

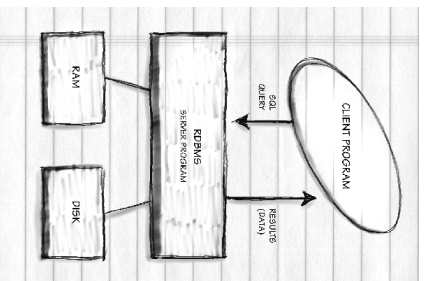
Databasestøttet Webpublicering, forår 2002

Forelæsning 9

Teknikker for databasestøttede web-sites og check af formvariablet

- Webservere med faste databaseforbindelser og scripting
- Java Applets
- Sikkerhed
- Opsætning og start af AOLserver på hug.it.edu
- Checking formvariables using ttpvar

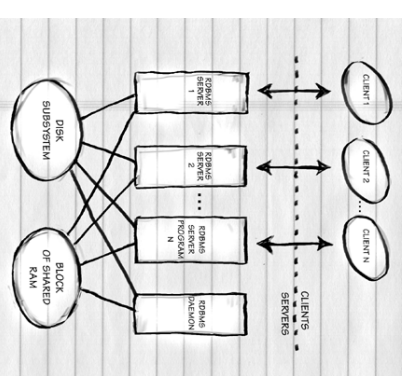
- #### Teknikker for databasestøttede web-sites
- CGI-scripts
 - Webservere med faste databaseforbindelser og scripting (f.eks. AOLserver med Tcl)
 - Java-applets



Billede: <http://www.arsdigita.com/books/panda/naphkin/12.1.gif>

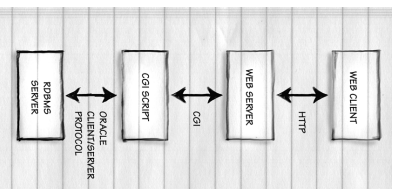
Generel Klient-Server Databasearkitektur

- Klienter kører potentielt på forskellige fysiske maskiner
- Databaseprocesser kører på samme fysiske maskine
- En databaseprocess etableres for hver klientforbindelse



Billede: <http://www.arsdigita.com/books/panda/naphkin/12.2.gif>

- #### CGI-scripts
- CGI står for Common Gateway Interface
- Fordele:**
- alle betydelige web-servere understøtter CGI-scripts
 - programmeringsprogs-afhængig
- Ulemper:**
- CGI-programmet startes som en ny operativsystemprocess for hver forespørgsel
 - En ny databaseserver-process startes for hver forespørgsel
 - CGI-programmet skal autoriseres af den nystartede databaseserver
 - Det tager tid at lukke CGI-programmet

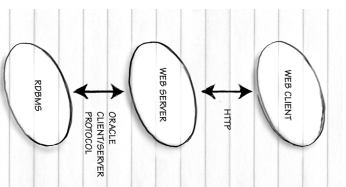


Langsom database-forbindelse med CGI.

Billede: <http://www.photo.net/wtr/chebook/nspk:log1AOLserver> der anvender faste database-

forbindelser:

Billede: <http://www.photo.net/wtr/chebook/nspk:in/12.3.gif>



Java Appliers

Fordele:

- Hurtig eksekvering på klienten (browseren), på trods af høj downloadtid
- Potentiale for god usability

Ulemper:

- Problemer med sikkerhed - ekstra åben port til databasen
- Problemer med licensen - da en port til databaseadgang skal være åben skal der typisk betales licens for antallet af samtidige brugere
- Potentielt mange databaseforbindelser og mange kørende databaseservere - hvornår kan man antage at en databaseforbindelse ikke bliver anvendt?
- Understøttes kun af få browsere - dette bliver bedre med tiden

Webservere med faste databaseforbindelser og scripting

Fordele:

- Hurtig opstart af fortolket program
- Hurtig tilgang til eksisterende databaseforbindelser

Ulemper:

- Stærkt knyttet til en bestemt web-server - tab af portabilitet

Sikkerhed

For at en hacker kan angribe data kræves:

1. Forbindelse til databasens IP-adresse
2. Kodeord til databasens server

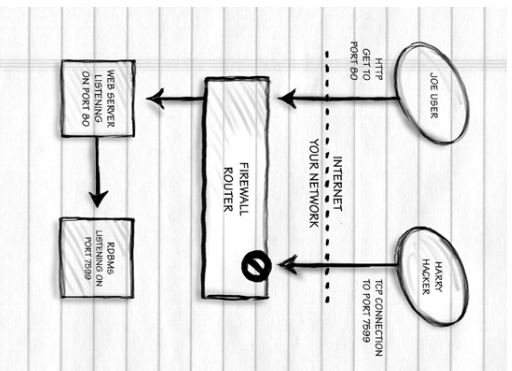
eller

1. Forbindelse til serverens IP-adresse
2. Kodeord til root, ved at hacke maskinen eller ...

For at undgå angreb:

- Flyt databasens server bag firewall og tillad kun forbindelse for webserver
- Flyt webserver bag firewall og tillad kun adgang udefra via port 80

Databasestruktur og Web-server gemt bag firewall



Billede: <http://www.arsdigita.com/books/panda/hapkin/12.5.gif>

```
Opstilling og start af AOLserver på hug.it.edu
Opstart af AOLserver fra UNIX-prompt:
cd /usr/local/aolserver/
bin/nsd-oracle -t nsd.mael.tcl

Konfigurationsfilen /usr/local/aolserver/nsd.mael.tcl:
Jeg har klippet en del i filen!
#-----
# mael's server
#-----
set home /usr/local/aolserver
set host [ns_info hostname]

ns_section "ns/parameters"
ns_param Home /usr/local/aolserver
ns_param serverlog /web/mael/log/server.log
ns_param pidfile /web/mael/log/nspid.mael
ns_param user mael
```

```
Konfigurationsfilen /usr/local/aolserver/nsd.mael.tcl, fortsat
ns_section "ns/server/mael"
ns_param directoryfile index.html
ns_param pageroot /web/mael/www
ns_param enabletcpipages on

ns_section "ns/server/mael/tcl"
ns_param library /web/mael/tcl

ns_section "ns/server/mael/module/nslog"
ns_param file /web/mael/log/access.log

ns_section "ns/server/mael/module/nssock"
ns_param port 8002
ns_param hostname $host
```

```
Konfigurationsfilen /usr/local/aolserver/nsd.mael.tcl, fortsat
Her følger konfiguration af Oracle i AOLserver.
ns_section "ns/db/drivers"
ns_param ora8 ora8.so

ns_section "ns/db/pool/main"
ns_param Driver ora8
ns_param Connections 2
ns_param User mael
ns_param Password #####
ns_param Verbose On

ns_section "ns/db/pools"
ns_param main main

ns_section "ns/server/mael/db"
ns_param Pools *
ns_param DefaultPool main
```

Checking Form Variables - The Easy Way

We have already seen how to check form variables:

- We use `info exists` to see if a form variable has been defined.
- If a form variable exists, then we normally use a regular expression to test the contents of the variable.

Until now, we have tested form variables by copying code into the scripts.

Using the TCL command `upvar` we can build a library of procedures checking both the existence and contents of form variables.

The library is a collection of procedures, each procedure testing one type of form variables, for instance integers or emails.

Given the library, lets call it `formvar.tcl`, we can have the scripts call one procedure in the library for each form variable to test.

Checking Form Variables - The Easy Way

If the script contains 10 form variables, then the script will contain only 10 procedure calls in order to test the form variables. Without the library, it would take maybe 50 lines of code.

On top of that, we can have the library automatically return an error page, in case a form variable fails a test.

To build the library we must

1. learn about `upvar`
2. build a general error page for our service, file `formvar.tcl`
3. build procedures testing for various types of form values, file `formvar.tcl`
4. call the procedures from the scripts

The upvar command (upvar.tcl)

Consider the procedure

```
proc add4 { var } {
  upvar $var v
  set v [expr {$v+4}]
}
```

and the program

```
set x 32
puts "x before add4: $x"
add4 x
puts "x after add4: $x"

set y 2
puts "y before add4: $y"
add4 y
puts "y after add4: $y"
```

The procedure `add4` accesses the variable with name `$var` in the enclosing procedure call. In `add4` that variable is named `v`.

In the first call, the variable `x` is accessed in `add4` as `v`.

In the second call, the variable `y` is accessed in `add4` as `v`.

This works, even if `add4` is located in a separate file!

A Standard Error Page

We write the library procedures in file `formvars.tcl`

You must define a standard error page, such that all errors are reported consistently.

A simple error page may use the well known back key.

Given an error message `error_msg` we may use the following outline:

```
We had a problem processing your entry:
• error_msg
Please back up using your browser, correct it, and resubmit your entry.
Thank you
```

```
proc return_page { title body } {
  ns_return 200 text/html "
<html>
<title> $title </title>
<body>
  $body
</body>
</html>"
}
```

Code for a very simple error page

```
proc return_error { err_msg } {
    set title "Problem with Your Input"
    set body "We had a problem processing your entry: <ul>"
    append body " <li>${err_msg}\n"
    append body " </ul>"
    Please back up using your browser, correct it, and
    resubmit your entry.
<p>
Thank you.
"
return_page $title $body
}
exit
}
```

Checking Email, file formvars . tcl

Because we are only writing a test procedure for each form type once, we can actually put some effort into reporting good error messages.

A procedure chk_email for checking an email:

```
proc chk_email {form_var {error_msg ""}} {
    upvar $form_var email
    if {[info exists email]} ||
        ! [regexp "\[^\t ]+@[^\.\[\]\+\@.\[^\n ]+\]" $email]} {
        if {[empty_string $error_msg]} {
            if {[info exists email]} {
                set error_msg "You must enter an <b>email</b> address"
            } else {
                set error_msg "You must enter a valid <b>email</b> address -
                    '$email' is not one!"
            }
        }
        append error_msg "<blockquote> Example email addresses are
        <ul>
        <li>user@supernet.com
        <li>FirstLastname@very.big.compagny.com
        </ul></blockquote>"
        return_error $error_msg
    }
    return
}
```

Check Email

In `chk_email`, the variable `email` accesses the form variable `$form_var` in the enclosing procedure call, that is, the script.

In the script, where we check the form variable `email`, we write

```
source "/web/nh/www/formvars/formvars.tcl"
set_form_variables 0
chk_email email
# Now we can safely use the form variable email
```

Try http://nhug.it.edu:8077/formvars/chk_formvars.tcl

Checking a range [a.....b], file formvars . tcl

A procedure checking that two form variables `a` and `b` forms a range `[a...b]`, where `a < b`

```
proc chk_range {form_var1 form_var2} {
    error_msg "You must enter a <b>range</b> \[a,...,b\], where a < b" } {
    upvar $form_var1 first
    upvar $form_var2 last
    if {[info exists first]} ||
        ! [regexp {(0|[1-9]([0-9]*))}$ $first]} ||
        ! [info exists last]} ||
        ! [regexp {(0|[1-9]([0-9]*))}$ $last]} ||
        ($last - $first < 0) {
        return_error $error_msg
    }
    return
}
```

In the script we write:

```
source "/web/nh/www/formvars/formvars.tcl"
set_form_variables 0
chk_range a b
```

#Now, I can safely use `a` and `b`

where the two form variables are named `a` and `b`.

Introduktion til Øvelse 8

Konstruktion af et web-baseret projektbørs

Se <http://www.it.edu/courses/W2/F2002/Lb/Lb8.html>