

Kap. 7

Enemies of Secure Code

Daniel Lundekvam

daniell@ii.uib.no

14.11.2005

INF 329 Utvikling av sikre applikasjoner



Kap. 7 Enemies of Secure Code

- Fokus på sikkerhet er større en noen gang
- Likevel utvikles det web-applikasjoner med store sikkerhets hull
- Hvorfor?



Uvitenhet (1)

- Den største trusselen mot sikker kode er programmererens mangel på kunnskap
- Forfatteren gjorde et eksperiment og 'skummet' mange data bøker i en bokhandel:
 - Bøker om programmering av e-handels løsninger
 - De fleste nevnte ikke ordet sikkerhet
 - Fant ikke en bok som ikke ga kode eksempler med store sikkerhetshull
 - Bøker om sikkerhet
 - Mye om infrastruktur som kryptering, brannmurer osv
 - Ingen fokus på kode som en viktig del av den totale sikkerheten
- Bøkene om programmering tok ikke opp sikkerhet, og sikkerhets bøkene tok ikke opp koding.
- Dette har bedret seg siden
- Vanskelig å lære seg å lage sikker kode, når det ikke finnes litteratur om det



Uvitenhet (2)

- Sikkerheten må være en integrert del av utviklingsprosessen, og ikke noe man tar seg av på slutten
- Hvis man hopper over sikkerheten underveis, kan man måtte gå tilbake og lukke sikkerhets hull. Dette vil ta mye langer tid.
- Hvis vi ta opp kodesikkerhet tidlig i planleggingen, f.eks ved å lage et godt rammeverk, kan man gjøre det vanskelig å introdusere usikker kode.



Rot

- De fleste er late.
- Programmerer fort og for noe som ser ut til å virke, så jobber videre med en annen del av programmet.
- Resultat blir et system der alle delene er tett vevd sammen. Hver del er avhengig av de andre og det er ikke noe klart grensesnitt mellom delene.
- En endring i en del kan fort føre til at man også må endre andre deler.
- Det kan da være vanskelig å identifisere hvilke deler som da må endres, og hvordan man kan endre disse uten å måtte foreta ny endringer andre steder osv...



Rot

- Eksempel på ekstrem latskap:

```
If Request.form("foo") = "bar" Then  
    x = "bar"
```

```
    [Thirty lines of code operating on the variable 'x']
```

```
Else
```

```
    x = "gazonk"
```

```
    [The same thirty lines of code operating on the variable 'x']
```

```
End If
```

- Programmererne hadde funnet et sikkerhets hull og fikset det. Men de hadde bare fikset hullet i den delen som der parameteren "foo" er lik "bar", som var det mest vanlige i denne applikasjonen.
- En angriper kunne fremdels forandre "foo" parameteren og tvinge programmet til å utføre Else delen, og så utnytte sikkerhets hullet.



Rot

- Latskap virker ikke i det lange løp. Det gir oss problemer som må fikses. Problemer som ikke er lette å fikse og ofte kommer mot slutten av prosjektet.
- "best practices"
 - *Unngå duplisert kode*
 - *Isoler funksjonalitet*
 - *Begrens lengden av kode-blokker*
 - *Begrens funksjonaliteten til funksjoner*
 - *Begrens nødvendigheten av kommentarer*
 - *Begrens levetiden til variabler*
 - *Lag enhets tester*



Best practices

- ***Unngå duplisert kode***
 - Vanskeligere å vedlikeholde
 - Bug-fiksing blir gjort der man oppdager feilen og ikke i andre deler av programmet med 'samme' kode
- ***Isoler funksjonalitet***
 - F.eks. all database kommunikasjon behandles et sted
 - Slipper å endre mange steder hvis man finner ut at man er sårbar for f.eks. SQL Injection
 - Gjør det lettere å bytte ut komponenter
- ***Begrens lengden av kode-blokker***
 - En funksjon bør ikke være større enn at man kan se hele i et skjermbilde
 - Lettere å lese
 - Forfatterens eget ordtak: "Hvis funksjonen er over 20 linjer, bør man vurdere å dele den opp. Hvis funksjonen er over 100 linjer skulle man ha delt den opp for lenge siden."



Best practices

- ***Begrens funksjonaliteten til funksjoner***
 - Bruk beskrivende navn på funksjonene
 - Navn skal beskriver hva som skjer, ikke hvordan
- ***Begrens nødvendigheten av kommentarer***
 - Koden bør være selvforklarende
 - Når koden forandres glemmer man ofte å forandre kommentarene
 - For å få en selvforklarende kode må man:
 - Ha gode beskrivende navn på variabler og funksjoner
 - Bruke navngitte konstanter
(f.eks: MAX_NAME_LENGTH istedenfor 25)
 - Korte og lettfattelige kode-snutter



Best practices

- ***Begrens levetiden til variabler***
 - Globale variabler gjør ofte koden vanskelig å lese
 - Hvis en funksjon kan forandre store mengder med eksterne variabler kan det bli veldig vanskelig å få et klart bilde av hva funksjonen gjør
 - Med fler-trå programmer kan man risikere at en global variabel er forandret midt i en operasjon (race conditions)
- ***Lag enhets tester***
 - Lag tester for alle funksjoner og kode-snutter
 - Etter en forandring kjør alle testene og identifiser hvilken del av koden som eventuelt ikke fungerer lenger
 - Forandringer i enhetstestene kan også hjelpe til å finne bugs



Rot

- Hvis man ikke gjør noe for å unngå 'rot' vil utviklingen gå raskt i starten, for så å gå saktere og saktere jo større programmet blir
- Prøver man å unngå rotete-kode vil utviklingen gå saktere i starten (pga bruker mer tid på planlegging), men vil et mer stabilt tempo gjennom hele prosjektet.



Deadlines

- Hvis sikkerhet er noe som man ikke tenker på før til slutt, er det lett å ofre den når deadline nærmer seg
 - Kunden vil lett merke om det mangler funksjonalitet, men ser kanskje ikke om sikkerheten er mangelfull
- Hvis sikkerheten er en del av prosessen kan tidsfrister også skape problemer
 - Når deadline nærmer seg uten at vi klarer å tilfredstille kravene, setter vi opp tempoet i programmeringen
 - Dette kan igjen fører til rotete-kode og usunne snarveier



Deadlines

Eksempel på snarvei:

- Sikker kode:

```
PreparedStatement ps = con.prepareStatement(
    "INSERT INTO Node (id, title, abstract, text) "
    + "VALUES (?, ?, ?, ?) " );
...
ps.setInt(1, id);
ps.setString(2, title);
ps.setString(3, abstract);
ps.setString(4, text);
ps.executeUpdate();
```

- Snarvei:

```
conn.createStatement().executeUpdate(
    "INSERT INTO Node (id,title,intro,text) "
    + "VALUES (" + id + ",'" + title + "',"
    + "'" + intro + "','" + text + "');" );
```



Selgere

- Markedsavdelingen i et firma er ofte de som dikterer kort deadlines
- De vet at de som lover å bruke minst tid og minst penger oftest for kontrakten
- Derfor kutter de ned så mye som mulig og overlater alle problemene dette fører med seg til programmererne.



Oppsummering Kap. 7

- Hva fører til usikker kode:
 - Uvitenhet hos programmereren
 - Rot i koden
 - For korte deadlines
 - Overivrige selgere
- Det er viktig å forstå at hvordan koden er skrevet, er en viktig del av den totale sikkerheten i en applikasjon

Kap.8: Oppsummering av Regler for Sikker Koding



Daniel Lundekvam

daniell@ii.uib.no

14.11.2005

INF 329 Utvikling av sikre applikasjoner



Regel 1: Ikke undervurder angriperne

- Hackere er ofte mer oppfinnsomme enn web programmerere, når det gjelder angrep
- Ikke tenk: "Hvis ikke jeg finner feil her, så finner ingen andre det heller"



Regel 2: Bruk POST request når handlinger har bivirkninger

- Ved bruk av GET kan brukeren gjøre samme request flere ganger
 - f.eks. ved å trykke "back" i nettleseren
- Dette er ikke holdbart når handlingen forandrer noe, som f.eks overføre penger i bank



Regel 3: På server siden er det ingenting som heter klient-side sikkerhet

- Alt som kommer fra klient-siden kan ha uventede verdier
- HTTP headere, cookies o.l. kan bli endret av angriperen
- Klient-side skript kan bli forandret eller bli omgått
- Java Applets kan bli dekompilert eller erstattet med andre programmer



Regel 4: Bruk aldri Referer header for autentisering eller autorisering

- Referer header kommer fra klient-siden og er dermed under brukerens kontroll
- En angriper kan enkelt endre en header for å omgå sikkerhets tiltak
- Mange brukere stiller også inn nettleseren til ikke å sende Referer header



Regel 5: Generer alltid en ny session ID når brukeren har logget inn

- Det blir ofte generert en session så snart noen besøker en web side. Denne ID kan så bli gitt flere privilegier etter hvert
 - f.eks. brukeren logger inn
- Det kan være mulig for en angriper å få tak i session ID'en når en bruker navigerer siden (f.eks med Session Fixation)



Regel 5: Generer alltid en ny session ID når brukeren har logget inn

Eksempel: Session Fixation

- En angriper besøker først mål-web siden og får en ny session ID, f.eks: ABC123
- Angriperen lurer offeret til å følge angriperens link til siden.
 - <https://bank.example.com/login.php?PHPSESSID=ABC123>
- Hvis mål-web siden støtter session IDer i URLen vil offeret nå bruke samme session ID som angriperen allerede har.
- Når offeret så logger inn blir denne sessionen autentisert som offeret, og angriperen får tilgang.



Regel 6: Gi aldri detaljerte feilmeldinger til klienten

- Feil meldinger kan gi viktig informasjon om hvordan applikasjonen fungerer.
 - Kode linje feilen oppstod
 - Feil meldinger fra subsystem
 - web server, OS osv..
- Feilmeldinger generelt er et tegn på svakhet. En angriper som klarer å fremprovosere feilmeldinger vet at systemet overlater håndtering ugyldige input til andre lag.



Regel 7: Identifiser alle mulige metacharacters til et subsystem

- Før man sender data til et subsystem bør man være sikker på at man vet om alle metacharater'ene til subsystemet
- Lignende subsystemer fra forskjellige leverandører kan gjøre ting for litt forskjellige måter
 - Noen database servere gir '/' inne i en streng en spesiell mening, mens andre ikke gjør det.



Regel 8: Håndter alltid metacharacters før de sendes til et subsystem

- Metacharacter'er lager problemer uansett om de kommer fra en ekstern bruker eller fra indre deler av applikasjonen
- Man må alltid behandle metacharacter'er før data sendes til subsystem, uansett hvor de kommer fra
- Ellers: Kan få rare feil meldinger og være sårbar mot andre ordens SQL Injection angrep
- Eks:
Innsetting navn på en person i en SQL tabell
Navnet: "O'Conner" eller "'; DELETE FROM Person "
Her er det klart at ' tegnet må vært behandlet på forhånd



Regel 9: Når det er mulig, ikke send data og kontroll informasjon sammen

- Hvis subsystemet vi kommuniserer med støtter data sendings mekanismer som tillater at bare data sendes, bør vi bruke disse for å unngå metacharacter problemer
- Eks:
 - Prepared statements for SQL
 - DOM for XML



Regel 10: Se opp for flerlags oversetting

- Noen ganger er det vi ser på som et subsystem bare en sti til et annet subsystem
- Dette kan føre til at man trenger ytterligere metacharakter håndtering

Eks:

- Når vi kaller et 'command shell' for å kjøre et eksternt program eller lager en SQL streng som kan bli brukt i en LIKE klausul
- Både 'command shell' og SQL strengen trenger metacharakter håndtering, men mål programmet og LIKE-gjenkjenneren kan trenge ytterligere metacharakter håndtering



Regel 10: Se opp for flerlags oversetting

- Eksempel

```
SELECT * FROM Usr WHERE RealName  
LIKE 'A%'
```

- SQL parseren vi lese strengen A%, og sende den videre til LIKE matcher'en. I denne vil prosent-tegnet bety "en vilkårlig sekvens av tegn"



Regel 11: Forsøk å alltid ha en backup sikkerhets mekanisme

- Siden mennesker og teknologi ikke virker perfekt hele tiden vil ting av og til gå galt
- Sikkerhet:
Vi bør alltid se etter muligheter til å legge inn en backup som stopper en angriper hvis en sikkerhets mekanisme feiler
- Eks:
Vi gjør alt vi kan for å hindre SWL Injection, men vi setter like vel database serveren opp til ikke å gi skrive-tilgang med mindre det er strengt nødvendig



Regel 12: Ikke stol blindt på API dokumentasjon

- Noen API dokumentasjoner kan feilaktig gi inntrykk at at sikkerheten er tatt hånd om
- Hvis det står vage utsagn om mulige returverdier fra en funksjon, ikke stol på det.



Regel 13: Identifiser alle kilder for input til applikasjonen

- Input er være mye mer en det brukeren skriver på tastaturet sitt
- Verdier fra "hidden" felt, checkboxer, cookies, HTTP headere osv kan være manipulert av brukeren for å utnytte sikkerhets hull
- Applikasjonen kan også få input fra filer, databaser osv.
- I noen tilfeller er ikke disse sikrere enn web-klienten og trenger validering



Regel 14: Vær oppmerksom på "the invisible security barrier": Valider alle data

- Vær sikker på at du har en god forståelse av når data er på klienten og når data er på serveren
- Data som er sendt fra serveren til klienten og tilbake har passert 'the invisible security barrier'
 - Eks: verdier fra "hidden" felter
- All data som sendes fra klienten er i usikkert, og må valideres



Regel 15: Bruk Whitelisting fremfor Blacklisting ved filtrering

- Blacklisting:
Identifiserer ugyldig data – og fjerner disse
- Whitelisting:
Identifiserer gyldige data – og fjerner alle andre
- Problem med blacklisting:
 - Filtrering ikke ut ukjente data
 - Hvis listen ikke er fullstendig er applikasjonen sårbar for angrep



Regel 16: Ikke forsøk å endre på ugyldig input for å gjøre den gyldig

- Vanskelig å gjøre bra
- Angripere kan finne en vei rundt 'masseringen' ved å legge opp angrepet etter masserings-algoritmen

```
http://www.bank.example.com/info.asp?file=info1.txt
```

```
http://www.bank.example.com/info.asp?file=../default.asp
```

```
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
```

- Ved bruker-generert invalid input:
Gi en feilmelding og be bruker prøve på nytt
- Ved server-generert invalid input:
Gi en enkel feilmelding og logg hendelsen



Regel 17: La applikasjonen lage sin egen log

- Web servere lager normalt en access log som inneholder informasjon om HTTP-requester og lignende
- En dynamisk web applikasjon vet mye mer om resultatene av requester, input valideringen, om inn loggete brukere og hva som er tillatt og ikke
- Applikasjonen bør derfor lage sin egen logg



Regel 18: Bruk aldri skript på klint-siden til sikkerhet

- Skript som kjøres i brukerens nettleser kan enkelt modifiseres
- Ikke bruk klient-side skript for validering, autorisering, autentisering eller andre sikkerhets mekanismer
- Aldri gi passord eller andre hemmeligheter i script som gis til klienten



Regel 19: Ikke referer resurser direkte i server-generert input

- For server-generert input, som data fra "hidden" felter, option tags og andre elementer som ikke modifiseres av bruken direkte
 - Ikke referer resurser direkte, men heller med f.eks en index

```
<a href="showlog.php?file=access_log">access log</a>  
<a href="showlog.php?file=error_log">error log</a>
```

```
$filename = $_GET["file"]  
Readfile("/var/log/httpd/" . $filename);
```

```
../../../../etc/passwd
```

```
$fileid = $_GET["file"];  
if ($fileid == "A")  
    $filename = "access_log";  
elseif ($fileid == "E")  
    $filename = "error_log";  
Else  
    exit("invalid file ID");
```



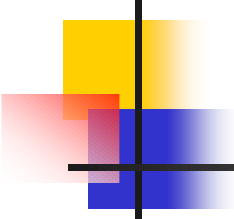
Regel 20: Gi så lite intern informasjon til klienten som mulig

- Data som sendes til klienten kan bli modifisert av en angriper før det sendes tilbake
- Ofte vanskelig å huske å revalidere data, bedre å sende så lite slik data som mulig



Regel 21: Ikke anta at requester vil komme i en spesiell rekkefølge

- En angriper kan oppe over en request eller gjøre samme request flere ganger
- Kan bruke "ute av rekkefølge" request for å forbigå sikkerhets tester



Regel 22: Filtrer all data før de inkluderes i web applikasjonen, uansett kilde

- For å unngå alle typer Cross-site Scripting angrip, bør all data som ikke skal inneholde markup bli kjørt igjennom et HTML-filter før det blir inkludert i web-siden
- Om mulig bør dette gjøres automatisk



Regel 23: Bruk eksisterende krypto- algoritmer, ikke lag din egen

- Du klarer ikke lage noe bra nok selv.
- Bruk det som finnes



Regel 24: Ikke ha passord i klartekst

- Folk bruker samme passord til flere ting
- Hvis en angriper får tilgang til passordene fra en side, kan han prøve passord/brukernavn kombinasjonen på andre, mer kritiske sider
- Bedre å lagre hash-signaturen av passord i databasen



Regel 25: Bruk aldri GET for hemmelig data, inkludert session IDer

- Parametere fra GET kan havne i Referer header som sendes fra nettleseren
- Dette kan føre til at man gir sensitiv informasjon andre

Eks:

- Forfatteren hadde konfigurert serveren sin til å logge referer headeren til alle innkomne requester.

`http://www.discuss.example/index.cgi?name=johndoe&passwd=Madonna`

- GET parametere kan også bli sett i web server og proxy logger og er tilgjengelig i nettleserens historie



Regel 26: Anta at server-side kode er tilgjengelig for angripere

- Det finnes mange måter for angripere å få tilgang til server-side kode
- Kompilering eller på annen måte tåkelegging av koden er ikke nok
- Denne regelen skal hindre oss i å tenke at:
 - "ingen vil noen gang finne ut av denne lille snarveien"
 - "ingen vil noen gang finne navnene på tabeller og attributter i databasen, så SQL Injection er ikke noe problem"



Regel 27: Sikkerhet er ikke et produkt; det er en prosess

- Ikke ta opp sikkerhet som et punkt på slutten av utviklingen
- Sikkerhet er en del av applikasjonen
- For hver linje man koder må man spørre seg selv:
 - "Hvordan kan denne linjen bli misbrukt?"
 - "Hvordan kan dette problemet løses, slik at angripere ikke blir fornøyd"

Appendix A: Bugs i Web Servere





Bugs i Web Servere

- Når det oppdages bugs i "hyllevare"-programmer, gir leverandøren ut en *patch* som skal tette dette hullet
- System administratoren må så sørge for at sin programvare til enhver tid er oppdatert.
- Hvis ikke dette blir gjort, kan angripere utnytte disse feilene



Jan 2000: Bug i Microsofts Index Server

- MS Index Server er en komponent som skal gjøre det enkelt å tilby søkefunksjonalitet i web hierarki
- Index Serveren skal søke igjennom synlige web sider, ikke skript-kode
- Men med denne URLen var det mulig å få listet ut innholder i skript filer:

```
http://someplace.example/null.htw?CiWebHitsFile=  
/default.asp%20&CiRestriction=none&CiHiliteType=Full
```



“Unicode bug” i Microsoft IIS

- Tillot angripere å kjøre hvilket som helt program på serveren

```
http://someplace.example/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\
```

- Sti til Windows NT sin kommando tolker:
 - `winnt/system32/cmd.exe`
- Etter URL dekoding regler får kommando tolkeren parametrene:
 - `/c dir c:/`
- Så den lister ut innholder på serverens: `c:`
- For å hindre at angripere går ut av scripts katalogen, sjekket de at URLen ikke inneholdt `../`
- URL dekoding: `%` og de to påfølgende heksadesimale siffer skal byttes ut med en enkel byte, med innhold gitt av de heksadesimale snifferne



“Unicode bug” i Microsoft IIS

- Med tegnsettet: ISO-8859-1
 - Byte verdien c0 blir tegnet: À
 - Byte verdien af blir tegnet: -

Som gir: /...À-.../

- OS bruker UTF-8 enkoding
Der kan et tegn kan representeres med flere byte
hvis den førte bytens verdi $\geq c0$
- Med UTF-8 blir bytesekvensen c0af dekodet til /
Dette gir: /.../.../



Bug i WebLogic fra BEA

`http://someplace.example/index.js%70`

- %70 dekodes til 'p'
- Gjør ikke URL dekoding før sjekk av request typen
- Ble ikke sendt til JSP tolkeren, men til default handler
- Et sted her er URL dekodingen gjort, og man får dermed listet ut innholder i filen index.jsp



Appendix B: Pakke sniffing



Pakke sniffing

- Pakke sniffing er en operasjon der en inntrenger kan få tilgang til hemmeligheter, som passord og session ID'er, ved å snappe opp informasjon mens den passerer gjennom nettverket.
- Pakke sniffing kan brukes mot protokoller som sender ukryptert data over nettverket, som HTTP, telnet, FTP osv.
- I gamle dager:
 - Alle maskiner i et nettverk koblet til samme kabel
 - Hver pakke når alle maskinene på nettverket
 - Nettverkskortet vil vanligvis overse de pakkene som ikke har matchene adresse, men det er mulig å sette det til å ta imot alle pakker
- Nå bruker vi hub/switch:
 - Hub: Fungerer som i gamle dager og sender alle pakker til alle
 - Switch: Har lister over hvilke adresser som er assosiert med hvilken port, og sender bare pakken ut riktig port
 - Hvis disse listene blir 'overflowed' vil de fleste switcher bli redusert til å fungere som en hub



Man-in-The-Middle Attack

- Et slik angrep innebærer at angriperen er mellom offeret og den offeret kommunisere med. Angriperen spiller rolle som offer ovenfor den andre parten, og omvendt. Angriperen kan endre informasjonen og sender den videre begge veier.
- På lokalt nettverk: ARP spoofing
 - Angriperen gjør at en maskin assosierer en IP adresse med feil MAC adresse, slik at pakken blir sendt til angriperens maskin
- På Internett: DNS spoofing
 - Angriperen forårsaker et falskt DNS svar, slik at offerets maskin mapper destinasjons domenet til angriperens IP



Man-In-The-Middle med HTTPS

- Et av målene med SSL/TLS er å beskyttelse mot man-in-the-middle angrep.
- Beskyttelsen er basert på sertifikater som er gitt av en Certificate Authority (CA).
- I SSL/TLS handshake sjekker man at begge parter har gyldig sertifikat.
- Dette systemet er sett på som skuddsikkert, og fullt ut beskyttet mot at noen kan utgi seg for å være server
- I 2000 kom det et angrep som gjorde man-in-the-middle angrep mot HTTPS mulig
- Det var ikke et angrep svakheter i SSL/TLS protokollen, men det svakeste leddet i sikkerhets-kjeden: brukeren
- Angrepet lagde et falskt server- sertifikat, for å kunne signere SSL/TLS-handshake informasjonen



Man-In-The-Middle med HTTPS

- Siden dette ikke er et gyldig sertifikat gir nettleseren opp en advarsel til brukeren.
- Angrepet virker hvis bruken trykker på "OK" eller "Fortsett" knappen for å få advarselen vekk.
- Hvor mange vanlige brukere forstår meningen bak advarsler som: "The name of the security certificat dose not match the name of the site"?



Appendix C: Sending av HTML formaterte E-mail med falsk avsender adresse

- Noen mail-klienter støtter både falsk avsender og HTML formatering
- Ellers kan man bruke telnet

```
HELO badguy.example
MAIL FROM: st.claus@northpole.example
RCPT TO: victim@somesite.example
DATA
From: st.claus@northpole.example
To: victim@somesite.example
Subject: Cool joke
MIME-Versjon 1.0
Content-Type: text/html
```

```
<form name="f" action=http://www.badguy.example/vote.asp method="post">
<input type="hidden" name="alt" value="2" />
</form>
<script>document.f.submit()</script>
.
QUIT
```