Web Bezbednost

Java Web

ITAcademy

Uvod u bezbednost

Tipovi napada

- XSS
- SQL Injection
- Remote code injection
- Account riding
- Session Request Forgery
- …

Tipovi bezbednosnih tehnika

- Klijentska validacija
- Serverska validacija
- Validacija u bazi podataka
- Pojam crne i bele liste

https://www.owasp.org/index.php/Main_Page

Bezbednost web formi

- Forma je dodirna tačka između korisnika i aplikacije, zbog toga je najčešće prva na udaru napadača
- Većina tehnika za narušavanje bezbednosti aplikacije upravo je vezana za forme
- Formu treba obezbediti na dva mesta.
  - Manje bitna – klijentska validacija
  - Veoma bitna – serverska validacija

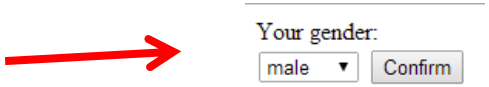
Spoofed form

- Spoofed form je situacija u kojoj korisnik šalje formu na server, ali ne u originalnom, već u izmenjenom obliku
- Server tretira dolazne podatke kao ispravne podatke sa forme, ali oni se logički ne uklapaju u kontekst sistema i eksploatišu ga

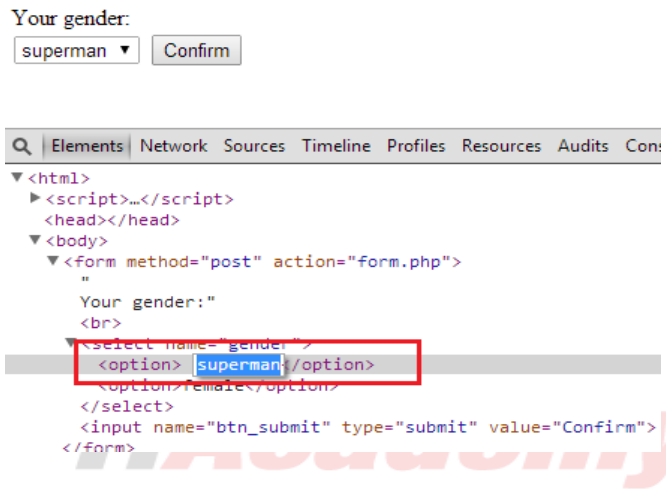
Spoofed form - primer

- Recimo da postoji jednostavna forma (na primer za unos pola):

```
<form method="post" action="form.php">
Your gender:<br />
<select name="gender">
<option>male</option>
<option>female</option>
</select>
<input name="btn_submit" type="submit" value="Confirm" />
</form>
```



- Korisnik može otvori konzolu browsera i zameni podatke u formi
- Takođe, korisnik može kopirati komletan html na sopstveni računar i odatle aktivirati formu



Spoofed forms – napredniji primer

Korisnik može napraviti aplikaciju koja će rukovati direktno sa soketom na serveru i na taj način u potpunosti kontrolisati sadržaj forme, odnosno, kompletnog zahteva

```
Socket s = new Socket("localhost", 8080);
OutputStreamWriter oos = new OutputStreamWriter(s.getOutputStream());
BufferedWriter bw = new BufferedWriter(oos);
InputStreamReader isr = new InputStreamReader(s.getInputStream());
BufferedReader br = new BufferedReader(isr);

String params = "username=pera peric";

bw.write("POST /WebApplication1/process HTTP/1.1\r\n");
bw.write("Host: localhost\r\n");
bw.write("Content-type: application/x-www-form-urlencoded\r\n");
bw.write("Content-length: "+params.length()+"\r\n\r\n");
bw.write(params);
bw.flush();

String line;
int bt;
while((bt=br.read())!=-1) {
    System.out.print((char)bt);
}

bw.close();
br.close();
```

Spoofed forms - rešenja

- Provera referera-a (nije dovoljno)
- Tokenizacija
- Filtracija svih podataka (nikada ne verujemo podacima koji dolaze spolja)

ITAcademy

Provera referrer-a

- Referer se nalazi među zaglavljima zahteva
- Na osnovu ovog podatka lako možemo uvesti bezbednosnu proveru referera prilikom obrade forme:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {

        String referrer = request.getHeader("referer");
        if(referrer==null||!referrer.equals("http://localhost:8080/WebApplication1/")){
            out.print("INVALID REQUEST");
            return;
        }

        out.print(request.getParameter("username"));
    }
}
```

Provera referrer-a

- Proverom referera oslobodili smo se mogućnosti da korisnik formu pošalje sa sopstvenog računara
- Provera referera se takođe lako prevazilazi ukoliko napadač komunicira direktno sa serverom pomoću soketa, jer je referer samo još jedno zaglavlje u http zahtevu

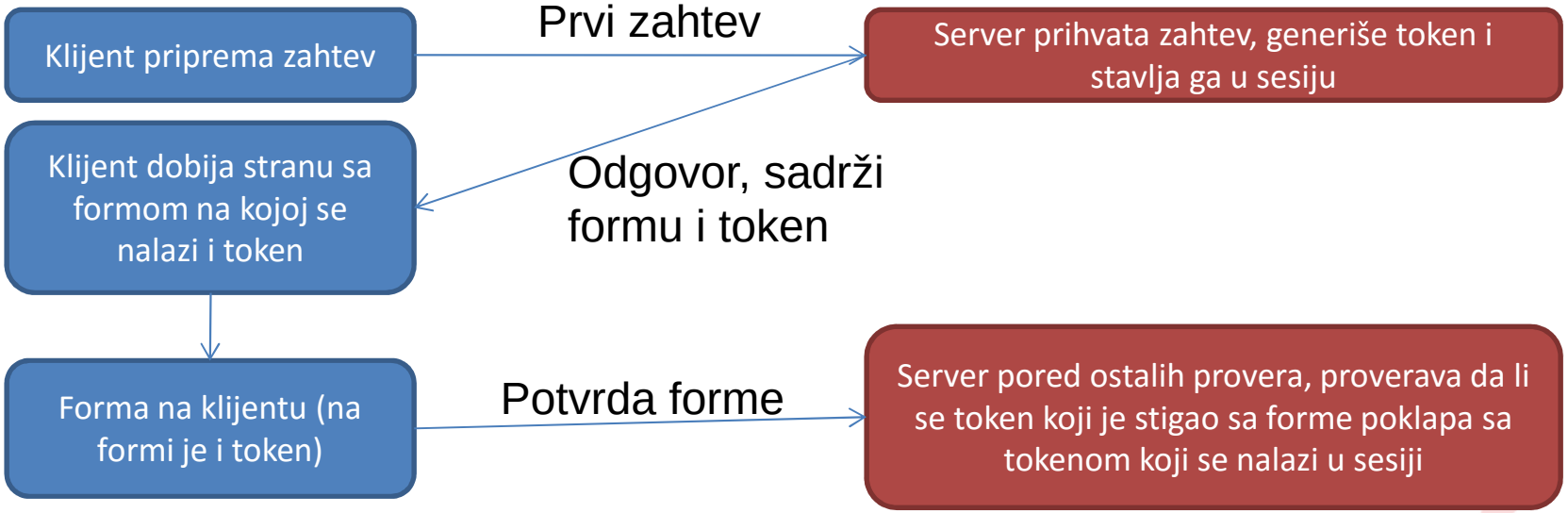
```
bw.write("Referer: http://localhost:8080/WebApplication1/\r\n");
```

Primer – provera referera

```
String referrer = request.getHeader("referer");
if(referrer==null||!referrer.equals("http://localhost:8080/WebApplication1/")){
    out.print("INVALID REQUEST");
    return;
}
```

Tokenizacija

- Da bi dodatno obezbedili formu, možemo korisniku poslati token, a zatim tražiti od njega taj token prilikom slanja forme.
- Korisnik ovo može da prevaziđe, ali u svakom slučaju za svaku formu koju pokuša da izvrši mora dobaviti novi token pa ga tako možemo pratiti ukoliko je potrebno



Tokenizacija

```
<form action="process" method="post">
    <%
        String guid = UUID.randomUUID().toString();
        session.setAttribute("token", guid);
    %>
    <input type="hidden" name="token" value="<%=guid%>" />
    <input type="text" name="username" />
    <input type="submit" value="login" />
```

Provera tokena

```
String sessionToken = (String)request.getSession().getAttribute("token");
String requestToken = request.getParameter("token");

if(
        sessionToken==null ||
        sessionToken.isEmpty() ||
        !sessionToken.equals(requestToken)
   ){
    out.print("INVALID TOKEN");
    return;
}
```

Cross Site Scripting (XSS)

- ***Cross-site scripting je*** emitovanje klijentski definisane skripte mimo “znanja” i “želje” same aplikacije

- Kada jednom uspe da pronađe prostor za izvršavanje sopstvene skripte, napadač može u potpunosti izmeniti tok i opis aplikacije, i uzrokovati štetu

- Cross-site scripting je jedan od glavnih razloga zašto **nikada ne verujemo korisničkom ulazu**

Cross Site Scripting (XSS) - primer

- Recimo da na sajtu postoji forma za unos komentara

```
<form action="process" method="post">
    <textarea name="comment" ></textarea>
    <input type="submit" value="Post" />
</form>
```

- Korisnik može uneti komentar i on će biti procesiran na serveru:

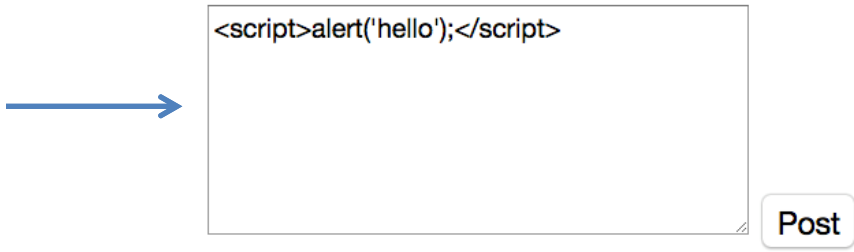
```
FileWriter fw  = new FileWriter("comments.txt", true);
String comment = request.getParameter("comment");
fw.write(comment);
fw.close();
```

- Na nekom mestu na sajtu, mogu se videti komentari:

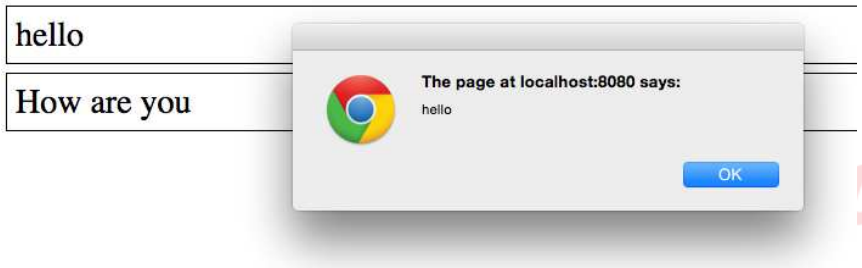
```
FileReader fr = new FileReader("comments.txt");
BufferedReader br = new BufferedReader(fr);
String line;
while((line=br.readLine())!=null){
    out.print("<div style='padding:4px;margin:4px;border:1px solid black;'>"+line+"</div>");
}
```

Cross Site Scripting (XSS) - primer

- Sistem će raditi, sve dok korisnici budu unosili smislene komentare.
- Ali ako korisnik unese na primer sledeći komentar?

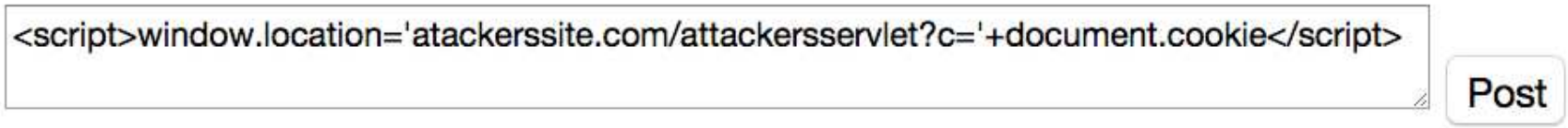


- Na stranici za prikaz komentara, umesto da bude prikazan tekst, biće izvršena skripta:



Cross Site Scripting (XSS) - primer

- Kada korisnik jednom uspe da izvrši skriptu i ustanovi da je sistem nebezbedan sa te tačke, lako može izvršiti i agresivnije skripte.
- Na primer:



Cross Site Scripting (XSS) - rešenja

- **Nikada ne verujemo korisniku ni korisničkom ulazu**
- Moramo prvo videti kako hoćemo da tretiramo ono što korisnik unosi. Od toga zavisi i bezbednosni pristup.
- Na primer:
  - nekada hoćemo da korisnik ne može uopšte da unese skriptu na stranu.
  - nekada hoćemo da može da unese skriptu i da ta skripta bude prikazana (ali ne i izvršena) na izlazu.
  - A nekada hoćemo da omogućimo korisniku unos skripte, i takođe da je izvršimo na strani

Validacija, filtracija i sanacija

- U bilo kom od slučajeva sa prethodnog slajda izvršićemo jednu od dve (ili obe) akcije nad unetim sadržajem. Te akcije su: **VALIDACIJA** i **SANACIJA**
- **Validacija**
  - Provera postojanja opasnih karaktera
- **Sanacija**
  - Izbacivanje opasnih karaktera

Validacija

- Validacija je proveravanje sadržaja.
- Proveravamo da li je ono što smo dobili, ono što smo i očekivali.
- Samom validacijom samo utvrđujemo stanje, ali ona ne podrazumeva reakciju
- Da li (ni)je uneto korisničko ime:

```
if(comment==null){
    //REAKCIJA OVDE
}
```

```
if(comment.trim().isEmpty()){
    //REAKCIJA OVDE
}
```

Sanacija

- Pod sanacijom podrazumevamo akciju koja će “razoružati” određeni string pre nego što ga upotrebimo.
- Ono od čega treba “razoružati” jedan string zavisi od sistema u kome taj string treba da se nađe, ali to najčešće podrazumeva uklanjanje ili modifikaciju karaktera koji mogu da dovedu do XSS-a i Injection-a.
- Podsetimo se da moramo znati šta tačno hoćemo da omogućimo korisniku da uradi pre nego što uradimo nešto na unetom podatku

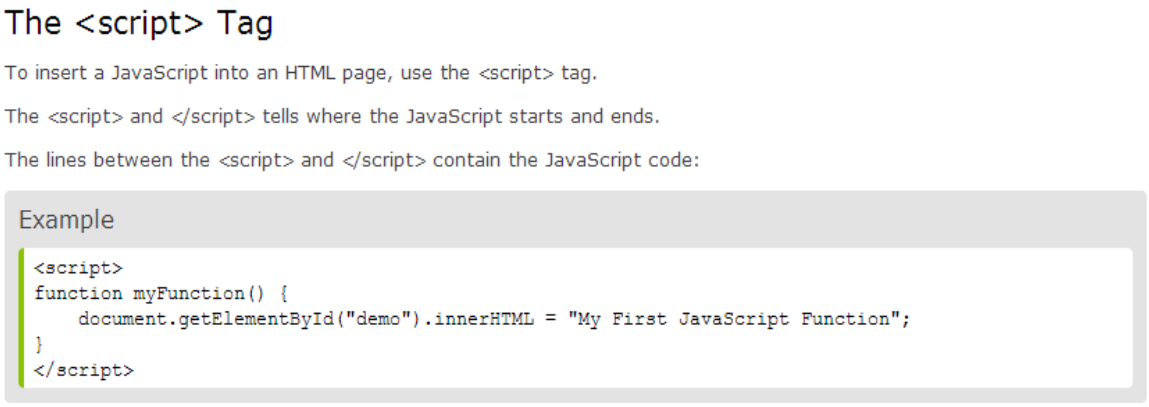
Sanacija – uklanjanje sadržaja

- Nekada jednostavno hoćemo da se oslobodimo potencijalne opasnosti. Na primer, ukoliko korisnik unosi korisničko ime (ili lozinku), nema nikakve potrebe da se u njima nalaze script tagovi. Korisničko ime najčešće ne sadrži oznake: **<** ili **>**
- Ovo znači da ovakve oznake možemo jednostavno eliminisati iz stringa.

```
comment = comment.replace("<", "");
comment = comment.replace(">", "");
```

Sanacija – prikaz sadržaja

- Postoji scenario u kome ne želimo da uklonimo sadržaj, već da ga prikažemo. Na primer, u sledećem delu strane (http://www.w3schools.com/), skriptu treba prikazati a ne obrisati:



The <script> Tag

To insert a JavaScript into an HTML page, use the <script> tag.

The <script> and </script> tells where the JavaScript starts and ends.

The lines between the <script> and </script> contain the JavaScript code:

Example

```
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "My First JavaScript Function";
}
</script>
```

Sanacija – prikaz sadržaja

- Da bi korisniku prikazali tagove, a pri tom izbegli njihovo izvršavanje, koristimo html entitete. Ovo su specijalne oznake kojima predstavljamo browseru karaktere. Ove oznake koristimo kako browser ne bi tretirao karaktere kao html, već kao običan tekst.
- Oznake **<** i **>**, mogu se kroz html entitete predstaviti kao: < i >
- Ako bi se prethodni primer modifikovao na sledeći način:

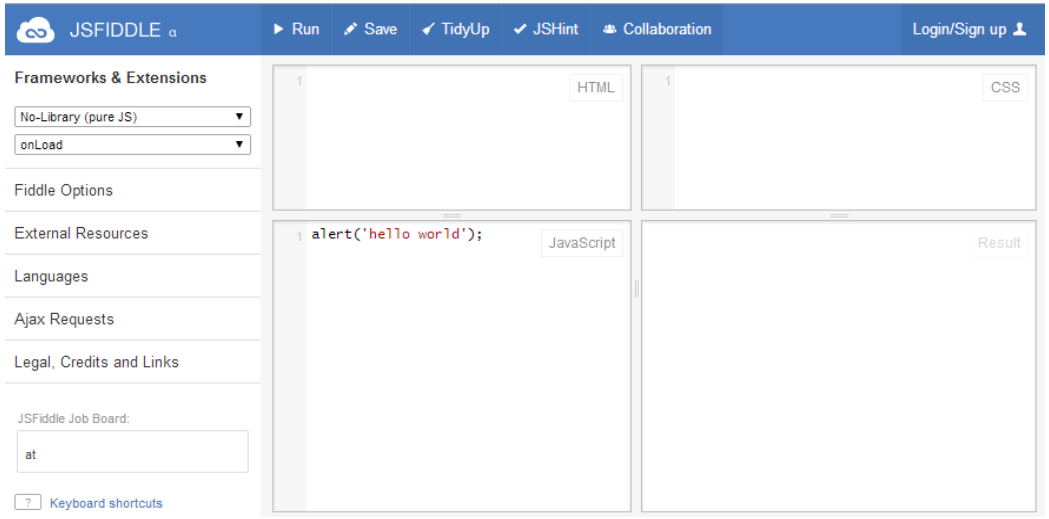
```
comment = comment.replace("<", "&lt;");
comment = comment.replace(">", "&gt;");
```

- Prikazao bi ispravan (i bezbedan) rezultat

<script> function myFunction() { document.getElementById("demo").innerHTML = "My First JavaScript Function"; } </script>

Sanacija – aktivacija sadržaja

- Najopasniji scenario je onaj u kome korisnik treba da ima mogućnost unosa izvršivog sadržaja.



- Tada moramo izvršiti skriptu, ali je i proveriti i ovo se ne može uraditi automatski.

Sanacija – aktivacija sadržaja

- Ako hoćemo da izvršimo skriptu, a pri tom i zadržimo njenu bezbednost, možemo je “razoružati” prilikom ulaska u sistem, a “naoružati” prilikom izlaska iz njega:

```
comment = comment.replace("&lt;", "<");
comment = comment.replace("&gt;", ">");
```

- Ovim nismo postigli puno, ali smo bar onemogućili injection skripte u sistem. I dalje, skripta će biti izvršena kada dođe do klijenta, a pri tom će se tretirati kao skripta sa domena, što je nebezbedno.

Crna i bela lista

- Kada se vrši dobavljanje nekih podataka ili izvršavanje neke akcije na osnovu korisničkog ulaza, mora postojati sistem koji će izolovati sve što ne odgovara tom sistemu (**crna lista**), ili sve što mu odgovara (**bela lista**)
- Bela lista je uvek bolje rešenje, jer se uvek proveravaju podaci iz predefinisanog seta. Na žalost, ona nije uvek moguća (sve je zabranjeno, osim onoga što je dozvoljeno)
- Crna lista je nešto lošije rešenje, jer uvek nudi prostor napadaču za eksperimentisanje (sve što nije zabranjeno, dozvoljeno je)
- U primeru za spoofed forms, korisnik je mogao da unese bilo šta u select kontrolu, i to bi prošlo na serveru. Da je na serveru postojala bela lista koja bi prihvatala samo stringove: “male” i “female” spoofed forma ne bi prošla.

Bezbednost sesija i cookie-a

- Često su meta napada cookie-s i sesije. Oni nisu previše opasni sa stanovišta funkcionalnosti sistema, već više sa stanovišta integriteta i bezbednosti samih podataka. Odnosno, identiteta korisnika.
- Cookie, sam po sebi ne može mnogo nauditi sistemu, ali činjenica da sadrži id sesije korisnika, može da omogući eksploataciju sistema
- Najčešće vrste napada/eksploatacije vezanih za sesije su **session fixation** i **session hijacking**
- **session hijacking**
  - Napadač je preuzeo korisnikovu sesiju i koristi je kako bi se lažno predstavio na sistemu
- **session fixation**
  - Napadač korisniku dodeljuje sopstvenu sesiju kako bi uradio nešto u njegovo ime

Cross Site Request Forgery (CSRF)

Ovaj tip napada eksploatiše korisnikove privilegije bez njegovog znanja, koristeći se njegovom sesijom i http metodama.

```
<img
   src="http://transfernovca.com?izvrsiTransfer
   =1....." />
```

Podešavanje sesija na strani

Provera user agent-a

```
out.print(request.getHeader("user-agent"));
```

Bezbednost baze podataka

- SQL injection

select count(*) from users

where username= ’”+username+”' and pass=’”+password+”'

http://www.jasypt.org/hibernate.html

http://pastie.org/10257837

- Hash šifre u bazi podataka

```
String pass = "mypassword";
MessageDigest md = MessageDigest.getInstance("MD5");
md.update(pass.getBytes());
byte byteData[] = md.digest();
StringBuffer sb = new StringBuffer();
for (int i = 0; i < byteData.length; i++) {
   sb.append(Integer.toString((byteData[i] & 0xff) + 0x100, 16).substring(1));
}
System.out.println(sb.toString());
```

Man in the middle

- Simetricna enkripcija / dekripcija
- Asimetricna enkripcija / dekripcija
- SSL

ITAcademy

http://pastie.org/10255319

Simetrična enkripcija / dekripcija

```
String poruka = "hello world";

KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey secretKey = keyGen.generateKey();
Cipher encCipher = Cipher.getInstance("AES");
encCipher.init(Cipher.ENCRYPT_MODE, secretKey);
byte[] encryptedBytes = encCipher.doFinal(poruka.getBytes());
String encrypted = Base64.getEncoder().encodeToString(encryptedBytes);
String key = Base64.getEncoder().encodeToString(secretKey.getEncoded());

System.out.println("Kriptovano: " + encrypted);
System.out.println("Kljuc: " + key);


SecretKey decSecretKey = new SecretKeySpec(Base64.getDecoder().decode(key),"AES");
Cipher deCipher = Cipher.getInstance("AES");
deCipher.init(Cipher.DECRYPT_MODE, decSecretKey);
byte[] decodedBytes = deCipher.doFinal(encryptedBytes);
String decodedString = new String(decodedBytes);
System.out.println(decodedString);
```

ITAcademy

http://pastie.org/10255882

Asimetrična enkripcija / dekripcija

```
//Generisanje i izdavanje kljuceva
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
KeyPair myPair = kpg.generateKeyPair();
String kljuc = Base64.getEncoder().encodeToString(myPair.getPublic().getEncoded());
String pkljuc = Base64.getEncoder().encodeToString(myPair.getPrivate().getEncoded());
System.out.println("Javni kljuc: " + kljuc);
System.out.println("Privatni kljuc: " + pkljuc);

//Klijent kriptuje sadrzaj javnim kljucem
X509EncodedKeySpec spec = new X509EncodedKeySpec(Base64.getDecoder().decode(kljuc));
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PublicKey key = keyFactory.generatePublic(spec);
Cipher c = Cipher.getInstance("RSA");
c.init(Cipher.ENCRYPT_MODE, key);
String tekst = "How are you";
byte[] kriptovano = c.doFinal(tekst.getBytes());
String rezultat = Base64.getEncoder().encodeToString(kriptovano);
System.out.println("Kriptovani rezultat: " + rezultat);

//Server dekriptuje privatnim kljucem
byte[] keyBytes = Base64.getDecoder().decode(pkljuc);
PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(keyBytes);
KeyFactory fact = KeyFactory.getInstance("RSA");
PrivateKey priv = fact.generatePrivate(keySpec);
Cipher cDec = Cipher.getInstance("RSA");
cDec.init(Cipher.DECRYPT_MODE, priv);
byte[] dekriptovano = cDec.doFinal(Base64.getDecoder().decode(rezultat));
System.out.println(new String(dekriptovano));
```

SSL

- Provera keystore-a
  - ***keytool -list -keystore keystore.jks (sifra changeit)***
- Kreiranje novog ključa
  - ***keytool -genkey -keysize 2048 -genkey -alias nekimojdomen.com -keyalg RSA -keystore keystore.jks***
- Kreiranje CSR-a
  - ***keytool -certreq –alias nekimojdomen.com -keystore keystore.jks -file nekimojdomen.csr***
- Instaliranje sertifikata
  - ***keytool -import -trustcacerts -alias root -file budlefajlodCA.crt -keystore keystore.jks***
  - ***keytool -import -trustcacerts -alias nekimojdomen.com -file sub.nekimojdomen.com.crt*** **-keystore keystore.jks**