100 miljøer, bruker du den samme mekanismen
- SSO kan implementeres av JSON Web tokens eller en annen mekanisme mot en sentral database som AD eller LDAP

- I noen tilfeller er et alternativ å ikke ha brukerautentisering til den andre serveren, hvis det er mulig å sette opp en sikker mekanisme mellom de to serverne, vil brukeren da ikke ha tilgang til den andre serveren direkte – er vanligvis tilfellet mellom en frontend og en backend server, tilgang til backend vil da bli sikret med MTLS, OAuth2 + OpenID Connect eller lignende (ikke brukerens passord)

Logge inn på en backend database

- Passord for ressurser skal ikke være i kildekoden
- Utviklere bør ikke ha tilgang til produksjons passord

- I C har du hele operativsystemet til rådighet, og også Trusted Platform Module i CPU ☺

Brute force angrep

- En hver form for autentisering vil bli angrepet av en hacker, en hacker vil prøve flere typer passord angrep avhengig av hvor «lyst» han har til å komme inn i løsningen…
- En angriper starter først med de 1000 mest brukte passord, er passordet til EN av brukerne på systemet «password» eller «1234» så blir passordet funnet i løpet av noen minutter
- Så vil en angriper kjøre mer sofistikerte regel og hybrid angrep, en angriper kan ha scan mot brukere og passord kjørende i flere måneder før han «gir opp»
- En angriper som virkelig vil inn i en løsning vil til slutt prøve alle mulige kombinasjoner av bokstaver, tall og spesialtegn, det tar bare tid – men ofte for mange (tusen) år
- Problemet er at en angriper kan ha FLAKS

Brute force beskyttelse

- Hvis en brukers passord er forsøkt 3 ganger så vent i 10 sekunder med validering av passordet
- Hvis en brukers passord er forsøkt 4 ganger – 20 sekunder

  5 ganger – 40 sekunder

  6 ganger – 80 sekunder

  osv
- Hvis samme IP adresse har forsøkt å knekke passordet til forskjellige brukere 10 ganger – blokker all trafikk fra den IPen i 30 minutter

- Mer sofistikerte algoritmer har som formål å ikke være til hinder for en vanlig bruker som har glemt passordet, men stoppe brute force angrep (også distribuerte angrep) ved å øke tiden det tar å sjekke et passord

Offensiv Programmering

- Ekstreme tolkninger av Defensive Programming prinsippene tilsier at ingen funksjon / metode skal stole på noe data – alle data fra alle kilder skal antas å være skadelig kode
- Offensive Programming sier at man kun skal bruke Defensive prinsippene for data som mottas utenfra, men skal derimot stole «blindt» på alle data som kommer inne fra systemet / applikasjonen selv
- I Offensive Programming er det ansett som en fordel om en applikasjon krasjer ved interne feil i koden, for da er det lettere å finne feilen

- Dette betyr i praksis at hvis en database server kun skal kunne (by design) motta data fra en frontend server, da skal database serveren stole på alle data den får fra frontend serveren

That’s all folks

Høyskolen Kristiania

Excersises …

Høyskolen Kristiania

(Some slides have typos; my webserver use lowercase folders and files – and doesnt support mixed case… Or you can get them from Canvas.)

http://www.eastwillsecurity.com/pg3400/leksjon2.zip
http://www.eastwillsecurity.com/pg3400/leksjon3.zip
http://www.eastwillsecurity.com/pg3400/leksjon4.zip
http://www.eastwillsecurity.com/pg3400/leksjon5.zip
http://www.eastwillsecurity.com/pg3400/leksjon6.zip
http://www.eastwillsecurity.com/pg3400/leksjon7.zip
http://www.eastwillsecurity.com/pg3400/leksjon8.zip
http://www.eastwillsecurity.com/pg3400/leksjon9.zip

Exam has started, suggest you work on topics you feel will help you the most on your exam.

No exercises for today.