PCI DSS Standard

I grupa

1. Svi delovi sistema moraju biti zaštićeni od neautorizovanog pristupa

   Da bi se pristupilo sistemu i njegovim funckionalnostima, korisnik mora biti autentifikovan. Za autentifikaciju i autorizaciju postoji poseban mikroservis - **Auth Service** koji izdaje JWT token koji se nalazi u Cookie-u pomoću kojeg je kasnije omogućena autetifikacija korisnika.

   Primer login funkcije koja vraca Cookie prikačen na Response objekat:

```
public LoginResponse login(LoginRequest loginRequest, HttpServletResponse response) throws Exception {
    Authentication authentication = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(
            loginRequest.getEmail(),
            loginRequest.getPassword()
    ));
    User user = (User) authentication.getPrincipal();

    SecurityContextHolder.getContext().setAuthentication(authentication);
    String accessToken = tokenProvider.createAccessToken(authentication);
    Integer tokenExpirationSeconds = appProperties.getAuth().getTokenExpirationSeconds();
    CookieUtils.addCookie(
            response,
            TokenAuthenticationFilter.ACCESS_TOKEN_COOKIE_NAME,
            accessToken,
            tokenExpirationSeconds
    );


    Long expiresAt = tokenProvider.readClaims(accessToken).getExpiration().getTime();

    return new LoginResponse(user.getId().toString(), expiresAt, user.getRole());
}
```

   Kada zahtev stigne na PSP, potrebno je autentifikovati korisnika kako bi se zahtev prosledio odgovarajućem mikroservisu za plaćanje.

   Verifikacije korisnika na PSP mikroservisu:

```
try {
    authenticate(req, httpRequest);
} catch (Exception e) {
    logger.error( message: "Authentication failed. Message: " + e.getMessage(), new Date().toString(), service: "PaymentServiceProvider", req);
    return ResponseEntity.status(500).body(Map.of( k1: "Error", v1: "Failed to make the request"));
}
```

```java
private void authenticate(HttpServletRequest httpRequest) throws IOException {
    Cookie[] cookies = httpRequest.getCookies();
    Cookie accessCookie = Arrays.stream(cookies) Stream<Cookie>
            .filter(cookie -> cookie.getName().equals("access_token"))
            .findFirst() Optional<Cookie>
            .orElseThrow(() -> new RuntimeException("No access token found"));
    String token = accessCookie.getValue();
    OkHttpClient okHttpClient = new OkHttpClient.Builder()
            .addInterceptor(chain -> {
                Request originalRequest = chain.request();
                Request.Builder builder = originalRequest.newBuilder()
                        .header( name: "Authorization",  value: "Bearer " + token);
                Request newRequest = builder.build();
                return chain.proceed(newRequest);
            })
            .build();

    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://localhost:8001/auth/verify/")
            .addConverterFactory(GsonConverterFactory.create())
            .client(okHttpClient)
            .build();
    AuthCall auth = retrofit.create(AuthCall.class);
    Call<Map<String, Object>> call = auth.authenticate( url: "http://localhost:8001/auth/verify/");
    Response<Map<String, Object>> response = call.execute();
    if (!response.isSuccessful()) {
       throw new RuntimeException("Authentication failed");
    }
}
```

2. Definisanje rola i odgovornosti

U sistemu postoje dve uloge: User i Admin. Uloga se dodaje svakom korisniku.
Primer klase Role:

```java
@Table(name = "ROLE")
public class Role implements GrantedAuthority {
    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;

    @Column(name = "name")
    String name;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "role_privilege",
            joinColumns = @JoinColumn(name = "role_id", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name = "privilege_id", referencedColumnName = "id"))
    private Set<Privilege> privileges;

    @JsonIgnore
    @ManyToMany(mappedBy = "roles" , fetch = FetchType.LAZY)
    private Set<User> users;
```

Svaka ulogu ima privilegiju, dodela privilegija rađena je po principu **least privilege**.

Admini mogu da kreiraju nove korisnike, ažuriraju njihove podatke i vrše plaćanje,, korisnici mogu da vrše plaćanje.

Primer zaštite ruta izgleda ovako:

```java
    Anđela
    @GetMapping()
    @PreAuthorize("hasRole('ADMIN')")
    public List<UserInfoResponse> getLawyers() { return userService.getLawyers(); }

    Anđela
    @GetMapping("/{userEmail}")
    @PreAuthorize("hasRole('ADMIN')")
    public UserDetailResponse getLawyer(@PathVariable String userEmail) { return userService.getLawyer(userEmail

    Anđela
    @PutMapping()
    @PreAuthorize("hasRole('ADMIN')")
    public UserDetailResponse editLawyer(@RequestBody UserDetailsRequest user) { return userService.editLawyer(u

    Anđela
    @DeleteMapping("/{userEmail}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseOk deleteUser(@PathVariable String userEmail) {
        userService.deleteUser(userEmail);
        return new ResponseOk("User logically deleted.");
    }
```

3. Ne koristiti protokole za koje je ustanovljeno da nisu bezbedni – poput SSL v1, FTP

   Ne koriste se prokoli poput SSL v1 i FTP, za Web komunikaciju koristi se HTTPS (SSL TLS) prokol i prilikom komunikacije komponenata unutar sistema, koristi se HTTP prokol.

4. Ne koristiti podrazumevane naloge ili lozinke

U sistemu se ne koriste podrazumevani nalozi niti lozinke. Lozinka korisnika mora sadržati velika i mala slova, specijalne karaktere i biti minimum 12 karaktera dugačka.

```
public Boolean checkPasswordStrength(String password) {
    if (password.length() < 12) {
        return false;
    }
    boolean hasUppercase = !password.equals(password.toLowerCase());
    boolean hasLowercase = !password.equals(password.toUpperCase());
    boolean hasNumber = password.matches( regex: ".*\\d.*");
    boolean hasSpecialChar = password.matches( regex: ".*[!@#$%^&*()_+].*");
    return hasUppercase && hasLowercase && hasNumber && hasSpecialChar;
}
```

II grupa

1. Zastititi sacuvane podatke naloga

Sve lozinke korisnika se heširaju. Heširanje lozinke postiže se korišćenjem *PasswordEncoder* interfejsa iz Spring Security biblioteke, specifično kroz **BCryptPasswordEncoder** koji se specificira u konfiguraciji.

Heširanje lozinke:

```
Lawyer.setPassword(passwordEncoder.encode(registrationRequest.getPassword()));
```

Pored lozinke, potrebno je i enkriptovati korisnikove podatke o kartici. Ono što se enkriptuje jeste PAN, SecurityCode i CardholderName.
Enkripcija se radi prilikom čuvanja podataka u bazu na sledeći način:

```java
public static String encrypt(String value) {
    try {
        IvParameterSpec iv = new IvParameterSpec(initVector.getBytes(StandardCharsets.UTF_8));
        SecretKeySpec skeySpec = new SecretKeySpec(key.getBytes(StandardCharsets.UTF_8), algorithm: "AES");

        Cipher cipher = Cipher.getInstance(algo);
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv);

        byte[] encrypted = cipher.doFinal(value.getBytes());
        return Base64.encodeBase64String(encrypted);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return null;
}
```

Ovde se radi o simetričnoj enkripciji koristeći AES algoritam u CBC (Cipher Block Chaining) modu sa PKCS5 paddingom.

2. Zaštitite podatke vlasnika kartice snažnom kriptografijom prilikom prenosa preko otvorene javne mreže

III grupa

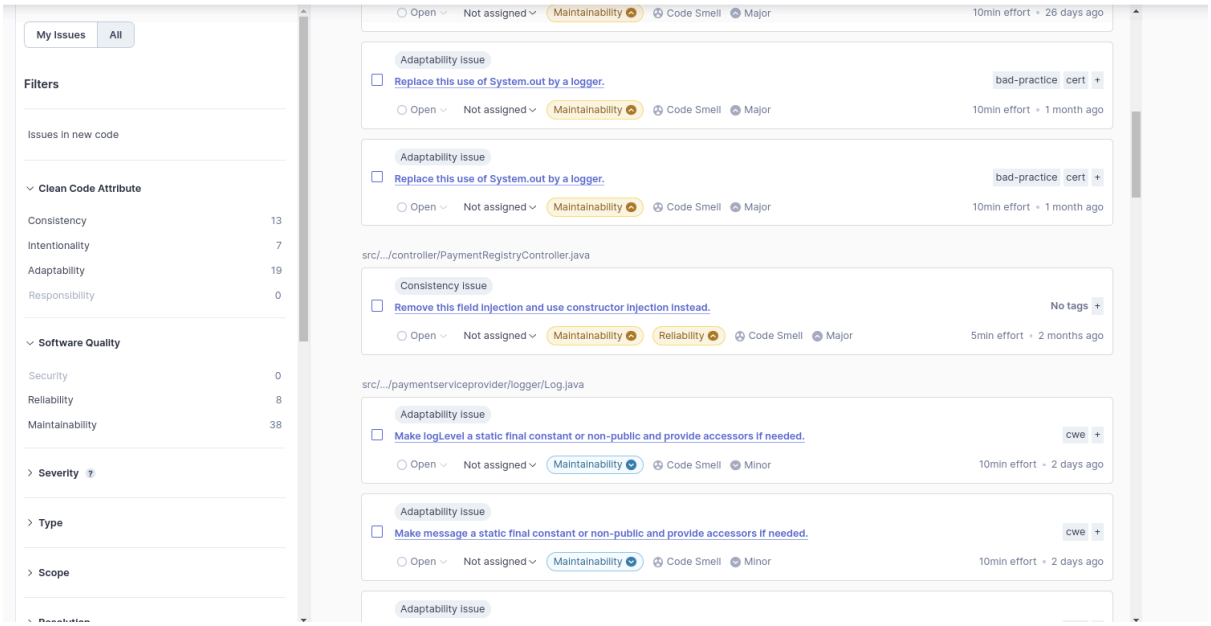
1. Periodično skeniranje ranjivosti

Skeniranje ranjivosti vrši se uz pomoć **SonarQube** alata. Potrebno je pozicionirati se u svaki od projekta i izvršiti komandu:

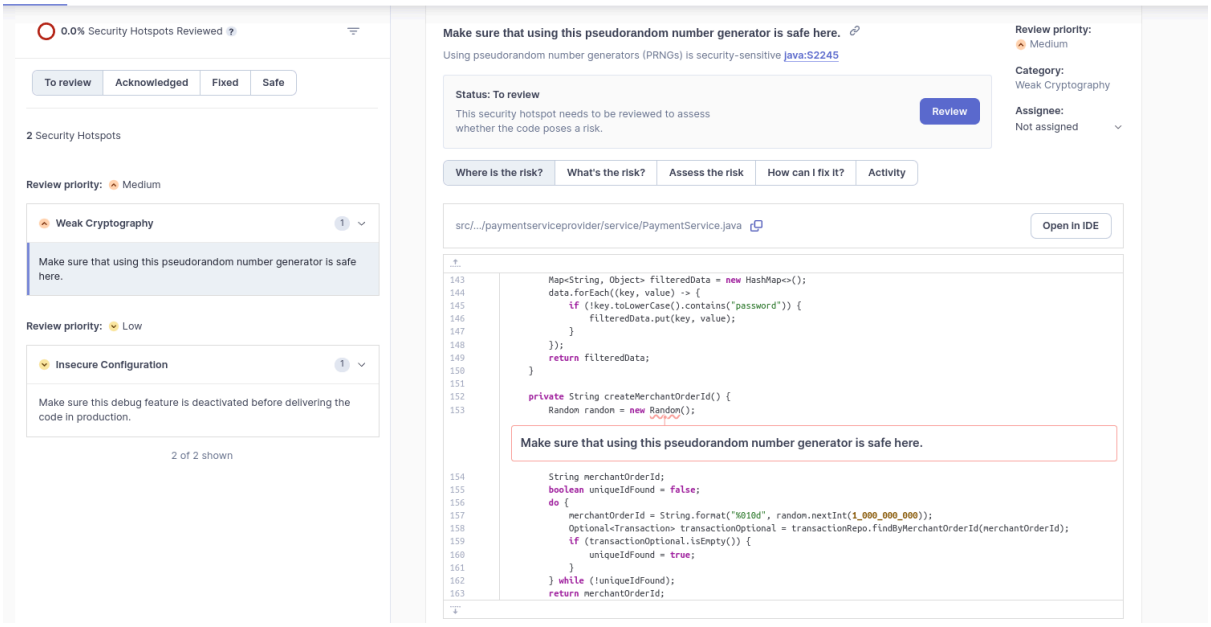
```
Unset
mvn clean verify sonar:sonar \
-Dsonar.projectKey=sep \
 -Dsonar.projectName='sep' \
 -Dsonar.host.url=http://localhost:9000 \
 -Dsonar.token=sqp_77f770af99b1ad42b0564c047ae059bd6ddcf62f
```

U SonarQube-u mogu se videti potencijalni bagovi, kvalitet koda (pokrivenost testovima, dupliciran kod), sigurnosni propusti i još mnogo toga.

Na slici je prikazana analiza za **payment-service-provider** mikroservis:



Provera bezbednosti:



Pregled čitavog sistema:

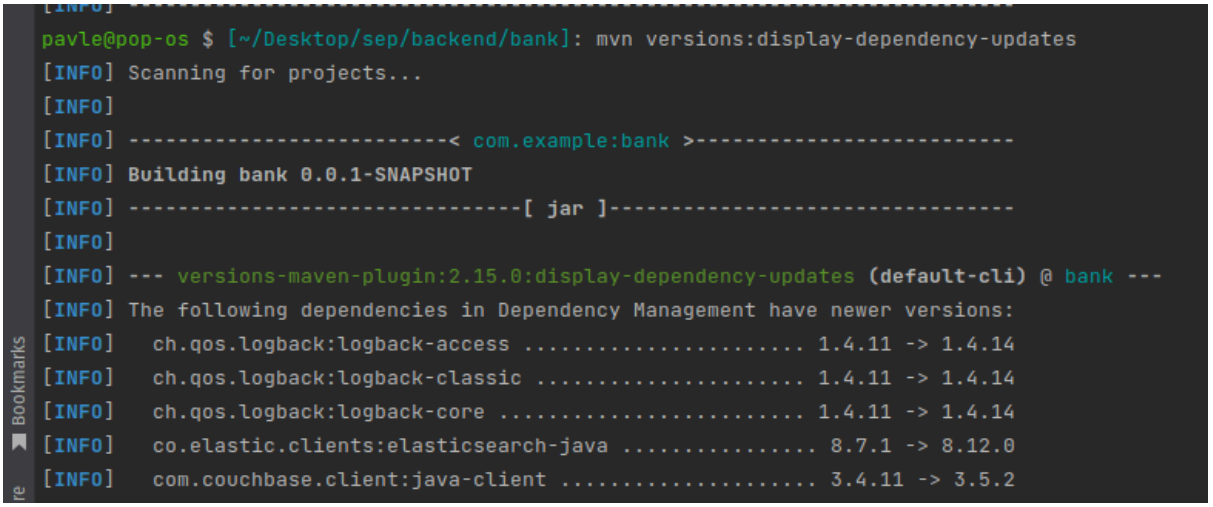
2. Upravljanje zakrpama softvera

Uz pomoć **Maven dependency management**. Potrebno je pozicionirati se u odgovarajući mikroservis i pokrenuti dve komande:

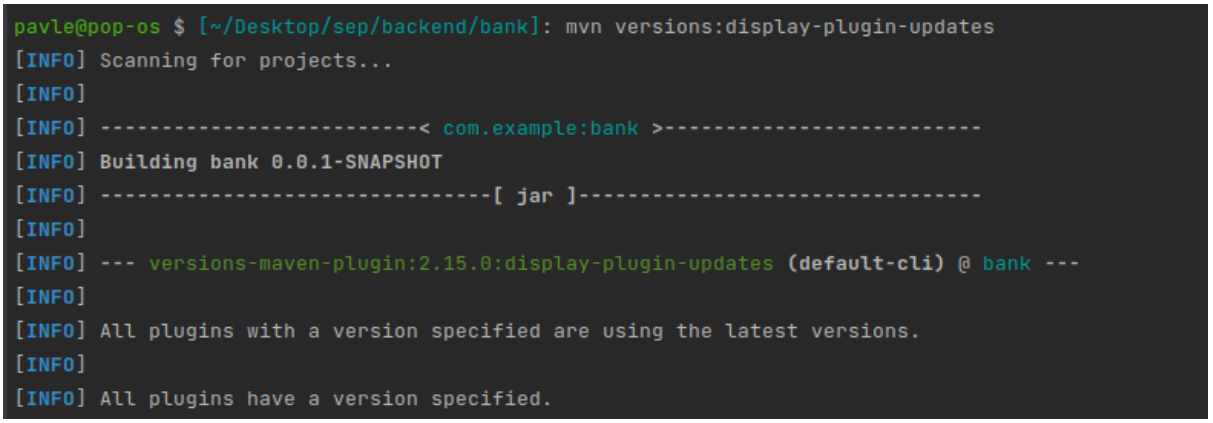
```
Unset
mvn versions:display-dependency-updates
mvn versions:display-plugin-updates
```

Komande se koriste za proveru verzija zavisnosti i pluginova koje se nalaze u pom.xml paketu.

Primer pokrenute komanda za upravljanje zavisnostima u **bank** mikroservisu:



Primer pokrenute komanda za upravljanje pluginova u **bank** mikroservisu:



IV grupa

1. Ograničiti pristup podacima vlasnika kartice

Pristup podacima se ograničava uz pomoć definisanih uloga sa **least privilege principom**. Takođe, prilikom svakog plaćanja vrši se autentifikacija korisnika uz pomoć JWT tokena.

2. Identifikacija i potvrda pristupa komponentama sistema

Svi entiteti u sistemu su jednoznačno određeni svojim identifikatorom. S obzirom na to da se koristi MongoDB baza podataka, svakom korisniku pri čuvanju u bazu, automatski se kreira objekat koji je tipa **ObjectId**. Autentifikacija korisnika se vrši na osnovu identifikatora.

V grupa

1. Pratiti i nadgledati sav pristup mrežnim resursima i podacima o vlasnicima kartica

Svaka akcija u sistemu se loguje. Logovi se čuvaju u MongoDB.
Za potrebe logovanja napravljena je klasa Logger koja poseduje tri funckije kako bi se označile tri vrste logova:

```
public class Logger {

    @Autowired
    private LoggerRepo LoggerRepo;

    13 usages  Aleksandra Nedic
    public void info(String message, String timestamp, String service, Object payload) {
        Log log = new Log(LogLevel.INFO, message, timestamp, service, payload);
        LoggerRepo.save(log);
    }

    no usages  Aleksandra Nedic
    public void warn(String message, String timestamp, String service, Object payload) {
        Log log = new Log(LogLevel.WARN, message, timestamp, service, payload);
        LoggerRepo.save(log);
    }

    19 usages  Aleksandra Nedic
    public void error(String message, String timestamp, String service, Object payload) {
        Log log = new Log(LogLevel.ERROR, message, timestamp, service, payload);
        LoggerRepo.save(log);
    }
}
```

Kao što se može videti iz priloženog, logovi sadrže vreme kreiranja loga, servis u kom se log kreirao i dodatni payload kako bi se bliže opisala data akcija.

Primer u kom se čuva log jeste kada se kreira transakcija, izvrši plaćanje, ili kada se dogodi neka greška, kao što je na primer nedostatak sredstava na računu ili nemogućnost pronalaska korisnika.

Primer nekih od logova koji se nalaze u bazi:

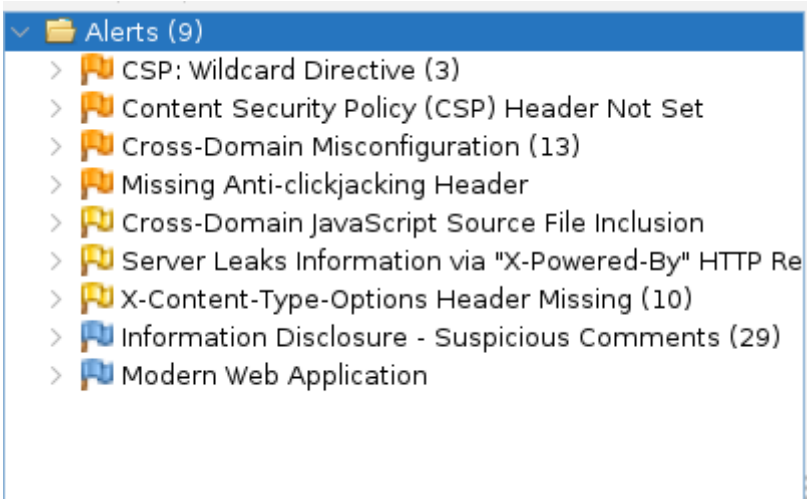

```
_id: ObjectId('65b5381fc28003210de9f259')
logLevel: "ERROR"
message: "Payment failed. Message: Not Found"
timestamp: "Sat Jan 27 18:06:39 CET 2024"
service: "PaymentServiceProvider"
▸ payload: Object
_class: "com.example.paymentserviceprovider.logger.Log"
```

```
_id: ObjectId('65b6db281a5e2d5757908051')
logLevel: "ERROR"
message: "Authentication failed. Message: Use JsonReader.setLenient(true) to acc…"
timestamp: "Sun Jan 28 23:54:32 CET 2024"
service: "PaymentServiceProvider"
▸ payload: Object
_class: "com.example.paymentserviceprovider.logger.Log"
```

```
_id: ObjectId('65b6dbaec42c6268b71525c9')
logLevel: "INFO"
message: "Payment successful"
timestamp: "Sun Jan 28 23:56:46 CET 2024"
service: "PaymentServiceProvider"
▸ payload: Object
_class: "com.example.paymentserviceprovider.logger.Log"
```


2. Redovno testiranje bezbednosnih sistema i procesa

Skeniranje bezbednosti frontend aplikacije izvršeno je OWASP ZAP alatom. Pri skeniranju, dobijeni su sledeći rezultati:



Možemo primetiti da alat nije uspeo da pronađe rizik visokog prioriteta. Kao što je prikazano na slici, za prve dve stavke potrebno je postaviti odgovarajuća CSP zaglavlja. Cross-Domain configuration dešava se zato što je Access-Allow-Origin zaglavlje postavljeno na vrednost *, što svakako ne bi trebalo biti slučaj u produkcionom okruženju. Za Missing Anti-clickjacking

Header potrebno je dodati zaglavlje koje onemogućava da se sajt integriše u okviru iframe elementa. Ostala upozorenja su niskog prioriteta.

Za testiranje sigurnosti korišćen je još jedan alat, a to je DirBuster koji pronalazi rute koje mogu sadržati osetljive informacije. Pronađena je samo ruta /error.