1. Injection

Injection napadi se dešavaju kada korisnik ima mogućnost slanja neproverenih podataka s namerom da prevari aplikaciju (sistem) i dobije pristup informacijama za koje nije autorizovan ili izvrši neželjene komande. Injection napadi mogu biti SQL upiti, LDAP upiti ili komande operativnog sistema.

Naše rešenje: Koriste se repozitorijumi spring frameworka za pravljenje upita nad bazom. Repozitorijumi automatski vrše procesiranje ulaznih parametara I samim tim nas štite od injection napada.

UserRepository.java

```java
package ftn.xmlwebservisi.firme.repository;

import org.springframework.data.repository.CrudRepository;

public interface UserRepository extends CrudRepository<User, Integer> {

    User findByUsername(String username);
}
```

Moguća poboljšanja: Uraditi „sanitizaciju“ ulaznih podataka implementacijom „whitelisting“ na serverskoj strani .

2. Broken Authentication

Ovo je vrsta napada u kojoj napadač pokušava da iskoristi neadekvatno implementirane funkcije za kontrolu autentifikacije ili sesije i tako dođe do poverljivih informacija.

Naše rešenje: Korišćenjem brcrypt hashing algoritma sprečili smo credential stuffing (brute force) napade jer je bcrypt adaptivna funkcija koja vremenom postaje sporija I samim tim je imuna na brute force napade.

```java
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    // Using AuthenticationManagerBuilder to create an instance of AuthenticationManager
    // which is used for user authentication
    auth.userDetailsService(userService)
        .passwordEncoder(bCryptPasswordEncoder());
}

@Bean
public BCryptPasswordEncoder bCryptPasswordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Prilikom registracije potrebno je uneti šifru određene jačine.

```
PasswordValidator validator = new PasswordValidator(Arrays.asList(
        // length between 8 and 50 characters
        new LengthRule(8, 50),
        // at least one upper-case character
        new CharacterRule(EnglishCharacterData.UpperCase, 1),
        // at least one digit character
        new CharacterRule(EnglishCharacterData.Digit, 1),
        // at least one symbol (special character)
        new CharacterRule(EnglishCharacterData.Special, 1),
        // no white spaces
        new WhitespaceRule()
        ));

RuleResult result = validator.validate(new PasswordData(value));
if (result.isValid()) {
    return true;
}
```

Postavljeno je kratko vreme trajanja korisničkog tokena kako bi umanjili štetu ukoliko korisnik zaboravi da se izloguje a ulogovao se sa nekog javno dostupnog kompjutera.

```
@Component
public class TokenHandler {

    private Logger logger = LoggerFactory.getLogger(TokenHandler.class);
    private final SignatureAlgorithm SIGNATURE_ALGORITHM = SignatureAlgorithm.HS512;

    private final long EXPIRES_IN = 1800000; // 30 minutes
```

Moguća poboljšanja: Implementacija multifactor autentifikacije . Testiranje aplikacije spram liste top 10000 worst passwords. Limitirati broj pokušaja za logovanje.

3. Sensitive data exposure

Do ove vrste napada dolazi ukoliko web aplikacija ili API ne rukuju pravilno osetljivim podacima. Napadači mogu prisluškivati i modifikovati nezaštićene podatke. Najčešći propusti pri rukovanju osetljivim podacima su korišćenje slabih ključeva za enkripciju, ne korišćenje sigurnih protokola prilikom razmene podataka između klijenta i servera.

Naše rešenje: Sva komunikacija između klijenta I servera se odvija preko HTTPS protokola.

```
# Tell Spring Security (if used) to require requests over HTTPS
security.require-ssl=true
# The format used for the keystore
server.ssl.key-store-type=jks
# The path to the keystore containing the certificate
server.ssl.key-store=classpath:keystore.jks
# The password used to generate the certificate
server.ssl.key-store-password=123456
# The alias mapped to the certificate
server.ssl.key-alias=tomcat
server.ssl.protocol=TLS
server.ssl.enabled=true
```

Korisničke lozinke su pre čuvanja u bazi hash-ovane bcrypt hashing algoritmom.

```
public User createUser(UserDTO userDto) {
    User newUser = new User();
    newUser.setUsername(userDto.getUsername());

    newUser.setPassword(bCryptPasswordEncoder.encode(userDto.getPassword()));

    Role role = roleRepository.findByName("USER");
    newUser.addRole(role);

    return userRepository.save(newUser);
}
```

Podaci u tranzitu (sadržaj SOAP poruke) su enkriptovani po XML signature and encrypton standardu.

```
public Document encrypt(Document document, String elementToEncrypt, SecretKey secretKey, PublicKey publicKey) {
    try {
        org.apache.xml.security.Init.init();
        //Inicijalizacija algoritma za simetricno sifrovanje xml podataka
        XMLCipher xmlContentCipher = XMLCipher.getInstance(XMLCipher.TRIPLEDES);
        xmlContentCipher.init(XMLCipher.ENCRYPT_MODE, secretKey);

        //Inicijalizacija algoritma za asimetricno sifrovanje tajnog kljuca. Sifruje se javnim kljucem primaoca
        XMLCipher keyCipher = XMLCipher.getInstance(XMLCipher.RSA_v1dot5);
        keyCipher.init(XMLCipher.WRAP_MODE, publicKey);
        EncryptedKey encryptedKey = keyCipher.encryptKey(document, secretKey);

        //U sadrzaj EncryptedData elementa se dodaje sifrovan tajni kljuc
        EncryptedData encryptedData = xmlContentCipher.getEncryptedData();
        KeyInfo keyInfo = new KeyInfo(document);
        keyInfo.addKeyName("Encrypted SecretKey");
        keyInfo.add(encryptedKey);
        encryptedData.setKeyInfo(keyInfo);

        //Ako je prosledjeno ime elementa sifruje sve elemente sa datim imenom, u suprotnom sifruje celokupan dokument
        if(elementToEncrypt == null) {
            xmlContentCipher.doFinal(document, document);
        } else {
            NodeList elementsToEncrypt = document.getElementsByTagNameNS("http://www.ftn.xml/banke", elementToEncrypt);
            for(int i=0; i < elementsToEncrypt.getLength(); i++) {
                xmlContentCipher.doFinal(document, ((Element) elementsToEncrypt.item(i)), true);
            }
        }
        return document;
```

Moguća poboljšanja: Izbeći čuvanje osetljivih podataka gde god je to moguće. Osetljive podatke koji se čuvaju treba enkriptovati korišćenjem sigurnih algoritama.

4. XML External Entities (XXE)

Aplikacija (XML bazirani web servis) je podložna XXE napadima ukoliko omogućava preuzimanje XML dokumenata koji sadrže reference ka „spoljašnjim entitetima“ koji se parsiraju pomoću loše konfigurisanih XML parsera. Ovo se može iskoristiti kako bi se ukrali osetljivi podaci ili izvršili neki maliciozni zadaci.

Naše rešenje: Onemogućeno je parsiranje dokumenata putem DTD šeme. Koristi se XML parser (DocumentBuilderFactory) koji je konfigurisan tako da štiti od XXE napada.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
// Onemogucava parsiranje eksternih entiteta
dbf.setExpandEntityReferences(false);

// Onemogucava DTD
String FEATURE = "http://apache.org/xml/features/disallow-doctype-decl";
dbf.setFeature(FEATURE, true);
```

XML dokumenti se validiraju korsiteći XSD šemu.

```
@Override
public void addInterceptors(List<EndpointInterceptor> interceptors) {
    PayloadValidatingInterceptor validatingInterceptor = new PayloadValidatingInterceptor();
    validatingInterceptor.setValidateRequest(true);
    validatingInterceptor.setXsdSchema(soapSchema());
    interceptors.add(validatingInterceptor);
}

@Bean
public XsdSchema soapSchema() {
    return new SimpleXsdSchema(new ClassPathResource("banka_soap.xsd"));
}
```

Moguća poboljšanja: Koristiti manje kompleksne formate podataka (JSON). Implementirati „whitelisting“ za validaciju ulaznih podataka. Koristiti web application firewall za detekciju i blokiranje XXE napada.

5. Broken Access Control

Broken access control dovodi do toga da korisnik ima pristup resursima za koje nije autorizovan. To dovodi do neovlašćenog otkrivanja informacija, modifikacije ili uništenja svih podataka.

Naše rešenje: Implementirana autorizacija po RBAC modelu gde svaki korisnik ima predefinisan set rola (uloga) na osnovu kojih se određuje da li može da pristupi resursu.

```
// Entry points
http.authorizeRequests()
    .antMatchers(HttpMethod.POST, "/register").permitAll()
    .antMatchers("/").permitAll()
    .anyRequest().authenticated()
```

```
@PreAuthorize("hasAnyRole('INSIDER')")
@GetMapping
public ResponseEntity<List<NalogZaPlacanje>> nadjiSveNaloge() {
    List<NalogZaPlacanje> response = nalogZaPlacanjeServis.nadjiSveNalogeZaPlacanje();
    if(response != null) {
        return new ResponseEntity<List<NalogZaPlacanje>>(response, HttpStatus.OK);
    } else {
        return new ResponseEntity<List<NalogZaPlacanje>>(response, HttpStatus.BAD_REQUEST);
    }
}
```

6. Security Misconfiguration

Konfiguracija se vrši na svim nivoima sistema, nivou aplikacije, servera, baze podataka i drugim uređajima koji su potrebni za rad sistema stoga je veoma važno da te konfiguracije budu u skladu sa sigurnosnim zahtevima.

Naše rešenje: Uzimajući u obzir da se radi o studentskom projektu koji se koristi u demonstrativne svrhe sva podešavanja servera, baze podataka i servera baze podataka su podrazumevana.

```
spring.datasource.url = jdbc:mysql://localhost:3306/xmldb
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
server.port = 8081
```

Moguća poboljšanja: Ukoliko bi došlo do deployment-a aplikacije bilo bi potrebno upoznati se sa najboljim praksama za menadžment konfiguracija
https://www.owasp.org/index.php/Testing_for_configuration_management

7. Cross site scripting (XSS)

XSS je vrsta injection napada do koje dolazi ukoliko napadač ima mogućnost da ubaci nedozvoljene podatke tj skripte u web stranicu korisnika. Ubačeni podaci odnosno skripte bivaju izvršene od strane browser-a što može dovesti do krađe korisničkih informacija, krađe sesije...

Naše rešenje: AngularJS framework automatski „eskejpuje“ html karaktere

8. Insecure Deserialization

Insecure deserialization predstavlja vrstu napada gde napadač koristi lošu konfiguraciju sistema tako da pri deserializaciji objekta sistem prihvati maliciozni sadržaj bez njegove provere (ili uz neadekvatnu proveru) i time dopusti napadaču pristup osetljivim funkcijama ili podacima.

Naše rešenje: Prilikom prijema SOAP poruke vrši se provera digitalnog potpisa dokumenta i time se potvrđuje da je sadržaj dospeo od kredibilnog korisnika. Nakon potvrde identiteta pošiljaoca poruke xml sadržaj se parsira u Java objekat u skladu sa postojećom xml šemom.

```
Document decryptedDocument = xmlSignAndEncryptUtility.veryfyAndDecrypt(inStream);
JAXBElement<PosaljiNalogZaPlacanjeRequest> posaljiNalogZaPlacanjeRequest = null;
try {
    JAXBContext jaxbContext;
    if((decryptedDocument != null) &&
            (decryptedDocument.getDocumentElement().getLocalName().equals(PosaljiNalogZaPlacanjeRequest.class.getSimpleName()))) {
        //Unmarshal dokumenta u objekat
        jaxbContext = JAXBContext.newInstance(PosaljiNalogZaPlacanjeRequest.class);
        Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
        posaljiNalogZaPlacanjeRequest = unmarshaller.unmarshal(decryptedDocument, PosaljiNalogZaPlacanjeRequest.class );
```

9. Using components with known vulnerabilities

10. Insuficient logging and monitoring

Kako bi registrovali I sprečili napadača pre nego što kompromituje naš sistem potrebno je logovati I pratiti sve aktivnosti kako bi mogli lakše da pratimo sumnjivo ponašanje.

Naše rešenje: Implementirano je logovanje neuspešne autentifikacije kao i logovanje bitnih izuzetaka

```
    TokenService tokenService = new TokenService();
    Claims claims = tokenService.getClaims(token, publicKey);
    if(claims != null) {
        Date expiration = claims.getExpiration();
        if (new Date(System.currentTimeMillis()).before(expiration)) {
            String id = claims.getId();
            if(!jwtIDs.contains(id)) {
                jwtIDs.add(id);
                return true;
            } else {
                logger.info("JWT token invalid, token ID already used, Obj={}", id);
            }
        } else {
        Date now = new Date(System.currentTimeMillis());
        logger.info("JWT token expired @time={}, expiration time=, Obj={}", now, expiration, token);
        }
    }
} catch (CertificateException e) {
    logger.error("Invalid certificate: Obj={}", e.getCause(), e);
    e.printStackTrace();
} catch (FileNotFoundException e) {
    logger.error("Certificate file could not be found: Obj:{}", e.getCause(), e);
    e.printStackTrace();
}
```

Moguća poboljšanja: Uzimajući u obzir da se radi o studentskom projektu koji se koristi u demonstrativne svrhe implementirani logovi ne pružaju mogućnost detaljne analize uzroka problema. U realnom sistemu bio bi korišten centralizovani “log management” sistem uz servis koji prati log fajlove I ukazuje na potencialne napade.