ZAŠTITA PODATAKA

Sigurnosni propusti u razvoju softvera

Sigurnosni propusti u razvoju softvera

- Razvoj bezbednog softvera podrazumeva pre svega uključivanje sigurnosnog aspekta u životni ciklus razvoja softvera
- Ipak, kako prilikom razvoja velikog broja aplikacija koje su već u produkciji nije vođeno računa o ovoj problematici, a nažalost i dalje se razvija softver tako da se ne obraća pažnja na sigurnost, značajna oblast postaje i pronalaženje sigurnosnih propusta u već razvijenim aplikacijama
- Dve relevantne organizacije koje objavljuju listu najzastupljenijih i najopasnijih sigurnosnih propusta su:
  - *SANS* (*Sysadmin, Audit, Network, Security*) institut i
  - *OWASP* (*Open Web Application Security Project*)

OWASP 2017. top 10

- Injekcija - *SQL, NoSQL, OS* i *LDAP* injekcija – podrazumeva slanje nepouzdanih podataka interpreteru kao deo komande ili upita;
- Nepravilno implementirana autentikacija – omogućava napadaču da preuzme nečiji identitet;
- Izloženost osetljivih podataka – *API* koji ne štiti podatke na pravi način;
- Eksterni entiteti u *XML* dokumentima – omogućavaju pristup internim fajlovima;
- Nepravilno implementirana kontrola pristupa – omogućava pristup nedozvoljenim funkcionalnostima;

OWASP 2017. top 10 (2)

- Neispravna sigurnosna konfiguracija – ako se ostave podrazumevana podešavanja koja nisu dovoljno bezbedna;
- Izvršavanje skripti između sajtova – omogućava izvršavanje zlonamernih skripti na sajtu žrtve;
- Nebezbedna deserijalizacija – često omogućava izvršavanje udaljenog koda;
- Korišćenje nebezbednih komponenti – komponente koje se koriste imaju iste privilegije kao i sama aplikacija;
- Nedovoljno beleženje i praćenje akcija korisnika – otežava detekciju zloupotrebe sigurnosnih propusta;

SQL injection

- napad u kom napadač podmeće zlonamerni SQL kod kroz predviđeni način za unos korisničkih podataka
- ako se ne proverava sadržaj korisničkog unosa i dinamički konstruiše SQL upit –zadaju se neželjene SQL komande
- izvršavanjem ovih komandi, napadač može:
  - zaobići predviđenu autentikaciju
  - izmeniti ili obrisati delove sadržaja baze podataka ili
  - preuzeti kontrolu nad serverom baze podataka

SQL injection – ilustracija napada

- aplikacija treba da uloguje korisnika

```
String username = request.getParameter("userName");
String password = request.getParameter("password");
query = "SELECT * FROM users"
        + " WHERE name = '" + username + "'"
        + " AND password = '" + password + "';"

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(query);
if (rs.next()) {
    LoginUserByID(rs.getInt("userID"));
}
```

- ako nema provere unosa korisnika

```
userName:   ' OR '1' = '1
password:   ' OR '1' = '1
```

- upit koji se izvrši

```
SELECT * FROM users
WHERE name = '' OR '1' = '1'
AND password = '' OR '1' = '1';
```

- vrati sve korisnike koji postoje u sistemu, uloguj prvog (često administrator)

SQL injection – ilustracija napada (2)

- ako aplikacija ispisuje rezultat – napadač dobija sadržaj tabele
- ako unese

```
userName:   '; DROP TABLE users; --
password:   doesn't matter
```

- izvršiće se

```
SELECT * FROM users WHERE name = ''; DROP TABLE users; --' WHERE...
```

- što bi obrisalo tabelu sa podacima o korisnicima

SQL injection – tipovi napada

- Tautologije
- Nelegalni upiti
- Upiti sa unijom
- Nadovezani upiti
- napadi zaključivanjem
- pretpostavka – logovanje, aplikacija ne vrši provere, upit

```
query = "SELECT * FROM users"
        + " WHERE name = '" + userName + "'"
        + " AND password = " + password + ";"
```

SQL injection – tautologije

- cilj - prevazilaženje autentikacije, pristup podacima
- unos takvih sekvenci, da uslov u WHERE klauzuli SQL upita bude ispunjen za sve redove tabele
- zahteva poznavanje konstrukcije logičkog izraza u WHERE klauzuli
- obično se izvodi dodavanjem operatora OR i uslova koji je uvek ispunjen
- posledice zavise od načina obrade rezultata upita
  - preduzima se akcija ukoliko rezultat upita sadrži bar jedan red - prevazilaženje autentikacije
  - ispišu se svi redovi dobijenog rezultata - neovlašćeni pristup podacima iz baze.
- prethodni primer

SQL injection – nelegalni upiti

- cilj - otkrivanje ranjivih parametara, otkrivanje šeme baze podataka, pristup podacima
- unos sekvenci koje će rezultovati greškom na serveru baze podataka
- prikaziuje se stranica za ispis podataka o nastalom problemu koja često može sadržati podatke o arhitekturi sistema koje korisnici ne bi trebalo da vide
- Tipično napadači izazivaju sledeće greške:
  - sintaksne greške - mogu otkriti ranjive parametre
  - greška pri konverziji tipova - mogu otkriti tipove kolona ili sam sadržaj pojedinih kolona
  - logičke greške - mogu otkriti strukturu baze podataka, imena tabela i kolona i njihove tipove

SQL injection – nelegalni upiti (2)

- primer

```
userName:  someone
password:  convert(int, (select top 1 name from sysobjects where xtype = 'u'))
```

- upit

```
SELECT * FROM users
WHERE name = 'someone'
AND password = convert(int, (select top 1 name from sysobjects
                             where xtype = 'u'));
```

- *ako je prva tabela Users* i koristi se Microsoft SQL Server
- dobija se poruka:
  - *"Conversion failed when converting the nvarchar value 'Users' to data type int."*
- saznajemo da se radi o SQL Server bazi i da se jedna od tabela naziva *Users*

SQL injection – upiti sa unijom

- cilj - pristup podacima, prevazilaženje autentikacije
- koristi se SQL operator *UNION*
- neophodno je poznavanje konstrukcije upita
- konstruiše se upit takav da se uz aplikacijom predviđeni rezultat *UNION* operatorom dodaje proizvoljni sadržaj
  - najčešće je to sadržaj pojedinih kolona neke druge tabele, za čije postojanje napadač zna
  - upravo zbog toga, ovom napadu najčešće prethodi serija napada *izazivanjem grešaka u sql upitu*, kako bi napadač otkrio šemu baze podataka.
- ukoliko aplikacija prikazuje rezultat upita, napadač na ovaj način može doći do podataka iz tabela koje aplikacija čak i ne koristi

SQL injection – upiti sa unijom (2)

- aplikacija prilikom logovanja izvrši upit nad tabelom sa korisnicima po nekim kriterijumima, pa zatim proverom da li je rezultat prazan ili ne odlučuje da li korisnik ima pravo pristupa
- u tom slučaju se dodavanjem nekih fiksnih vrednosti *UNION* operatorom, prevazilazi autentikacija
- ovakav način logovanja korisnika je prilično čest u internet aplikacijama

```
userName:  ' UNION SELECT cardNo from CreditCards where acctNo=10032; --
password:  doesn't matter
```

- upit

```
SELECT accounts FROM users
WHERE name = ''
UNION SELECT cardNo from CreditCards where acctNo=10032;
--' AND password = doesn't matter;
```

- napadač dolazi do broja kreditne kartice, iako aplikacijom to nije previđeno

SQL injection – nadovezani upiti

- cilj - pristup podacima, dodavanje, izmena ili brisanje podataka, izvođenje DoS napada
- napadač ne pokušava da izmeni predviđeni upit
- napadač uz predviđeni upit dodaje svoje SQL upite
- napadač ne mora u celosti da poznaje konstrukciju predviđenog upita - dovoljno je poznavanje jednog od parametara, koji se direktno ugrađuje u upit
- Moguće posledice
  - ukoliko aplikacija prikazuje rezultat upita, napadač na ovaj način može doći do podataka iz tabela koje aplikacija čak i ne koristi
  - dodavanjem *INSERT*, *UPDATE* ili *DELETE* naredbi, napadač može da menja zatečeno stanje u bazi podataka
  - ukoliko aplikacija nema ograničene privilegije nad serverom baze podataka, napadač može izvršiti neke sistemske komande na serveru i time izvršiti DoS napad

SQL injection – nadovezani upiti (2)

- unos

```
userName:  does not matter
password:  1; DROP TABLE users
```

- upit

```
SELECT * FROM users
WHERE name = 'does not matter'
AND password = 1;
DROP TABLE users;
```

- obrisani podaci o korisnicima – DoS napad

SQL injection – napadi zaključivanjem

- cilj - otkrivanje ranjivih parametara, otkrivanje šeme baze podataka, pristup podacima
- napadač ne može da dobije povratnu informaciju čak ni kada je SQL injection uspešno prosleđen do baze
- otkrivanje informacija o bazi postavljanjem jednog pitanja tipa true/false po napadu
- zahteva niz SQL injection napada - potrebno mnogo vremena za otkrivanje većeg skupa podataka
- Odgovore na postavljena true/false pitanja, napadač dobija indirektno, na jedan od sledeća dva načina:
  - Slepo podmetanje
  - Vremenski napadi

SQL injection – slepo podmetanje

- uz neki parametar ubacuje se SQL tautologiju koja je u jednom slučaju uvek tačna, a u drugom uvek netačna
- ukoliko aplikacija ne poseduje zaštitu od SQL injection napada, vrlo verovatno će preduzete akcije u ova dva slučaja biti različite i to će moći da se primeti
- unos

```
userName:   ' AND 1=0; --
password:   doesn't matter
```

- upit

```
SELECT * FROM users
WHERE name = '' AND 1=0;
--' AND password = doesn't matter;
```

- neuspešno logovanje

SQL injection – slepo podmetanje (2)

- zatim unos

```
userName:   ' OR 1=1; --
password:   doesn't matter
```

- upit

```
SELECT * FROM users
WHERE name = '' OR 1=1;
--' AND password = doesn't matter;
```

- ako se ponašanje u ova dva slučaja razlikuje – zaključak parametar userName je osetljiv

SQL injection – vremenski napadi

- ubacuje SQL kod koji predstavlja kontrolnu strukturu *IF*, sa odgovarajućim uslovom i telom
- uslov je proizvoljan i ako napad uspe, napadač dobija ishod tog uslova
- telo *IF* kontrolne strukture čini neka od naredbi čije trajanje predstavlja znatan udeo u odzivu aplikacije
  - najčešće je to naredba *WAITFOR DELAY*
- unos

```
userName:  '; IF ASCII(SUBSTRING((SELECT TOP 1 name FROM sysobjects WHERE
           xtype = 'u'),1,1)) > ASCII('X') WAITFOR DELAY '00:00:03'; --
password:  doesn't matter
```

- upit

```
SELECT * FROM users
WHERE name = '';
IF ASCII(SUBSTRING((SELECT TOP 1 name FROM sysobjects WHERE
   xtype = 'u'),1,1)) > ASCII('X') WAITFOR DELAY '00:00:03';
--' AND password = doesn't matter;
```

- da li je izvršen na osnovu dužine trajanja + informacija iz IF-a

SQL injection – dodatni rizici

- uskladištene procedure – iste opasnosti kao za upite generisane u aplikaciji

```
CREATE PROCEDURE dbo.VerifyUser
    @userName varchar(30),
    @pass varchar(30)
AS
BEGIN
    DECLARE @sql nvarchar(500);
    SET @sql = 'SELECT * FROM users
                WHERE name = ''' + @userName + '''
                AND password = ''' + @pass + ''' ';
    EXEC (@sql);
    IF @@ROWCOUNT > 0 RETURN 1
    ELSE RETURN 0
END
GO
```

- alternativni kodovi – zaobilaženje zaštite prepoznavanjem specijalnih karaktera i reči SQL jezika u unosu (npr. shutdown)

```
userName:  '; exec(char(0x73687574646f776e)) --
password:  doesn't matter
```

```
SELECT * FROM users
WHERE name = ''; exec(char(0x73687574646f776e))
--' AND password = doesn't matter;
```

SQL injection – metode zaštite

- Minimalne privilegije nad bazom podataka
  - privilegije aplikacije nad bazom podataka postave se na minimalne potrebne
  - sprečava izvršavanje sistemskih naredbi nad serverom baze podataka, kao i DDL naredbi
- Siguran način pisanja koda
  - skup saveta programerima
    - provera tipa - najčešće se ulazni podaci unose kao tekst (tip *String*) - treba u aplikaciji proveriti sve argumente koji su nekog posebnog tipa (*int*, *double*, *Date*), odnosno nisu tipa *String*
    - korišćenje sigurnih API-a - većina programskih jezika ima razvijene biblioteke koje se mogu nositi sa SQL injection napadom npr. Java JDBC Statement vs PreparedStatement (unos tretira kao parametar upita, a ne kao deo konstrukcije upita)
    - provera formata parametra - ukoliko se za neki od korisničkih unosa očekuje tačno definisan format, korisno je proveriti da li dobijeni unos odgovara očekivanom obrascu (npr. broj telefona)
    - neutralizovanje specijalnih SQL karaktera - za svaki korisnički unos neutrališu se specijalni SQL karakteri (‘, --, %, …) npr. Java Encoder

SQL injection – metode zaštite (2)

- korišćenje alata za statičku analizu koda
  - alati za statičku analizu koda, pokušavaju detektovati tok podataka u aplikaciji, koji potiče iz nekog ulaza aplikacije i završava u SQL upitu koji se šalje na izvršavanje
  - ulaz aplikacije je svako mesto na kom se prihvata korisnički unos
  - ukoliko postoje tokovi, koji prenose korisnički unos direktno do SQL upita, aplikacija je ranjiva na SQL injection napad
  - mana ovog metoda je što može detektovati samo poznate načine toka podataka

SQL injection – metode zaštite (3)

- korišćenje alata za dinamičku analizu
  - poseduju parser za programski jezik u kom se aplikacija razvija, ali i parser za sintaksu odgovarajuće SQL baze podataka
  - koristeći parsere naizmenično, alat prvo statičkom analizom koda nauči konstrukcije svih SQL upita koji se dinamički kreiraju u aplikaciji, a nakon toga, za vreme izvršavanja aplikacije, svi upiti koji se šalju na izvršavanje, pre prosleđivanja bazi, ponovo se interpretiraju od strane alata
  - ukoliko konstrukcija upita ne odgovara prethodno naučenoj, alat detektuje SQL injection napad i sprečava prosleđivanje zaraženog SQL koda bazi podataka
  - mana ovog metoda je to što produžava vreme izvršavanja

SQL injection – metode zaštite (4)

- korišćenje alata za slepo testiranje
  - alati koji simuliraju napade na aplikaciju i na taj način otkrivaju ranjivost
  - poseduju znanje o različitim tipovima i načinima izvođenja SQL injection napada
  - napadi se izvode nad pokrenutom aplikacijom koja se testira
  - najveća mana ovog metoda je što zavisi od ograničenog broja poznatih napada

XSS

- Dinamički sajtovi omogućavaju efektniji izgled, bolji UI i personalizovani prikaz informacija korisniku
- *XSS* (engl. *Cross Site Scripting*) je široko rasprostranjena ranjivost koda kod koje korisnik veruje da je povezan sa veb serverom od poverenja, dok simultano prosleđuje informacije i lažnom serveru
- Kao i kod sigurnosnih propusta umetanja, i ovde je zlonameran kod prosleđen na mestima gde se očekuje neka informacija od korisnika
- Aplikacija obrađuje ove podatke bez provere da li je u pitanju neka skripta ili *HTML* kod i smešta ih u odgovor koji šalje korisniku
- Pretraživač korisnika prikazuje odgovor koji je dobio od aplikacije i izvršava zlonameran kod

XSS (2)

- Još jedan način za umetanje zlonamernih skripti jeste kroz poruke u elektronskoj pošti ili u linkovima
- U trenutku otvaranja takve poruke ili klikom na link izvršava se zlonamerni kod
- Uspešno obavljeni *XSS* napadi mogu dovesti do dohvatanja informacija čuvanih u korisničkoj sesiji, kolačićima ili fajlovima i na taj način preuzeti poverljive informacije
- Moguće je prouzrokovati instaliranje virusa, izmenu sadržaja veb stranice tako da prikazuje lažne informacije, preusmeravanje korisničkih informacija na drugu lokaciju i niz drugih nepoželjnih aktivnosti

XSS (3)

- Postoje tri tipa *XSS* sigurnosnog propusta koji se razlikuju po tome kako se zlonameran kod ubacuje u stranicu i kada se on izvršava
  - Privremeni XSS
  - Stalni XSS
  - DOM baziran XSS

Privremeni XSS

- Ulazni podaci korisnika se obrađuju od strane veb servera i generiše se sledeća veb stranica za prikaz odgovora
  - ukoliko ti podaci nisu prošli kroz proces validacije i rezultujuća veb stranica je generisana bez enkodovanja sadržaja, na klijentskoj strani je po učitavanju stranice moguće izvršavanje skripti
  - ukoliko žele da iskoriste ovu ranjivost, napadači moraju navesti korisnika da ode na veb stranicu koja ima ugrađen izvršiv kod u *HTTP* odgovoru.
  - Kada se u pretraživaču otvori stranica vrši se dohvatanje parametara. Element sa oznakom *<script>* označava da ovaj sadržaj treba interpretirati kao skriptu i ona će se izvršiti.

Stalni XSS

- Razlika u odnosu na prethodno opisani privremeni *XSS* je mesto i vreme izvršavanja zlonamernog koda
  - Ulazni podaci korisnika se često ne vraćaju odmah u *HTTP* odgovoru korisniku, već se smeštaju u bazu podataka ili u fajl sistem na veb serveru
  - Ovaj sačuvani podatak se kasnije dohvata i prikazuje prilikom kreiranja stranice
  - Ovo je glavna snaga ovog napada
  - Svaki put kada korisnik zatraži stranicu unutar koje se ovaj kod ugrađuje napad će iznova biti izvršen
- Za razliku od privremenog *XSS* koji napada samo jednog korisnika, stalni *XSS* može pogoditi veći broj korisnika
- Jednom umetnut zlonameran kod, perzistentno sačuvan, slaće se svakom posetiocu ranjive stranice

DOM baziran XSS

- *DOM* (engl. *Document Object Model*) predstavlja programski interfejs ka *HTML* dokumentu
- Zajednička osobina privremenog i stalnog *XSS* je da se zlonamerni kod umeće na serverskoj strani dok se kod *DOM* baziranog tipa izvršava samo na klijentskoj strani
  - *JavaScript* kod, koji je ugnježden unutar *HTML* sadržaja veb stranice, izvršava se u pretraživaču
  - Tada se automatski kreira nekoliko objekata koji predstavljaju *DOM.*
  - Ako *JavaScript* kod ugnježden u stranici dohvati neki od DOM objekata i koristi je za generisanje novog *HTML* sadržaja koji se upisuje u istu veb stranicu, postoji mogućnost pojave *DOM* baziranog *XSS* sigurnosnog nedostatka
  - Izvršavanje ovog napada omogućava upravljanje nekim objektima koji se nalaze na korisnikovom disku

Metode zaštite

- izbegavanje posećivanja neproverenih linkova dobijenih putem elektronske pošte ili dobijenih na neproveren nacin
- zabrana izvršavanja skriptnih kodova u internet pretraživačima – stranica gubi neke funkcionalnosti
- jedna od osnovnih tehnika pomoću koje se programeri mogu zaštititi od *XSS* napada je konvertovanje specijalnih *HTML* karaktera u njihove tekstualne entitete - nakon ovako izvršenog konvertovanja *HTML* kod će sigurno biti ispisan onako kako je unet, pa tako skriptni kod neće biti izvršen