

Innocent Code

Kapittel 1: The Basics

Kapittel 2: Passing Data to Subsystems

**INF329, høsten 2005
Utvikling av sikre applikasjoner**

**Martin Lie
martin@mq.no
2005-11-02**

Kapittel 1: The Basics

HTTP

Sesjoner

HTTPS

HTTP

- Kort om web-[en](#) (verdensveven? akk!)
 - WWW ”oppfunnet” i 1989-90 av Tim Berners-Lee og gjengen
 - Tre sentrale spesifikasjoner: HTTP, HTML og URL-er
 - Sikkerhetsaspekter ved alle tre
 - HTTP minst kjent

HTTP

- Request response model
 - Koble til web-serveren
 - Sende en forespørsel
 - (Behandle forespørselen)
 - Sende svar
 - Avslutte forbindelsen

HTTP, request (GET)

- Eksempel:
Forespørsel til URL-en `http://example.com/`

```
GET / HTTP/1.1
```

```
Host: example.com
```

```
Accept: text/html, text/plain, image/*
```

```
Accept-Language: en
```

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US;  
rv:1.5a)
```

- Første linje kalles *request line* og inneholder: *method token*, *request URI* og *HTTP version*
- Så *request header*-linjer, fulgt av to linjeskift
- Til slutt *request data*

HTTP, response

- Eksempel: Svar på forrige sides forespørsel

```
HTTP/1.1 200 OK
```

```
Date: Wed, 02 Nov 2005 10:27:10 CET
```

```
Server: Apache/2.0.55 (Unix)
```

```
Content-Length: 84
```

```
Connection: close
```

```
Content-Type: text/html
```

```
<html>
```

```
<head><title>Test</title></head>
```

```
<body><p>Hello, world!</p></body>
```

```
</html>
```

- Første linje kalles *status line* og inneholder:
HTTP version, status code og *reason phrase*
- Så header-linjer, fulgt av to linjeskift
- Til slutt data

HTTP, request (POST)

- Eksempel:
Forespørsel til URL-en `http://example.com/`

```
POST /login.jsp HTTP/1.1
```

```
Host: example.com
```

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ...)
```

```
Referer: http://example.com/login.jsp
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 43
```

```
username=jalla&password=mekk&login=Logg+inn
```

- URL-encoding (eks: `& = %26`)

HTTP, idempotence

- Eks: header filer, pause knapp
- GET er ment å være idempotent (men ikke POST)
- Idempotence og caching

HTTP, sikkerhet

- Manipulering av request informasjon
 - Telnet (port 80)
 - Proxy

(Kapittel 3 har mye mer om dette)

- Referer- headeren
 - Kan ikke "brukes" ifm. sikkerhet
 - Link leakage (kap 6)

HTTP, cookies

- Liten informasjonskapsel (i klienten)
- Settes vha. en response header

```
Set-Cookie: Customer="78"; Version="1";  
Path="/"; Max-Age=1800
```

- Fjernes ved å sette *Max-Age* til 0
- En cookie uten *Max-Age* fjernes når nettleseren lukkes (typisk for sesjon cookies)
- Nettleser sender tilbake vha. en request header

```
Cookie: $Version="1"; Customer="78"; $Path="/"
```

HTTP, sesjoner

- Cookies gir mulighet for statefulness, men
 - Cookies er små
 - Cookies lagres på klienten
- Løsning: Sesjoner
 - Sesjonsid i en cookie
 - Resten av tilstanden på serveren
 - Sesjonsid = korttidspassord

HTTP, session hijacking

- Sesjonsiden er utsatt
 - Gjetting, kalkulering, brute-forcing
 - *Cross-site scripting* (XSS, kap 4)
 - Referer-headeren
- Risikobegrensning
 - Knytte id til IP
(men angriper kan ha samme IP som målet)
 - Knytte til *User-Agent*
(men dette kan gjettes av angriper)
 - Ugyldiggjøre sesjon ved mistenkeligheter
 - Endre sesjonsid ved hver request

HTTPS

- Mål
 - Hindre avlytting vha. kryptering
 - Hindre *man in the middle*-angrep (MITM) vha. sertifikater
 - (Autentisere klienter)
- Forbindelse etableres ved et *handshake*
 - Blir enige om krypto- og hash-algoritmer
 - Utveksler og validerer sertifikater (PKI)
 - Utveksler symmetrisk krypteringsnøkkel
 - Voila: Sikker kommunikasjon!

HTTPS, problemer

- Hva gjør *du* når du ser
 - "Sertifikat har utløpt for to måneder siden"
 - "Sertifikatet er signert av en ukjent autoritet"
 - ...og hva gjør onkelen din?
- skandiabanken.no vs. skandia-banken.no
- Kan man stole på *Certification Authorities* (CA)?
- Brukes *Certificate Revocation Lists* (CRL) eller *Online Certificate Status Protocol* (OCSP)?
- Standardene er sikre (nok), men hva med *folk*?

HTTPS, problemer

- HTTPS fungerer bra, med mindre
 - Brukerne ignorerer advarsler fra nettleseren
 - Nettleseren tillater dette
 - Brukeren faller for billige domenenavn-triks
 - CA-er gir ut feilaktige sertifikater
 - Nettleseren stoler på CA-er som ikke brukeren ville gjort
 - Nettleseren har bugs i SSL/TLS-implementasjonen
 - Brukerens maskin kontrolleres av en angriper

Kapittel 2:

Passing Data to Subsystems

SQL injection

Command injection

Løsninger

Subsystemer

- Eksempler
 - SQL-databaser
 - Operativsystemer
 - Biblioteker
 - Kommandotolkere (shells)
 - XPath-tolkere
 - XML-dokumenter
 - Gamle, utdaterte systemer
 - Nettlesere

Subsystemer

- Metadata
 - Tolkes som noe annet enn ren data
 - Problem når utviklere tror de sender ren data, men det tolkes som metadata
 - Utnyttelse avhenger av kontekst-bytte

SQL injection

- Eksempel 1

```
name = request.getParameter("name");  
query = "INSERT INTO User (name) VALUES ('" + name + "')";
```

- OK hvis navnet er Ola Nordmann
- Men hva med James O'Connor?

SQL injection

- Eksempel 2

```
user = request.getParameter("user");  
pass = request.getParameter("pass");  
query = "SELECT * From User WHERE user='" + user + "' "  
        + "AND pass='" + pass + "'";
```

- Noen problemer her?
- Hva skjer med følgende "navn":
jens' --
... WHERE user='jens' --' AND pass=''
- Løsning: Filtrere vekk "--"?

SQL injection

- Eksempel 3

- Hva skjer med følgende "navn":

```
jens' OR 'x'='y
```

```
... WHERE user='jens' OR 'x'='y' AND pass=''
```

```
... WHERE user='jens' OR FALSE
```

- Problem: Kontekst byte ved single quote
- Løsning: Escape alle single quotes?

SQL injection

- Eksempel 4

```
id = request.getParameter("id");  
query = "SELECT * From User WHERE id=" + id;
```

- Hva skjer med følgende "id":
 - 1; DELETE FROM User
... WHERE id=1; DELETE FROM User
 - 2; UPDATE User SET address=(SELECT pass FROM User WHERE user='jens') WHERE id=123
- <http://example.com/index.php?id=3;shutdown>
- Problem: Kontekst-bytte ved alle ikke-numeriske tegn

SQL injection, løsning

- Ikke gi ut informasjon i feilmeldinger
- Identifiser og nøytraliser alle mulige metakarakterer (avhenger av datatype)

```
function SQLString($s) {  
    $s = str_replace("'", "''", $s);  
    $s = str_replace("\\", "\\\\", $s);  
    return "'" . $s . "'";  
}
```

SQL injection, løsning

- Prepared statements

```
PreparedStatement ps = conn.prepareStatement(  
    "UPDATE User SET name=? WHERE id=?");  
...  
...  
ps.setString(1, name);  
ps.setInt(2, id);  
ResultSet rs = ps.executeQuery();
```

- Slipper å huske alle metakarakterer
- Parses kun én gang, så de kjører (bittelitt) raskere

Command injection

- Eksempel (Perl)

```
$user = $form{'user'};  
print `finger $user`;
```

- `$form{'user'}` er et brukernavn som vi får inn fra et web-skjema
- Dumt med følgende brukernavn:
`test; rm -rf /`
- Metakarakterer for typiske shell:
`" $ & ' () * ; < > ? [\] ` { | } ~ space tab`
- I tillegg: Ord og bokstaver, avhengig av posisjonen
- Konklusjon: Unngå bruk av bruker-input for kommandolinje-argumenter

Kjøreregler

- Ta alltid hånd om metakarakterer når du sender data til subsystemer
- Hvis mulig, send data og kontrollinformasjon separat
- God arkitektur – lagdeling, innkapsling, grensesnitt bruk, black-boxing
- Defense in depth