Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.4:621.395
DOI: https://doi.org/10.24867/29OI02Simeunovic

DEMO OKRUŽENJE ZA ANALIZU NAJČEŠĆIH SLABOSTI APLIKATIVNIH PROGRAMABILNIH INTERFEJSA (API)

DEMO ENVIRONMENT FOR ANALYZING THE MOST COMMON WEAKNESSES OF APPLICATION PROGRAMMING INTERFACES (API)

Milica Simeunović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – INFORMACIONA BEZBEDNOST

Kratak sadržaj – *Neprofitna organizacija OWASP redovno objavljuje listu od deset najčešćih slabosti aplikativnih programabilnih interfejsa (API). Cilj ovog rada je razvoj demo okruženja za praktično prikazivanje tih slabosti, odnosno prikaz optimalnih mera bezbednosti za otklanjanje istih. Demo okruženje se sastoji od slabe i zaštićene verzije API-ja koji je razvijen korišćenjem okvira NodeJS. U ovom radu su prikazane prvih pet slabosti i opisane optimalne mere bezbednosti za njihovo rešavanje.*

Ključne reči: *Informaciona bezbednost, aplikativnih programabilni interfejs (API), OWASP, bezbednosni rizici*

Abstract – *The non-profit organization OWASP regularly publishes a list of the ten most common application programming interface (API) weaknesses. The goal of this work is the development of a demo environment for the practical presentation of those weaknesses, that is, the presentation of optimal security measures to eliminate them. The demo environment consists of a weak and protected version of the API developed using the NodeJS framework. This paper presents the first five weaknesses and describes the optimal security measures for solving them.*

Keywords: *Information security, application programming interface (API), OWASP, security risk*

1. UVOD

U savremenom digitalnom ekosistemu, aplikativni programabilni interfejsi (eng. Application Programming Interface - API) postali su nezaobilazna karika koja omogućava interakciju između različitih softverskih sistema. Organizacije širom sveta se sve više oslanjaju na API-je kako bi postigle bržu razmenu podataka. Da bi se identifikovali i razumeli najbitniji bezbednosni rizici u kontekstu API-ja, Open Worldwide Application Security Project (OWASP) redovno sprovodi istraživanje i objavljuje OWASP Top 10 API bezbednosne rizike.

Cilj rada je dublje istraživanje i analiza identifikovanih bezbednosnih rizika. Rad pruža uvid u najnovije trendove i najbolje prakse u API bezbednosti, nudeći konkretne smernice i preporuke za obezbeđivanje API bezbednosti u različitim scenarijima.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Imre Lendak, vanr. prof.

1.1. Common vulnerability scoring system - CVSS

CVSS predstavlja okvir koji se koristi za procenu ozbiljnosti sigurnosnih ranjivosti u softveru. Tabela 1 predstavlja veličinu i ozbiljnost njihovog potencijalnog uticaja. Cilj CVSS-a je pomoći organizacijama da prioritetizuju svoje odgovore na sigurnosne pretnje.

Tabela 1. *CVSS bodovi*

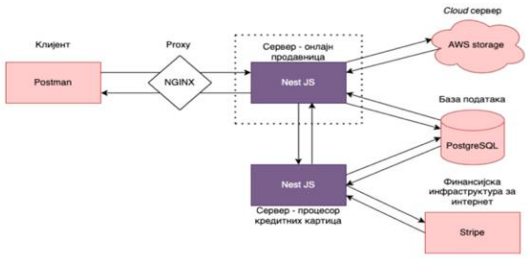
Nivo ozbiljnosti	CVSS bodovi
Nijedan	0.0
Nizak	0.1 – 3.9
Srednji	4.0 – 6.9
Visok	7.0 – 8.9
Kritičan	9.0 – 10.0

2. KORIŠTENE TEHNOLOGIJE I ALATI

NestJS je napredni Node.js okvir za izgradnju efikasnih, pouzadnih i skalabilnih aplikacija na strani servera [2]. Koristi progresivan JavaScript i u potpunosti podržava TypeScript. PostgreSQL je objektno-relaciona baza podataka koja koristi i nasleđuje SQL jezik u kombinaciji sa brojnim funkcijama [3]. Postman je alat za testiranje i razvoj API-ja koji olakšava testiranje API-ja tako što omogućava korisnicima da šalju HTTP zahteve određenom API-ju i analiziraju odgovore [4].

3. OPIS SISTEMA

Sistem se sastoji od NestJS servera – onlajn prodavnica i procesor kreditnih kartica, PostgreSQL baze podataka i AWS servera za skladištenje fotografija. Sistem pruža niz funkcionalnosti koje omogućavaju registraciju korisnika, upravljanje proizvodima, kao i efikasno obavljanje transakcija. Proces transakcije omogućen je integracijom sa Stripe finansijskom infrastrukturom. Na slici 1 prikazan je dijagram sistema. U skladu sa OWASP Top 10 API 2023 smernicama, sistem je projektovan sa namerno umetnutim slabostima kako bi pokazao loše i dobre prakse u razvoju sistema baziranih na upotrebi API-ja.



Slika 1. *Sistemska arhitektura sistema*

4. OWASP TOP 10 API BEZBEDONOSTI RIZICI

4.1. A01:2023 - Broken Object Level Authorization (BOLA)

BOLA se odnosi na nedostatke u kontroli pristupa koja se primenjuje na resurse, odnosno objekte unutar sistema.

BOLA je čest u aplikacijama u kojima se pristup objektima zasniva na identifikatorima, što može dovesti do otkrivanja podataka neovlašćenim stranama ili čak preuzimanja naloga [1].

Rešenje ovog problema zahteva uključivanje preciznije kontrole pristupa na nivou objekta i sprečavanje neovlašćenog pristupa, kako bi se zaštitio integritet sistema i osetljivih informacija.

Napadači mogu da iskoriste API sa slabom autorizacijom manipulisanjem identifikatorom objekta koji se šalje u okviru zahteva. Identifikatori objekata mogu se sastojati, od niza celih brojeva, UUID-ova ili generičkih stringova.

4.1.1. Eksploatacija slabosti

API zahtev GET {{host}}/api/v1/card/1 vraća informacije o platnoj kartici sa navedenim identifikatorom. Server kod koji izvršava ovu akciju je prikazan na listingu 1.

```
 async getOneBad(id: number): Promise<CardResult> {
   const card = await Card.findOne({ where: { id_number: id,
is_deleted: false } });
   if (!card)
    throw new NotFoundException(`Card with provided id does not
exist!`);

   let cardResult = new CardResult();
   cardResult.name = card.name;
   cardResult.number = card.number;
   card.cvc = card.cvc;
   return cardResult;}
```

Listing 1. *Kod za dobavljanje informacija o kartici*

Napadač može da iskoristi ranjivost funkcije promenom vrednosti prosleđenog parametra id. Naime, ukoliko napadač promeni vrednost identifikatora u zahtevu, može pristupiti informacijama o kartici drugih korisnika.

Dodatni propust u implementaciji je korišćenje broja kao identifikatora, što je problematično zbog svoje predvidljivosti, umesto unikatnog identifikatora, kao što je UUID.

4.1.2. Mitigacija

Svaka API krajnja tačka koja prima identifikator objekta i izvršava bilo koju radnju na objektu treba da implementira provere autorizacije na nivou objekta. Provere treba da potvrde da li korisnik koji šalje zahtev ima dozvole da izvrši traženu radnju na traženom objektu.

Listing 2 sadrži bezbedan kod koji prikazuje način dobavljanja informacija o kartici putem mehanizma kontrole pristupa.

```
async getOneGood(id: number, userId: string): Promise<CardResult>
{
   const card = await this.entityManager.createQueryBuilder(Card, 'c')
    .where({ id_number: id, is_deleted: false })
    .appendCardFilter('id_number', userId) // get only one that the
logged in user can see (access is allowed)
    .getOne();}
```

Listing 2. *Bezbedan način pristupa objektu*

Primećujemo upotrebu specifičnog upita koji se šalje bazi podataka, koristeći funkciju .appendCardFilter(), listing 3. Ovaj kod poziva funkciju u bazi podataka čija implementacija je navedena na listingu 4. Na ovaj način, osigurava se da će biti prikazane samo kartice kojima trenutno ulogovani korisnik ima pristup.

```
SelectQueryBuilder.prototype.appendCardFilter = function
<Entity>(this: SelectQueryBuilder<Entity>, column: string, userId:
string): SelectQueryBuilder<Entity> {
  const subQueryDbFunction: string = "get_user_allowed_card_ids";
  const sqlCondition = `${this.alias}.${column} IN (SELECT id FROM
${subQueryDbFunction}('${userId}'))`;
  this.andWhere(sqlCondition);
  return this;}
```

Listing 3. *appendCardFilter()*

```
CREATE OR REPLACE FUNCTION
"get_user_allowed_card_ids"(userId uuid)
   RETURNS TABLE("id" int4) AS $BODY$
   SELECT id_number
   FROM card
   WHERE is_deleted = false
   AND created_by_id = userId;
```

Listing 4. *Funkcija baze podataka*

4.2. A02:2023 - Broken Authentication (BA)

BA je na drugom mestu na OWASP listi Top 10. Obuhvata niz problema vezanih za nepravilnu autentifikaciju i implementaciju sesije, otvarajući put napadačima da dobiju neovlašćen pristup korisničkim nalozima i poverljivim informacijama [1].

Do grešaka u identifikaciji i autentifikaciji može doći kada funkcije koje se odnose na identitet korisnika, autentifikaciju ili upravljanje sesijom nisu pravilno implementirane. Potencijalni uticaji ovih ranjivosti uključuju gubitak administrativnog pristupa, otkrivanje osetljivih informacija i izvođenje radnji u ime drugih korisnika.

Napadači koriste niz tehnika, uključujući iscrpni napad, napad na kredencijale, otimanje sesije, fiksiranje sesije, lažiranje zahteva s druge lokacije.

4.2.1. Eksploatacija slabosti

Sledi uvid u API sistem koji je podložan analiziranoj slabosti. Navedeni primeri služe kao ilustracija dela propusta koji se mogu javiti u implementaciji sistema, fokusirajući se konkretno na proces registracije i prijavljivanja na sistem.

Tokom registracije, korisnik je obavezan da unese različite informacije, uključujući i lozinku. Listing 5 prikazuje podatke koji se šalju prilikom registracije. Sa listinga, primećujemo upotrebu biblioteke "class-validator" koja sprovodi provere, da li su navedena polja prisutna u zahtevu (@IsNotEmpty()) i da li su vrednosti tekstualne (@IsString()).

```
import { IsEmail, IsNotEmpty, IsString } from "class-validator";
export class RegisterDto {
  @IsString()
  @IsNotEmpty()
  firstName: string;

  @IsString()
  @IsNotEmpty()
  lastName: string;
```

```
  @IsEmail()
  @IsNotEmpty()
  email: string;

  @IsString()
  @IsNotEmpty()
  password: string;
}
```

Listing 5. *Očekivani podaci prilikom registracije*

Međutim, značajan propust nastaje u nedostatku adekvatne validacije lozinke, što omogućava lošu praksu u vezi sa postavljanjem lozinki. Naime, trenutna implementacija ne postavlja dovoljno restriktivne zahteve za lozinke, što znači da lozinka može biti bilo šta, sa bilo kojim brojem karaktera i vrstom karaktera.

4.2.2. Mitigacija

Na listingu 6, prikazani su dekoratori @MinLength(), @MaxLength() i @Matches(). Korišćenjem ovih dekoratora biblioteke „class-validator“ postavljaju se specifična pravila za kreiranje lozinke, čime se osigurava da dužina lozinke bude između 12 i 16 karaktera, te da lozinka sadrži najmanje jedno veliko slovo i jedan specijalni karakter.

```
@IsString()
 @MinLength(12)
 @MaxLength(16)
 @Matches(/^(?=.*[A-Z])(?=.*[!@#$%^&*])/,
   { message: 'Password must contain at least one uppercase letter
and one special character.' })
 password: string;
```

Listing 6. *Validacija lozinke prilikom registracije*

4.3. A03:2023 - Broken Object Property Level Authorization (BOPLA)

BOPLA omogućava napadačima neovlašćen pristup specifičnim atributima objekta.

Neovlašćen pristup privatnim/osetljivim svojstvima objekata može dovesti do otkrivanja podataka, gubitka podataka ili oštećenja podataka. Pod određenim okolnostima, neovlašćen pristup svojstvima objekta može dovesti do eskalacije privilegija ili delimičnog/potpunog preuzimanja naloga [1].

4.3.1. Eksploatacija slabosti

U kontekstu analize ranjivosti, fokusiraćemo se na funkcionalnost prikaza proizvoda.

Svaki proizvod je opisan svojim karakteristikama, uključujući naziv, opis, fotografiju, kao i svojstvo isActive, koje pokazuje da li je proizvod aktivan ili ne. Međutim, ovo svojstvo predstavlja potencijalnu ranjivost jer krajnja tačka API-ja ne vrši adekvatnu verifikaciju pristupa korisnika ovom svojstvu. To znači da redovni korisnici mogu da pristupe i vide ovo svojstvo, iako u skladu sa funkcionalnim zahtevima, ne bi trebalo da mogu. Ova slabost bi mogla da dovede do zloupotrebe, gde bi napadač mogao da promeni vrednost svojstva isActive iz false u true koristeći krajnju tačku za ažuriranje proizvoda, čime bi otključao blokiran sadržaj.

Zbog toga je važno implementirati adekvatne mehanizme kontrole pristupa kako bi se osiguralo da samo ovlašćeni korisnici mogu da upravljaju ovim svojstvom i otključavaju blokiran sadržaj.

Listing 7 predstavlja funkcionalnost prikaza proizvoda. Treba primetiti da metoda getAll() direktno vraća sva svojsta proizvoda, ne uzimajući u obzir koji su to funkcionalni zahtevi za ovu krajnju tačku.

```
  async getAll(): Promise<Product[]> {
    return this.entityManager.createQueryBuilder(Product, 'p')
      .getMany();

  }
```

Listing 7. *Funkcionalnost prikaza proizvoda*

4.3.2. Mitigacija

Listing 8 prikazuje klasu koja sadrži samo ona svojsta objekta koja je neophodno vratiti, u skladu sa funkcionalnim zahtevima za ovu krajnju tačku.

```
export class ProductResult {
  id: string;
  name: string;
  description: string;
  image: string;
  price: number;
  createdBy: string;
}
```

Listing 8. *Svojstva objekta u skladu sa funkcionalnim zahtevima*

4.4. A04:2023 - Unrestricted Resource Consumption (URC)

URC nastaje kada aplikacija dozvoljava korisnicima (napadačima) da zloupotrebe resurse, poput CPU ili memorije, na način koji može dovesti do pada performansi ili čak do odbijanja usluge. Osnovna ideja iza URC je da napadač može izazvati potrošnju resursa izvan granica koje su normalno očekivane za funkcionalnost aplikacije. To može rezultirati smanjenjem dostupnosti aplikacije ili servisa za legitimne korisnike.

4.4.1. Eksploatacija slabosti

Zamislimo da zlonamerni korisnik kreira skriptu koja automatski šalje veliki broj zahteva za dodavanje fotografija. Ukoliko se API integriše sa AWS-om, a prostor za skladištenje fotografija se naplaćuje, zlonamerni korisnik može iskoristiti ovu slabost da nanese značajnu finansijsku štetu vlasnicima sistema**.**

4.4.2. Mitigacija

Prevencija uključuje ograničavanje potrošnje resursa po zahtevu, implementaciju ograničenja brzine i pravljenje adekvatnih kontrola pristupa kako bi se sprečila zloupotreba resursa od strane napadača. Na listingu 9 nalazi se pomoćna funkcija koja proverava da li korisnik ima dovoljno prostora u skladištu za izvršavanje akcije dodavanja fajla.

```
async hasEnoughSpace(em: EntityManager, fileSize: number):
Promise<boolean> {
    const storageInfo = await
QueryHelper.executeQueryFromFunction(
      em, 'get_storage_space_info', [this.id] );
    const necessarySpace = Number(storageInfo[0]['used_space']) +
fileSize;
    const allowedStorageSpace =
Number(this.storage_space[0]['allowed_space']);
    return necessarySpace < allowedStorageSpace; }
```

Listing 9. *Funkcija hasEnoughSpace*

Ova funkcija ima zadatak da proveri da li korisnik koji šalje zahtev za dodavanje fotografija raspolaže sa dovoljno prostora za skladištenje. Da bi ova provera bila moguća, bilo je neophodno proširiti tabelu User sa dodatnim kolonama storage_space i used_space.

4.5. A05:2023 - Broken Function Level Authorization (BFLA)

BFLA može nastati kada veb aplikacija ne proverava ispravno dozvole pristupa na nivou funkcija (ili radnji) koje korisnici mogu da izvrše.

4.5.1. Eksploatacija slabosti

Prilikom analize API-ja, fokusiramo se na specifičnu krajnju tačku POST /api/v1/product/update-availability i njenu implementaciju, prikazanu na listingu 10, gde je značajno primetiti odsustvo provera identiteta korisnika koji izvršava navedenu akciju.

```
  async updateAvailability(id: string, isActive: boolean): Promise<void>
{
    const product = await Product.findOne({ where: { id: id } });
    if (!product)
      throw new NotFoundException();
    product.is_active = isActive;
    await product.save(); }
```

Listing 10. *Akcija promene statusa proizvoda*

Svaki korisnik sistema koji uoči postojanje ove krajnje tačke može je pozvati i promeniti status objave/proizvoda, bez provere prava pristupa. Uprkos proveri postojanja samog proizvoda, nedostatak autentifikacije i autorizacije otvara vrata potencijalnim zloupotrebama, manipulacijama ili neovlaštenim izmenama statusa proizvoda.

4.5.2. Mitigacija

Identifikacija korisnika koji izvršava akciju i provera njegovih privilegija i ovlašćenja za izmenu statusa predstavljaju ključne elemente koje treba inkorporirati u kod kako bi se suzbila neovlašćena aktivnost.

Rešenje ovog problema je upotreba dekoratora *@Permission*. Potrebno je funkciju kontrolera označiti ovim dekoratorom, pri čemu je potrebno specificirati odgovarajuću permisiju koju korisnik koji poziva ovu akciju mora imati. Ovaj pristup omogućava efikasnu kontrolu pristupa i autorizaciju, ograničavajući izvršenje akcije na korisnike sa određenim privilegijama [5].

```
@Post('update-availability')
@Permissions(Permission.AdminProduct)
```

Listing 11. Dekorator *@Permission*

Sistem je potrebno konfigurisati tako da se prilikom svakog slanja API zahteva automatski poziva funkcija hasPermission(). Funkcija, zajedno sa stanjem table baze podataka UserRole, prikazani su na listingu 12 i 13.

```
  async hasPermission(permission: Permission) {
    const role = await this.user_role;
    const permissionBit =
role.permission_mask.charAt(role.permission_mask.length - 1 -
(permission - 1));
    return Boolean(Number(permissionBit)); }
```

Listing 12. *Funkcija hasPermissioin()*

```
INSERT INTO "public"."user_role"("id", "name", "permission_mask",
"description") VALUES (1, 'Admin',
'11111111111111111111111111111111', 'Global admin');
INSERT INTO "public"."user_role"("id", "name", "permission_mask",
"description") VALUES (2, 'Product admin',
'00000000000000000000000000000010', 'Product admin');
INSERT INTO "public"."user_role"("id", "name", "permission_mask",
"description") VALUES (5, 'User',
'00000000000000000000000000000000', 'Regular user');
```

Listing 13. Tablea *UserRole*

5. ZAKLJUČAK

Neprofitna organizacija OWASP redovno objavljuje listu od deset najčešćih slabosti aplikativnih programabilnih interfejsa (API). Cilj OWASP TOP 10 liste je podići svest o važnosti bezbednosti i pružiti smernice razvojnim timovima kako bi poboljšali bezbednost svojih aplikacija. Razumevanje OWASP Top 10 slabosti, prepoznavanje karakterističnih problema svake slabosti i razvijanje strategija za njihovu efikasnu neutralizaciju ključni su za bezbednost informacionih sistema. Cilj ovog rada je razvoj demo okruženja za praktično prikazivanje tih slabosti, odnosno prikaz optimalnih mera bezbednosti za otklanjanje istih.

Demo okruženje se sastoji od slabe i zaštićene verzije API-ja koji je razvijen korišćenjem okvira NodeJS. Odabran je zbog svoje popularnosti i jednostavnosti upotrebe. Svaka slabost je detaljno opisana, uključujući objašnjenje njenog značaja i potencijalne rizike za sigurnost sistema. Nakon identifikacije slabosti, prikazana je eksploatacija svake od njih. Za svaku identifikovanu slabost, razvijeni su primeri dobro napisanog koda koji demonstriraju implementaciju mitigacionih mera.

6. LITERATURA

[1] OWASP. (2023). OWASP Top 10 API Security Risks - 2023. Preuzeto iz https://owasp.org/API-Security/editions/2023/en/0x00-header/ (pristupljeno u februaru 2024.)

[2] NestJS. (2024). Dokumentacija. Preuzeto iz https://docs.nestjs.com/ (pristupljeno u februaru 2024.)

[3] PostreSQL. (2024, Februar 8). About. Preuzeto iz https://www.postgresql.org/about/ (pristupljeno u februaru 2024.)

[4] Postman. (2024). What is Postman? Preuzeto iz https://www.postman.com/product/what-is-postman/ (pristupljeno u februaru 2024.)

[5] NestJS. (2024). Dokumentacija. Preuzeto iz https://docs.nestjs.com/guards (pristupljeno u februaru 2024.)

Kratka biografija:

Milica Simeunović rođena je u Bijeljini 1997. god. Osnovu i srednju školu završila je u rodnom gradu. Diplomirala je 2021. godine na studijskog programu Primenjeno softversko inženjerstvo na Fakultetu tehničkih nauka u Novom Sadu.
kontakt: milica.simeunovic97@gmail.com